Formalizing MLTL in Isabelle/HOL

Zili Wang and Katherine Kosaian and Alec Rosentrater

March 17, 2025

Abstract

Building on the existing formalization of Mission-time Linear Temporal Logic (MLTL) [1], we formalize the notions of *language decomposition* and *language partition* for MLTL. More specifically, we formalize an algorithm to compute a language partition for MLTL and formally prove its correctness. Our algorithm is executable, and we export it Haskell via Isabelle/HOL's code generator.

Contents

theory *MLTL-Language-Partition-Algorithm*

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

begin

1 Extended MLTL Data Structure with Interval Compositions

Extended datatype that has an additional nat list associated with the temporal operators F, U, R to represent integer compositions of the interval

datatype (*atoms-mltl*: *'a*) *mltl-ext* =
True-mltl-ext (*True*$_c$)
| *False-mltl-ext* (*False*$_c$)
| *Prop-mltl-ext 'a* (*Prop*$_c$ *'(-')*)
| *Not-mltl-ext 'a mltl-ext* (*Not*$_c$ - [*85*] *85*)
| *And-mltl-ext 'a mltl-ext 'a mltl-ext* (- *And*$_c$ - [*82*, *82*] *81*)
| *Or-mltl-ext 'a mltl-ext 'a mltl-ext* (- *Or*$_c$ - [*81*, *81*] *80*)
| *Future-mltl-ext nat nat nat list 'a mltl-ext* (*F*$_c$ *'[-','-'] '<-'>* - [*88*, *88*, *88*, *88*] *87*)
| *Global-mltl-ext nat nat nat list 'a mltl-ext* (*G*$_c$ *'[-','-'] '<-'>* - [*88*, *88*, *88*, *88*] *87*)
| *Until-mltl-ext 'a mltl-ext nat nat nat list 'a mltl-ext* (- *U*$_c$ *'[-','-'] '<-'>* - [*84*, *84*, *84*, *84*] *83*)
| *Release-mltl-ext 'a mltl-ext nat nat nat list 'a mltl-ext* (- *R*$_c$ *'[-','-'] '<-'>* - [*84*, *84*, *84*, *84*] *83*)

Converts mltl ext formula to mltl by just dropping the nat list

fun *to-mltl*:: *'a mltl-ext* ⇒ *'a mltl* **where**
to-mltl True$_c$ = *True*$_m$
| *to-mltl False*$_c$ = *False*$_m$
| *to-mltl Prop*$_c$ (*p*) = *Prop*$_m$ (*p*)
| *to-mltl* (*Not*$_c$ *φ*) = *Not*$_m$ (*to-mltl φ*)
| *to-mltl* (*φ And*$_c$ *ψ*) = (*to-mltl φ*) *And*$_m$ (*to-mltl ψ*)
| *to-mltl* (*φ Or*$_c$ *ψ*) = (*to-mltl φ*) *Or*$_m$ (*to-mltl ψ*)
| *to-mltl* (*F*$_c$ [*a*,*b*] <*L*> *φ*) = (*F*$_m$ [*a*,*b*] (*to-mltl φ*))
| *to-mltl* (*G*$_c$ [*a*,*b*] <*L*> *φ*) = (*G*$_m$ [*a*,*b*] (*to-mltl φ*))
| *to-mltl* (*φ U*$_c$ [*a*,*b*] <*L*> *ψ*) = ((*to-mltl φ*) *U*$_m$ [*a*,*b*] (*to-mltl ψ*))
| *to-mltl* (*φ R*$_c$ [*a*,*b*] <*L*> *ψ*) = ((*to-mltl φ*) *R*$_m$ [*a*,*b*] (*to-mltl ψ*))

definition *semantics-mltl-ext*:: *'a set list ⇒ 'a mltl-ext ⇒ bool*
(- $\models_c$ - [*80*,*80*] *80*)
where $\pi \models_c \varphi = \pi \models_m$ (*to-mltl* φ)

definition *semantic-equiv-ext*:: *'a mltl-ext ⇒ 'a mltl-ext ⇒ bool*
(- $\equiv_c$ - [*80*, *80*] *80*)
where $\varphi \equiv_c \psi$ = (*to-mltl* φ) $\equiv_m$(*to-mltl* ψ)

definition *language-mltl-r* :: *'a mltl ⇒ nat ⇒ 'a set list set*
where *language-mltl-r* φ *r* =
{π. *semantics-mltl* π φ ∧ *length* $\pi \geq r$}

fun *convert-nnf-ext*:: *'a mltl-ext ⇒ 'a mltl-ext* **where**
convert-nnf-ext $True_c$ = $True_c$
| *convert-nnf-ext* $False_c$ = $False_c$
| *convert-nnf-ext* $Prop_c$ (*p*) = $Prop_c$ (*p*)
| *convert-nnf-ext* (φ And_c ψ) = ((*convert-nnf-ext* φ) And_c (*convert-nnf-ext* ψ))
| *convert-nnf-ext* (φ Or_c ψ) = ((*convert-nnf-ext* φ) Or_c (*convert-nnf-ext* ψ))
| *convert-nnf-ext* (F_c [*a*,*b*] <*L*> φ) = (F_c [*a*,*b*] <*L*> (*convert-nnf-ext* φ))
| *convert-nnf-ext* (G_c [*a*,*b*] <*L*> φ) = (G_c [*a*,*b*] <*L*> (*convert-nnf-ext* φ))
| *convert-nnf-ext* (φ U_c [*a*,*b*] <*L*> ψ) = ((*convert-nnf-ext* φ) U_c [*a*,*b*] <*L*> (*convert-nnf-ext* ψ))
| *convert-nnf-ext* (φ R_c [*a*,*b*] <*L*> ψ) = ((*convert-nnf-ext* φ) R_c [*a*,*b*] <*L*> (*convert-nnf-ext* ψ))

| *convert-nnf-ext* (Not_c $True_c$) = $False_c$
| *convert-nnf-ext* (Not_c $False_c$) = $True_c$
| *convert-nnf-ext* (Not_c $Prop_c$ (*p*)) = (Not_c $Prop_c$ (*p*))
| *convert-nnf-ext* (Not_c (Not_c φ)) = *convert-nnf-ext* φ
| *convert-nnf-ext* (Not_c (φ And_c ψ)) = ((*convert-nnf-ext* (Not_c φ)) Or_c (*convert-nnf-ext* (Not_c ψ)))
| *convert-nnf-ext* (Not_c (φ Or_c ψ)) = ((*convert-nnf-ext* (Not_c φ)) And_c (*convert-nnf-ext* (Not_c ψ)))
| *convert-nnf-ext* (Not_c (F_c [*a*,*b*] <*L*> φ)) = (G_c [*a*,*b*] <*L*> (*convert-nnf-ext* (Not_c φ)))
| *convert-nnf-ext* (Not_c (G_c [*a*,*b*] <*L*> φ)) = (F_c [*a*,*b*] <*L*> (*convert-nnf-ext* (Not_c φ)))
| *convert-nnf-ext* (Not_c (φ U_c [*a*,*b*] <*L*> ψ)) = ((*convert-nnf-ext* (Not_c φ)) R_c [*a*,*b*] <*L*> (*convert-nnf-ext* (Not_c ψ)))
| *convert-nnf-ext* (Not_c (φ R_c [*a*,*b*] <*L*> ψ)) = ((*convert-nnf-ext* (Not_c φ)) U_c [*a*,*b*] <*L*> (*convert-nnf-ext* (Not_c ψ)))

2 List Helper Functions and Properties

Computes the partial sum of the first i elements of list

definition *partial-sum* :: [*nat list*, *nat*] ⇒ *nat* **where**
partial-sum L i = *sum-list* (*take i L*)

Given interval start time a, and a list of ints L = [t1, t2, t3] Constructs

the list (of length 1 longer) of partial sums added to a: [a, a+t1, a+t1+t2, a+t1+t2+t3]

definition *interval-times* :: [*nat*, *nat list*] ⇒ *nat list* **where**
interval-times a L = *map* (λ*i*. *a* + *partial-sum L i*) [*0* ..< *length L* + *1*]

value *interval-times 3* [*1*, *2*, *3*, *4*, *5*] =
[*3*, *4*, *6*, *9*, *13*, *18*]

This function checks that L is a composition of n. A composition of an integer n is a way of writing n as the sum of a sequence of (strictly) positive integers

definition *is-composition* :: [*nat*, *nat list*] ⇒ *bool* **where**
is-composition n L = ((∀ *i*. *List.member L i* ⟶ *i* > *0*) ∧ (*sum-list L* = *n*))

Checks that every nat list in input of type mltl ext is a composition of its interval For example the formula F[2,7] has interval of length 7-2+1=6, and a valid composition would be L = [2, 3, 1]

fun *is-composition-MLTL*:: *'a mltl-ext* ⇒ *bool* **where**
is-composition-MLTL (*φ And*$_c$ *ψ*) = ((*is-composition-MLTL φ*) ∧ (*is-composition-MLTL ψ*))
| *is-composition-MLTL* (*φ Or*$_c$ *ψ*) = ((*is-composition-MLTL φ*) ∧ (*is-composition-MLTL ψ*))
| *is-composition-MLTL* (*G*$_c$[*a*,*b*] <*L*> *φ*) = ((*is-composition* (*b*−*a*+*1*) *L*) ∧ (*is-composition-MLTL φ*))
| *is-composition-MLTL* (*Not*$_c$ *φ*) = *is-composition-MLTL φ*
| *is-composition-MLTL* (*F*$_c$[*a*,*b*] <*L*> *φ*) = ((*is-composition* (*b*−*a*+*1*) *L*) ∧ (*is-composition-MLTL φ*))
| *is-composition-MLTL* (*φ U*$_c$[*a*,*b*] <*L*> *ψ*) = ((*is-composition* (*b*−*a*+*1*) *L*) ∧ (*is-composition-MLTL φ*) ∧ (*is-composition-MLTL ψ*))
| *is-composition-MLTL* (*φ R*$_c$[*a*,*b*] <*L*> *ψ*) = ((*is-composition* (*b*−*a*+*1*) *L*) ∧ (*is-composition-MLTL φ*) ∧ (*is-composition-MLTL ψ*))
| *is-composition-MLTL* - = *True*

definition *is-composition-allones*:: *nat* ⇒ *nat list* ⇒ *bool* **where**
is-composition-allones n L = ((*is-composition n L*) ∧ (∀ *i*<*length L*. *L*!*i* = *1*))

fun *is-composition-MLTL-allones*:: *'a mltl-ext* ⇒ *bool* **where**
is-composition-MLTL-allones (*φ And*$_c$ *ψ*) = ((*is-composition-MLTL-allones φ*) ∧ (*is-composition-MLTL-allones ψ*))
| *is-composition-MLTL-allones* (*φ Or*$_c$ *ψ*) = ((*is-composition-MLTL-allones φ*) ∧ (*is-composition-MLTL-allones ψ*))
| *is-composition-MLTL-allones* (*G*$_c$[*a*,*b*] <*L*> *φ*) = ((*is-composition-allones* (*b*−*a*+*1*) *L*) ∧ *is-composition-MLTL-allones φ*)
| *is-composition-MLTL-allones* (*Not*$_c$ *φ*) = *is-composition-MLTL-allones φ*
| *is-composition-MLTL-allones* (*F*$_c$[*a*,*b*] <*L*> *φ*) = ((*is-composition-allones* (*b*−*a*+*1*) *L*) ∧ (*is-composition-MLTL-allones φ*))
| *is-composition-MLTL-allones* (*φ U*$_c$[*a*,*b*] <*L*> *ψ*) = ((*is-composition-allones* (*b*−*a*+*1*) *L*) ∧ (*is-composition-MLTL-allones φ*) ∧ (*is-composition-MLTL-allones ψ*))

| *is-composition-MLTL-allones* (φ R_c[*a*,*b*] <*L*> ψ) = ((*is-composition-allones* (*b*−*a*+1) *L*) ∧ (*is-composition-MLTL-allones* φ) ∧ (*is-composition-MLTL-allones* ψ))
| *is-composition-MLTL-allones* - = *True*

3 Decomposition Function

fun *pairs* :: *'a list* ⇒ *'a list* ⇒ (*'a* × *'a*) *list* **where**
pairs [] *L2* = []
| *pairs* (*h1*#*T1*) *L2* = (*map* (λ*x*. (*h1*, *x*)) *L2*) @ (*pairs T1 L2*)

fun *And-mltl-list* :: *'a mltl-ext list* ⇒ *'a mltl-ext list* ⇒ *'a mltl-ext list* **where**
And-mltl-list D-φ *D-*ψ = *map* (λ*x*. *And-mltl-ext* (*fst x*) (*snd x*)) (*pairs D-*φ *D-*ψ)

fun *Global-mltl-list* :: *'a mltl-ext list* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ *'a mltl-ext list* **where**
Global-mltl-list D-φ *a b L* = *map* (λ*x*. *Global-mltl-ext a b L x*) *D-*φ

fun *Future-mltl-list* :: *'a mltl-ext list* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ *'a mltl-ext list* **where**
Future-mltl-list D-φ *a b L* = *map* (λ*x*. *Future-mltl-ext a b L x*) *D-*φ

fun *Until-mltl-list* :: *'a mltl-ext* ⇒ *'a mltl-ext list* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ *'a mltl-ext list* **where**
Until-mltl-list φ *D-*ψ *a b L* = *map* (λ*x*. *Until-mltl-ext* φ *a b L x*) *D-*ψ

fun *Release-mltl-list* :: *'a mltl-ext list* ⇒ *'a mltl-ext* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ *'a mltl-ext list* **where**
Release-mltl-list D-φ ψ *a b L* = *map* (λ*x*. *Release-mltl-ext x a b L* ψ) *D-*φ

fun *Mighty-Release-mltl-ext*:: *'a mltl-ext* ⇒ *'a mltl-ext* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ *'a mltl-ext*
where *Mighty-Release-mltl-ext x* ψ *a b L* =
(*And-mltl-ext* (*Release-mltl-ext x a b L* ψ)
(*Future-mltl-ext a b L x*))

fun *Mighty-Release-mltl-list* :: *'a mltl-ext list* ⇒ *'a mltl-ext* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ *'a mltl-ext list* **where**
Mighty-Release-mltl-list D-φ ψ *a b L* = *map* (λ*x*. *Mighty-Release-mltl-ext x* ψ *a b L*) *D-*φ

fun *Global-mltl-decomp* :: *'a mltl-ext list* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ *'a mltl-ext list* **where**
Global-mltl-decomp D-φ *a 0 L* = *Global-mltl-list D-*φ *a a* [*1*]
| *Global-mltl-decomp D-*φ *a len L* = *And-mltl-list* (*Global-mltl-decomp D-*φ *a* (*len*−*1*) *L*)
(*Global-mltl-list D-*φ (*a*+*len*) (*a*+*len*) [*1*])
value *Global-mltl-decomp* [*True-mltl-ext*, (*Prop-mltl-ext* (*0*::*nat*))] *0 2* [*3*] =
[(*G*$_c$ [*0*,*0*] <*1*> *True*$_c$ *And*$_c$ *G*$_c$ [*1*,*1*] <*1*> *True*$_c$) *And*$_c$ *G*$_c$ [*2*,*2*] <*1*> *True*$_c$,
(*G*$_c$ [*0*,*0*] <*1*> *True*$_c$ *And*$_c$ *G*$_c$ [*1*,*1*] <*1*> *True*$_c$) *And*$_c$ *G*$_c$ [*2*,*2*] <*1*>

$Prop_c\ (0)$,
$(G_c\ [0,0]\ <[1]>\ True_c\ And_c\ G_c\ [1,1]\ <[1]>\ Prop_c\ (0))\ And_c\ G_c\ [2,2]\ <[1]>\ True_c$,
$(G_c\ [0,0]\ <[1]>\ True_c\ And_c\ G_c\ [1,1]\ <[1]>\ Prop_c\ (0))\ And_c\ G_c\ [2,2]\ <[1]>\ Prop_c\ (0)$,
$(G_c\ [0,0]\ <[1]>\ Prop_c\ (0)\ And_c\ G_c\ [1,1]\ <[1]>\ True_c)\ And_c\ G_c\ [2,2]\ <[1]>\ True_c$,
$(G_c\ [0,0]\ <[1]>\ Prop_c\ (0)\ And_c\ G_c\ [1,1]\ <[1]>\ True_c)\ And_c\ G_c\ [2,2]\ <[1]>\ Prop_c\ (0)$,
$(G_c\ [0,0]\ <[1]>\ Prop_c\ (0)\ And_c\ G_c\ [1,1]\ <[1]>\ Prop_c\ (0))\ And_c\ G_c\ [2,2]\ <[1]>\ True_c$,
$(G_c\ [0,0]\ <[1]>\ Prop_c\ (0)\ And_c\ G_c\ [1,1]\ <[1]>\ Prop_c\ (0))\ And_c\ G_c\ [2,2]\ <[1]>\ Prop_c\ (0)]$

fun *LP-mltl-aux* :: *'a mltl-ext* ⇒ *nat* ⇒ *'a mltl-ext list* **where**
LP-mltl-aux φ *0* = $[\varphi]$
| *LP-mltl-aux* $True_c$ (*Suc k*) = $[True_c]$
| *LP-mltl-aux* $False_c$ (*Suc k*) = $[False_c]$
| *LP-mltl-aux* $Prop_c$ (*p*) (*Suc k*) = $[Prop_c\ (p)]$
| *LP-mltl-aux* $(Not_c\ (Prop_c\ (p)))$ (*Suc k*) = $[Not_c\ (Prop_c\ (p))]$
| *LP-mltl-aux* (φ And_c ψ) (*Suc k*)=
(*let D-*φ = (*LP-mltl-aux* (*convert-nnf-ext* φ) *k*) *in*
(*let D-*ψ = (*LP-mltl-aux* (*convert-nnf-ext* ψ) *k*) *in*
And-mltl-list D-φ *D-*ψ))
| *LP-mltl-aux* (φ Or_c ψ) (*Suc k*) =
(*let D-*φ = (*LP-mltl-aux* (*convert-nnf-ext* φ) *k*) *in*
(*let D-*ψ = (*LP-mltl-aux* (*convert-nnf-ext* ψ) *k*) *in*
(*And-mltl-list D-*φ *D-*ψ) @ (*And-mltl-list* $[Not_c\ \varphi]$ *D-*ψ) @
(*And-mltl-list D-*φ $[(Not_c\ \psi)]$))))
| *LP-mltl-aux* (G_c[*a*,*b*] <*L*> φ) (*Suc k*) =
(*let D-*φ = (*LP-mltl-aux* (*convert-nnf-ext* φ) *k*) *in*
(*if* (*length D-*$\varphi \leq 1$) *then* ($[G_c[a,b]\ <L>\ \varphi]$)
else (*Global-mltl-decomp D-*φ *a* (*b*−*a*) *L*)))
| *LP-mltl-aux* (F_c[*a*,*b*] <*L*> φ) (*Suc k*) =
(*let D-*φ = (*LP-mltl-aux* (*convert-nnf-ext* φ) *k*) *in*
(*let s* = *interval-times a L in*
(*Future-mltl-list D-*φ (*s*!*0*) ((*s*!*1*)−*1*) [(*s*!*1*)−(*s*!*0*)]) @ (*concat* (*map*
(λ*i*. (*And-mltl-list* [*Global-mltl-ext* (*s*!*0*) ((*s*!*i*)−*1*) [*s*!*i* − *s*!*0*] ($Not_c\ \varphi$)]
(*Future-mltl-list D-*φ (*s*!*i*) ((*s*!(*i*+*1*))−*1*) [*s*!(*i*+*1*)−(*s*!*i*)]))))
[*1* ..< *length L*])))))
| *LP-mltl-aux* (φ U_c[*a*,*b*] <*L*> ψ) (*Suc k*) =
(*let D-*ψ = (*LP-mltl-aux* (*convert-nnf-ext* ψ) *k*) *in*
(*let s* = *interval-times a L in*
(*Until-mltl-list* φ *D-*ψ (*s*!*0*) ((*s*!*1*)−*1*) [(*s*!*1*)−(*s*!*0*)]) @ (*concat* (*map*
(λ*i*. (*And-mltl-list* [*Global-mltl-ext* (*s*!*0*) ((*s*!*i*)−*1*) [*s*!*i* − *s*!*0*] (*And-mltl-ext* φ ($Not_c\ \psi$))]
(*Until-mltl-list* φ *D-*ψ (*s*!*i*) ((*s*!(*i*+*1*)−*1*)) [*s*!(*i*+*1*)−(*s*!*i*)])))
[*1* ..< *length L*])))))
| *LP-mltl-aux* (φ R_c[*a*,*b*] <*L*> ψ) (*Suc k*) =

```
  (let D-φ = (LP-mltl-aux (convert-nnf-ext φ) k) in
  (let s = interval-times a L in
   [Global-mltl-ext a b L ((Not_c φ) And_c ψ)] @
  (Mighty-Release-mltl-list D-φ ψ (s!0) ((s!1)−1) [(s!1)−(s!0)]) @ (concat (map
   (λi. (And-mltl-list [Global-mltl-ext (s!0) ((s!i)−1) [s!i − s!0] ((Not_c φ) And_c
ψ)]
            (Mighty-Release-mltl-list D-φ ψ (s!i) ((s!(i+1)−1)) [s!(i+1)−(s!i)])))
   [1 ..< length L])))))
| LP-mltl-aux - - = []

fun LP-mltl :: 'a mltl-ext ⇒ nat ⇒ 'a mltl list where
LP-mltl φ k = map (λx. to-mltl x)
(map (λx. convert-nnf-ext x) (LP-mltl-aux (convert-nnf-ext φ) k))
```

3.1 Examples

```
value LP-mltl-aux (F_c[0,9] <[3, 3, 3]> ((Prop_c (0::nat)) Or_c (Prop_c (1::nat))))
1 =
[F_c [0,2] <[3]> (Prop_c (0) Or_c Prop_c (1)),
 G_c [0,2] <[3]> (Not_c (Prop_c (0) Or_c Prop_c (1))) And_c F_c [3,5] <[3]> (Prop_c
(0) Or_c Prop_c (1)),
 G_c [0,5] <[6]> (Not_c (Prop_c (0) Or_c Prop_c (1))) And_c F_c [6,8] <[3]> (Prop_c
(0) Or_c Prop_c (1))]

value LP-mltl (True_c Or_c (Prop_c (0::nat))) 1 =
[True_m And_m Prop_m (0), False_m And_m Prop_m (0), True_m And_m Not_m Prop_m
(0)]

value LP-mltl ((Prop_c (0::nat)) U_c [2,5] <[4]> (Prop_c (1))) 1 =
      [Prop_m (0) U_m [2,5] Prop_m (1)]

value LP-mltl ((Prop_c (0::nat)) R_c[2,5] <[2, 2]> (Prop_c (1))) 1 =
[G_m [2,5] (Not_m Prop_m (0) And_m Prop_m (1)),
  Prop_m (0) R_m [2,3] Prop_m (1) And_m F_m [2,3] Prop_m (0),
  G_m [2,3] (Not_m Prop_m (0) And_m Prop_m (1)) And_m (Prop_m (0) R_m [4,5] Prop_m
(1) And_m F_m [4,5] Prop_m (0))]

value LP-mltl ((F_c[0,3] <[1,1,1,1]> (Prop_c (0::nat))) Or_c
            (G_c[0,3] <[1,1,1,1]> (Prop_c (1)))) 3 =
[F_m [0,0] Prop_m (0) And_m G_m [0,3] Prop_m (1),
  (G_m [0,0] (Not_m Prop_m (0)) And_m F_m [1,1] Prop_m (0)) And_m G_m [0,3] Prop_m
(1),
  (G_m [0,1] (Not_m Prop_m (0)) And_m F_m [2,2] Prop_m (0)) And_m G_m [0,3] Prop_m
(1),
  (G_m [0,2] (Not_m Prop_m (0)) And_m F_m [3,3] Prop_m (0)) And_m G_m [0,3] Prop_m
(1),
  G_m [0,3] (Not_m Prop_m (0)) And_m G_m [0,3] Prop_m (1),
  F_m [0,0] Prop_m (0) And_m F_m [0,3] (Not_m Prop_m (1)),
  (G_m [0,0] (Not_m Prop_m (0)) And_m F_m [1,1] Prop_m (0)) And_m F_m [0,3] (Not_m
```

$Prop_m\ (1)$),
($G_m\ [0,1]\ (Not_m\ Prop_m\ (0))\ And_m\ F_m\ [2,2]\ Prop_m\ (0))\ And_m\ F_m\ [0,3]\ (Not_m\ Prop_m\ (1))$,
($G_m\ [0,2]\ (Not_m\ Prop_m\ (0))\ And_m\ F_m\ [3,3]\ Prop_m\ (0))\ And_m\ F_m\ [0,3]\ (Not_m\ Prop_m\ (1))$]

end
theory *MLTL-Language-Partition-Proof*

imports *MLTL-Language-Partition-Algorithm*

begin

4 Properties of convert nnf ext

lemma *convert-nnf-and-convert-nnf-ext*:
shows *to-mltl* (*convert-nnf-ext* φ) =
convert-nnf (*to-mltl* φ)
proof (*induct depth-mltl* (*to-mltl* φ) *arbitrary*: φ *rule*: *less-induct*)
case *less*
have *not*: ($\bigwedge\varphi$. *depth-mltl* (*to-mltl* φ)
< *Suc* (*depth-mltl* (*to-mltl* ψ)) $\Longrightarrow$
to-mltl (*convert-nnf-ext* φ) =
convert-nnf (*to-mltl* φ)) $\Longrightarrow$
$\varphi = Not_c\ \psi \Longrightarrow$
to-mltl (*convert-nnf-ext* ($Not_c\ \psi$)) =
convert-nnf (Not_m (*to-mltl* ψ)) **for** ψ
proof−
assume *ih*: ($\bigwedge\varphi$. *depth-mltl* (*to-mltl* φ)
< *Suc* (*depth-mltl* (*to-mltl* ψ)) $\Longrightarrow$
to-mltl (*convert-nnf-ext* φ) =
convert-nnf (*to-mltl* φ))
assume *shape*: $\varphi = Not_c\ \psi$
show *?thesis*
using *less ih shape* **by** (*induct* ψ) *simp-all*
qed
show *?case* **using** *less not*
by(*cases* φ) *auto*
qed

lemma *convert-nnf-ext-to-mltl-commute*:
shows (*convert-nnf* (*to-mltl* φ)) = (*to-mltl* (*convert-nnf-ext* φ))
proof(*induct depth-mltl* (*to-mltl* φ) *arbitrary*: φ *rule*: *less-induct*)
case *less*
then show *?case*
proof (*cases* φ)
case *True-mltl-ext*
then show *?thesis*

unfolding *True-mltl-ext convert-nnf.simps convert-nnf-ext.simps to-mltl.simps semantic-equiv-def*
by *simp*
next
case *False-mltl-ext*
then show *?thesis*
unfolding *False-mltl-ext convert-nnf.simps convert-nnf-ext.simps to-mltl.simps semantic-equiv-def*
by *simp*
next
case (*Prop-mltl-ext p*)
then show *?thesis*
unfolding *Prop-mltl-ext convert-nnf.simps convert-nnf-ext.simps to-mltl.simps semantic-equiv-def*
by *simp*
next
case (*Not-mltl-ext F*)
then have *φ-is*: $\varphi = Not_c\ F$
by *blast*
show *?thesis*
proof(*cases F*)
case *True-mltl-ext*
then show *?thesis* **using** *φ-is less semantic-equiv-def* **by** *auto*
next
case *False-mltl-ext*
then show *?thesis* **using** *φ-is less semantic-equiv-def* **by** *auto*
next
case (*Prop-mltl-ext p*)
then show *?thesis* **using** *φ-is less semantic-equiv-def* **by** *auto*
next
case (*Not-mltl-ext F1*)
then show *?thesis* **using** *φ-is less semantic-equiv-def* **by** *auto*
next
case (*And-mltl-ext F1 F2*)
have *r1*: Not_m (*to-mltl F1*) = *to-mltl* (Not_c *F1*)
by *simp*
have *r2*: Not_m (*to-mltl F2*) = *to-mltl* (Not_c *F2*)
by *simp*
have *rewrite*: (*Or-mltl* (*convert-nnf* (Not_m (*to-mltl F1*)))
(*convert-nnf* (Not_m (*to-mltl F2*)))) =
(*Or-mltl* (*convert-nnf* (*to-mltl* (Not_c *F1*)))
(*convert-nnf* (*to-mltl* (Not_c *F2*))))
using *r1 r2* **by** *simp*
have *ih1*: (*convert-nnf* (*to-mltl* (Not_c *F1*))) =
(*to-mltl* (*convert-nnf-ext* (Not_c *F1*)))
using *less*[*of* Not_c *F1*] **unfolding** *And-mltl-ext φ-is* **by** *simp*
have *ih2*: (*convert-nnf* (*to-mltl* (Not_c *F2*))) =
(*to-mltl* (*convert-nnf-ext* (Not_c *F2*)))
using *less*[*of* Not_c *F2*] **unfolding** *And-mltl-ext φ-is* **by** *simp*

```
  have (Or-mltl (convert-nnf (to-mltl (Not_c F1)))
              (convert-nnf (to-mltl (Not_c F2))))
 = (Or-mltl (to-mltl (convert-nnf-ext (Not_c F1)))
   (to-mltl (convert-nnf-ext (Not_c F2))))
    using ih1 ih2 unfolding semantic-equiv-def by auto
  then show ?thesis
    unfolding φ-is And-mltl-ext to-mltl.simps convert-nnf.simps
    unfolding convert-nnf-ext.simps to-mltl.simps
    by simp
next
  case (Or-mltl-ext F1 F2)
  have r1: Not_m (to-mltl F1) = to-mltl (Not_c F1)
    by simp
  have r2: Not_m (to-mltl F2) = to-mltl (Not_c F2)
    by simp
  have rewrite: (Or-mltl (convert-nnf (Not_m (to-mltl F1)))
                       (convert-nnf (Not_m (to-mltl F2)))) =
      (Or-mltl (convert-nnf (to-mltl (Not_c F1)))
              (convert-nnf (to-mltl (Not_c F2))))
    using r1 r2 by simp
  have ih1: (convert-nnf (to-mltl (Not_c F1))) =
                     (to-mltl (convert-nnf-ext (Not_c F1)))
    using less[of Not_c F1] unfolding Or-mltl-ext φ-is by simp
  have ih2: (convert-nnf (to-mltl (Not_c F2))) =
                     (to-mltl (convert-nnf-ext (Not_c F2)))
    using less[of Not_c F2] unfolding Or-mltl-ext φ-is by simp
  have
 (And-mltl (convert-nnf (to-mltl (Not_c F1)))
              (convert-nnf (to-mltl (Not_c F2)))) =
 (And-mltl (to-mltl (convert-nnf-ext (Not_c F1)))
   (to-mltl (convert-nnf-ext (Not_c F2))))
    using ih1 ih2 unfolding semantic-equiv-def by auto
  then show ?thesis
    unfolding φ-is Or-mltl-ext to-mltl.simps convert-nnf.simps
    unfolding convert-nnf-ext.simps to-mltl.simps
    by blast
next
  case (Future-mltl-ext a b L F)
  have r1: Not_m (to-mltl F) = to-mltl (Not_c F)
    by simp
  then have rewrite: (Global-mltl a b (convert-nnf (Not_m (to-mltl F)))) =
            (Global-mltl a b (convert-nnf (to-mltl (Not_c F))))
    by simp
  have ih: (convert-nnf (to-mltl (Not_c F))) =
                       (to-mltl (convert-nnf-ext (Not_c F)))
    using less[of Not_c F] φ-is unfolding Future-mltl-ext by simp
  have (Global-mltl a b (convert-nnf (to-mltl (Not_c F)))) =
 (Global-mltl a b (to-mltl (convert-nnf-ext (Not_c F))))
    using ih unfolding semantic-equiv-def by auto
```

```
  then show ?thesis
    unfolding φ-is Future-mltl-ext to-mltl.simps convert-nnf.simps
    unfolding convert-nnf-ext.simps to-mltl.simps
    using rewrite by blast
next
  case (Global-mltl-ext a b L F)
  have r1: Not_m (to-mltl F) = to-mltl (Not_c F)
    by simp
  then have rewrite: (Global-mltl a b (convert-nnf (Not_m (to-mltl F)))) =
            (Global-mltl a b (convert-nnf (to-mltl (Not_c F))))
    by simp
  have ih: (convert-nnf (to-mltl (Not_c F))) =
                        (to-mltl (convert-nnf-ext (Not_c F)))
    using less[of Not_c F] φ-is unfolding Global-mltl-ext by simp
  have (Future-mltl a b (convert-nnf (to-mltl (Not_c F)))) =
 (Future-mltl a b (to-mltl (convert-nnf-ext (Not_c F))))
    using ih unfolding semantic-equiv-def by auto
  then show ?thesis
    unfolding φ-is Global-mltl-ext to-mltl.simps convert-nnf.simps
    unfolding convert-nnf-ext.simps to-mltl.simps
    using rewrite by simp
next
  case (Until-mltl-ext F1 a b L F2)
  have r1: Not_m (to-mltl F1) = to-mltl (Not_c F1)
    by simp
  have r2: Not_m (to-mltl F2) = to-mltl (Not_c F2)
    by simp
  have rewrite: (Or-mltl (convert-nnf (Not_m (to-mltl F1)))
                         (convert-nnf (Not_m (to-mltl F2)))) =
      (Or-mltl (convert-nnf (to-mltl (Not_c F1)))
               (convert-nnf (to-mltl (Not_c F2))))
    using r1 r2 by simp
  have ih1: (convert-nnf (to-mltl (Not_c F1))) =
                      (to-mltl (convert-nnf-ext (Not_c F1)))
    using less[of Not_c F1] unfolding Until-mltl-ext φ-is by simp
  have ih2: (convert-nnf (to-mltl (Not_c F2))) =
                      (to-mltl (convert-nnf-ext (Not_c F2)))
    using less[of Not_c F2] unfolding Until-mltl-ext φ-is by simp
  have
 (Release-mltl (convert-nnf (to-mltl (Not_c F1))) a b
               (convert-nnf (to-mltl (Not_c F2)))) =
 (Release-mltl (to-mltl (convert-nnf-ext (Not_c F1))) a b
   (to-mltl (convert-nnf-ext (Not_c F2))))
    using ih1 ih2 unfolding semantic-equiv-def by auto
  then show ?thesis
    unfolding φ-is Until-mltl-ext to-mltl.simps convert-nnf.simps
    unfolding convert-nnf-ext.simps to-mltl.simps
    by blast
next
```

```
    case (Release-mltl-ext F1 a b L F2)
    have r1: Not_m (to-mltl F1) = to-mltl (Not_c F1)
      by simp
    have r2: Not_m (to-mltl F2) = to-mltl (Not_c F2)
      by simp
    have rewrite: (Or-mltl (convert-nnf (Not_m (to-mltl F1)))
                           (convert-nnf (Not_m (to-mltl F2)))) =
       (Or-mltl (convert-nnf (to-mltl (Not_c F1)))
                (convert-nnf (to-mltl (Not_c F2))))
      using r1 r2 by simp
    have ih1: (convert-nnf (to-mltl (Not_c F1))) =
                        (to-mltl (convert-nnf-ext (Not_c F1)))
      using less[of Not_c F1] unfolding Release-mltl-ext φ-is by simp
    have ih2: (convert-nnf (to-mltl (Not_c F2))) =
                        (to-mltl (convert-nnf-ext (Not_c F2)))
      using less[of Not_c F2] unfolding Release-mltl-ext φ-is by simp
    have
   (Until-mltl (convert-nnf (to-mltl (Not_c F1))) a b
               (convert-nnf (to-mltl (Not_c F2)))) =
   (Until-mltl (to-mltl (convert-nnf-ext (Not_c F1))) a b
     (to-mltl (convert-nnf-ext (Not_c F2))))
      using ih1 ih2 unfolding semantic-equiv-def by auto
    then show ?thesis
      unfolding φ-is Release-mltl-ext to-mltl.simps convert-nnf.simps
      unfolding convert-nnf-ext.simps to-mltl.simps
      by blast
  qed
 next
  case (And-mltl-ext F1 F2)
  show ?thesis
   unfolding And-mltl-ext to-mltl.simps convert-nnf.simps convert-nnf-ext.simps
semantic-equiv-def
     using less[of F1] less[of F2] And-mltl-ext unfolding semantics-mltl.simps
semantic-equiv-def by auto
 next
  case (Or-mltl-ext F1 F2)
  then show ?thesis
    unfolding Or-mltl-ext to-mltl.simps convert-nnf.simps convert-nnf-ext.simps
semantic-equiv-def
      using less[of F1] less[of F2] Or-mltl-ext unfolding semantics-mltl.simps
semantic-equiv-def by simp
 next
  case (Future-mltl-ext a b L F)
  show ?thesis
   unfolding Future-mltl-ext to-mltl.simps convert-nnf.simps convert-nnf-ext.simps
to-mltl.simps
   using less[of F] Future-mltl-ext unfolding semantic-equiv-def semantics-mltl.simps
by simp
 next
```

case (*Global-mltl-ext a b L F*)
then show *?thesis*
unfolding *Global-mltl-ext to-mltl.simps convert-nnf.simps convert-nnf-ext.simps to-mltl.simps*
using *less*[*of F*] *Global-mltl-ext* **unfolding** *semantic-equiv-def semantics-mltl.simps* **by** *simp*
next
case (*Until-mltl-ext F1 a b L F2*)
then show *?thesis*
unfolding *Until-mltl-ext to-mltl.simps convert-nnf.simps convert-nnf-ext.simps to-mltl.simps*
using *less*[*of F1*] *less*[*of F2*] *Until-mltl-ext* **unfolding** *semantic-equiv-def semantics-mltl.simps* **by** *simp*
next
case (*Release-mltl-ext F1 a b L F2*)
then show *?thesis*
unfolding *Release-mltl-ext to-mltl.simps convert-nnf.simps convert-nnf-ext.simps to-mltl.simps*
using *less*[*of F1*] *less*[*of F2*] *Release-mltl-ext* **unfolding** *semantic-equiv-def semantics-mltl.simps* **by** *simp*
qed
qed

lemma *convert-nnf-ext-preserves-semantics*:
assumes *intervals-welldef* (*to-mltl* φ)
shows (*convert-nnf-ext* φ) $\equiv_c$ φ
proof−
have *semantic-equiv* (*convert-nnf* (*to-mltl* φ)) (*to-mltl* φ)
using *assms convert-nnf-preserves-semantics*[*of* (*to-mltl* φ)]
unfolding *semantic-equiv-ext-def semantic-equiv-def* **by** *blast*
then show *?thesis*
using *convert-nnf-ext-to-mltl-commute*
unfolding *semantic-equiv-ext-def semantic-equiv-def* **by** *metis*
qed

lemma *convert-nnf-ext-convert-nnf-ext*:
shows *convert-nnf-ext* φ = *convert-nnf-ext* (*convert-nnf-ext* φ)
proof(*induction depth-mltl* (*to-mltl* φ) *arbitrary*: φ *rule*: *less-induct*)
case *less*
have *not-case*: ($\bigwedge F$. *depth-mltl* (*to-mltl F*) <
Suc (*depth-mltl* (*to-mltl G*)) $\Longrightarrow$
convert-nnf-ext (*convert-nnf-ext F*) = *convert-nnf-ext F*) $\Longrightarrow$
φ = Not_c *G* $\Longrightarrow$
convert-nnf-ext (*convert-nnf-ext* (Not_c *G*)) =
convert-nnf-ext (Not_c *G*) **for** *G*
proof −
assume *ind-h*: ($\bigwedge F$. *depth-mltl* (*to-mltl F*) <
Suc (*depth-mltl* (*to-mltl G*)) $\Longrightarrow$

convert-nnf-ext (*convert-nnf-ext F*) = *convert-nnf-ext F*)
assume *φ-is*: $\varphi = Not_c\ G$
show *?thesis* **using** *less φ-is* **by** (*cases G*) *simp-all*
qed
show *?case* **using** *less not-case*
by (*cases φ*) *fastforce+*
qed

4.1 Cases where to mltl is bijective

lemma *to-mltl-true-bijective*:
assumes *to-mltl* $\varphi = True_m$
shows $\varphi = True_c$
using *assms* **by** (*cases φ*) *simp-all*

lemma *to-mltl-false-bijective*:
assumes *to-mltl* $\varphi = False_m$
shows $\varphi = False_c$
using *assms* **by** (*cases φ*) *simp-all*

lemma *to-mltl-prop-bijective*:
assumes *to-mltl* $\varphi = Prop_m\ (p)$
shows $\varphi = Prop_c\ (p)$
using *assms* **by** (*cases φ*) *simp-all*

lemma *to-mltl-not-prop-bijective*:
assumes *to-mltl* $\varphi = Not_m\ (Prop_m\ (p))$
shows $\varphi = Not_c\ (Prop_c\ (p))$
using *assms* **by** (*cases φ*) (*simp-all add*: *to-mltl-prop-bijective*)

5 Lemmas about Integer Composition

lemma *composition-length-ub*:
fixes *n*::*nat* **and** *L*::*nat list*
assumes *is-composition n L*
shows *length L* $\leq n$
using *assms* **unfolding** *is-composition-def*
proof (*induct L arbitrary*: *n*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a L*)
have *listsum*: *sum-list* (*a* # *L*) = *a* + *sum-list L*
by *simp*
then have *ls-L*: *sum-list L* = *n* − *a*
using *Cons*(*2*) **by** *auto*
then have *Lprop*: (∀ *i*. *List.member L i* ⟶ *0* < *i*) ∧ *sum-list L* = *n* − *a*
using *Cons*(*2*)
by (*meson member-rec*(*1*))

```
  then have len-leq: length L ≤ n − a
    using Cons(1)[OF Lprop]
    by auto
  have a > 0
    using Cons(2)
    by (meson member-rec(1))
  then show ?case using len-leq
    using Cons.prems listsum by auto
qed

lemma composition-length-lb:
  fixes n::nat and L::nat list
  assumes is-composition n L
  assumes n > 0
  shows 0 < length L
proof−
  have ¬(0 < length L) ⟹ False
  proof−
    assume ¬(0 < length L)
    then have length L = 0
      by simp
    then have sum-list L = 0
      by simp
    then show ?thesis
      using assms unfolding is-composition-def
      by simp
  qed
  then show ?thesis using assms by blast
qed


lemma interval-times-length:
  fixes a::nat and L::nat list
  shows length (interval-times a L) = length L + 1
  unfolding interval-times-def by auto


lemma interval-times-first:
  fixes a::nat and L::nat list
  shows (interval-times a L)!0 = a
proof−
  have map (λi. a + partial-sum L i) [0..<length L + 1] ! 0 =
       (λi. a + partial-sum L i) 0
   by (metis Nat.add-0-right add-gr-0 less-numeral-extra(1) nth-map-upt zero-less-diff)

  then have map (λi. a + partial-sum L i) [0..<length L + 1] ! 0 = a
    unfolding partial-sum-def by auto
  then show ?thesis
    unfolding interval-times-def by blast
```

qed

lemma *interval-times-last*:
fixes *a b*::*nat* **and** *L*::*nat list*
assumes *int-welldef*: $a \leq b$
assumes *composition*: *is-composition* $(b-a+1)$ *L*
shows (*interval-times a L*)!(*length L*) = *b+1*
proof −
have *partial-sum L* (*length L*) = *sum-list L*
unfolding *partial-sum-def* **by** *auto*
then have *a* + *partial-sum L* (*length L*) = *b+1*
using *assms* **unfolding** *is-composition-def*
by *simp*
then show *?thesis*
unfolding *interval-times-def*
by (*metis add-0 add-diff-cancel-left′ less-add-one nth-map-upt*)
qed

lemma *interval-times-diff*:
fixes *a b i*::*nat* **and** *L*::*nat list*
assumes *int-welldef*: $a \leq b$
assumes *composition*: *is-composition* $(b-a+1)$ *L*
assumes *i-index*: $i <$ *length L*
assumes *s-is*: *s* = *interval-times a L*
shows *s*!$(i+1)$ − *s*!(i) = *L*!*i*
proof−
have *ip1*: *s* ! $(i+1)$ = *a* + *partial-sum L* $(i+1)$
using *s-is i-index* **unfolding** *interval-times-def*
by (*metis* (*no-types*, *lifting*) *add-0 add-mono1 diff-zero nth-map-upt*)
have *i*: *s* ! *i* = *a* + *partial-sum L i*
using *s-is i-index* **unfolding** *interval-times-def*
by (*metis* (*no-types*, *lifting*) *add.commute add-0 add-strict-increasing diff-zero less-numeral-extra*(*1*) *less-or-eq-imp-le nth-map-upt*)
have *s-iat*: *s* ! $(i+1)$ − *s* ! *i* = *partial-sum L* $(i+1)$ − *partial-sum L i*
using *ip1 i*
by *auto*
have *take-is*: *take* $(i+1)$ *L* = (*take i L*) @ [*L* ! *i*]
by (*simp add*: *i-index take-Suc-conv-app-nth*)
have *li*: *foldr* (+) [*L* ! *i*] *0* = *L* ! *i*
by *force*
have $\bigwedge$*a*::*nat*. *foldr* (+) *L a* = *a* + *foldr* (+) *L 0* **for** *L*::*nat list*
proof (*induct L*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons h T*)
then show *?case*
by (*metis add.left-commute foldr.simps*(*2*) *o-apply*)

```
  qed
  then have foldr (+) (take i L) (L!i) = L ! i + foldr (+) (take i L) 0
    by blast
  then have foldr (+) ((take i L) @ [L ! i]) 0 − foldr (+) (take i L) 0 = L ! i
    using  foldr-append[of (+) take i L [L ! i] 0] li
    by simp
  then have sum-list (take (i + 1) L) − sum-list (take i L) =  L ! i
    using i-index take-is by simp
    then show ?thesis
    using i-index composition unfolding is-composition-def
    partial-sum-def s-iat by blast
qed

lemma interval-times-diff-ge:
  fixes a b i::nat and L::nat list
  assumes int-welldef: a ≤ b
  assumes composition: is-composition (b−a+1) L
  assumes i-index: i < length L
  assumes s-is: s = interval-times a L
  shows s!(i+1) > s!(i)
proof−
  have diff: s!(i+1) − s!(i) = L!i
    using assms interval-times-diff by blast
  have gap: L!i > 0 using assms(2) unfolding is-composition-def
    by (meson i-index in-set-member nth-mem)
  show ?thesis using diff gap by simp
qed

lemma interval-times-diff-ge-general:
  fixes a b i j::nat and L::nat list
  assumes int-welldef: a ≤ b
  assumes composition: is-composition (b−a+1) L
  assumes j-index: j ≤ length L
  assumes i-le-j: i < j
  assumes s-is: s = interval-times a L
  shows s!j > s!i
  using assms
proof (induct j−1 arbitrary: i j)
  case 0
  then have i = 0 and j = 1
     by simp-all
  then show ?case
    using interval-times-diff-ge 0 by fastforce
next
  case (Suc x)
  then have j-eq: j = x+2
    by simp
  have high: s ! (x + 1) < s ! (x + 2)
    using interval-times-diff-ge[of a b L x+1 s] Suc by simp
```

```
  {
    assume i-eq: i = x+1
    then have ?case unfolding i-eq j-eq
      using high by simp
  } moreover {
    assume i-eq: i ≤ x
    then have s ! i < s ! (x + 1)
      using Suc.hyps(1)[of x+1 i] Suc by force
    then have ?case using high i-eq j-eq by simp
  }
  ultimately show ?case using Suc j-eq by linarith
qed

lemma trivial-composition:
  assumes n > 0
  shows is-composition n [n]
proof−
  have pos: (∀ i. List.member [n] i ⟶ 0 < i)
    unfolding List.member-def
    by (simp add: assms)
  have sum:  sum-list [n] = n
    by simp
  show ?thesis unfolding is-composition-def
    using pos sum by blast
qed


lemma sum-list-pos: (⋀x. x ∈ set (xs::nat list) ⟹ 0 < x)
                    ⟹ length xs > 0 ⟹ 0 < sum-list xs
  by (induction xs) auto

lemma take-prefix:
  assumes L = H@[t]
  assumes k ≤ length L − 1
  shows take k H = take k L
  using assms by auto

lemma take-interval-times:
  assumes length L ≥ k
  shows take (k+1) (interval-times a L) = interval-times a (take k L)
  using assms
proof(induct length L arbitrary: L)
  case 0
  then show ?case
    by (simp add: interval-times-length)
next
  case (Suc x)
  then obtain H t where L-eq: L = H@[t]
    by (metis length-Suc-conv-rev)
```

```
  have ih: take (k + 1) (interval-times a H) = interval-times a (take k H)
    using Suc.hyps(1)[of H] Suc L-eq
      by (metis Suc-eq-plus1 add-left-cancel interval-times-length le-SucE le-add1
length-append-singleton plus-1-eq-Suc take-all-iff)
  have length-it: length (interval-times a L) = length L + 1
    unfolding interval-times-def by auto
  {
    assume *: k ≤ length L − 1
    then have eq1: (take k H) = (take k L)
      by (simp add: L-eq)
    have (interval-times a H)@[a+(sum-list L)] = interval-times a L
      using L-eq unfolding interval-times-def partial-sum-def by auto
   then have eq2: take (k + 1) (interval-times a H) = take (k + 1) (interval-times
a L)
      using take-prefix[of interval-times a L interval-times a H a + sum-list L]
       by (metis Suc-eq-plus1 diff-Suc-1 eq1 ih interval-times-length not-less-eq-eq
take-all)
    have ?case using eq1 eq2 ih by argo
  } moreover {
    assume *: k = length L
    then have ?case
      by (simp add: length-it)
  }
  ultimately show ?case using Suc by linarith
qed

lemma index-list-index:
  fixes k::nat
  assumes j < k
  shows [0 ..< k] ! j = j
  using assms by simp


lemma interval-times-obtain-aux:
  assumes a ≤ b
  assumes is-composition (b − a + 1) L
  assumes s = interval-times a L
  assumes (s ! 1) ≤ t ∧ t ≤ b
  shows ∃ i. s ! i ≤ t ∧ t ≤ s ! (i + 1) − 1 ∧ 1 ≤ i ∧ i < length L
proof−
  have length-s: length s = length L + 1
    using assms interval-times-length by auto
  have first: s!0 = a
    using interval-times-first assms by blast
  have last: s!(length L) = b+1
    using interval-times-last assms by blast
  {
    assume length-L: length L = 0
    then have ?thesis using assms
```

```
    by (metis first last less-add-one verit-comp-simplify1(3))
  } moreover {
    assume length-L: length L ≥ 1
    have ?thesis using assms first last length-s length-L
    proof(induct length L − 1 arbitrary: s L a b t)
      case 0
      then show ?case by auto
    next
      case (Suc x)
      then have length-L: length L ≥ 2 by linarith
      then have length-s: length s ≥ 3 using Suc by linarith
      {
        assume ∗: t < s!(length L−1)
        let ?L' = take (length L−1) L
        let ?s' = take (length L) s
        let ?b' = b − (List.last L)
        have pos-L: (∀ i. List.member L i ⟶ 0 < i) and
             sum-L: sum-list L = b − a + 1
          using Suc(4) unfolding is-composition-def by auto
        have List.member L (last L) unfolding List.member-def
          by (metis Suc.prems(8) last-in-set length-0-conv not-one-le-zero)
        have sum-list-eq: sum-list L = sum-list (take (length L−1) L) + last L
          using length-L
        proof(induct length L arbitrary: L)
          case 0
          then show ?case by auto
        next
          case (Suc xa)
          then obtain h T where L-eq: L = h#T
            by (meson Suc-length-conv)
          then have L-decomp: sum-list L = sum-list T + h by simp
          {
            assume length L = 2
            then obtain x1 x2 where L = [x1, x2]
                 by (metis Suc-1 Suc-length-conv gen-length-code(1) gen-length-def
impossible-Cons le-add2 list.exhaust plus-1-eq-Suc)
            then have ?case by auto
          } moreover {
            assume length-L: length L > 2
            then have last: last T = last L
              using L-eq by auto
            have ∗: sum-list T = sum-list (take (length T − 1) T) + last T
              using Suc.hyps(1)[of T] L-decomp L-eq length-L
                by (metis Suc.hyps(2) add-diff-cancel-left' length-Cons less-Suc-eq-le
plus-1-eq-Suc)
            have ∗∗: h + sum-list (take (length T − 1) T) = sum-list (take (length
L − 1) L)
              using L-eq
          by (metis (no-types, opaque-lifting) Suc.prems Suc-1 Suc-eq-plus1 Suc-le-D
```

add-diff-cancel-right′ add-le-same-cancel2 length-Cons not-less-eq-eq sum-list.Cons take-Suc-Cons)
        have *?case* **using** ∗ ∗∗ *last*
          using *L-decomp* **by** *presburger*
        }
        ultimately show *?case* **using** *Suc.prems* **by** *fastforce*
      qed
      have *pos-preL*: (⋀*x*. *x* ∈ *set* (*take* (*length L* − *1*) *L*) ⟹ *0* < *x*)
        using *pos-L*
        by (*metis in-set-member in-set-takeD*)
      have *length-preL*: *0* < *length* (*take* (*length L* − *1*) *L*)
        using *length-L* **by** *auto*
      have *sum-preL-pos*: *sum-list* (*take* (*length L*−*1*) *L*) > *0*
        using *sum-list-pos*[*of take* (*length L* − *1*) *L*]
        using *pos-preL length-preL* **by** *blast*
      then have *sum-last*: *sum-list L* > *last L* **using** *pos-L length-L*
        using *sum-list-pos sum-list-eq* **by** *linarith*
      then have *b-lastL*: *b* ≥ *last L*
        using *sum-L* **by** *auto*
      then have *ba-lastL*: *last L* ≤ *b* − *a*
        using *sum-L sum-last* **by** *auto*
      have *first*: *s*!*0* = *a*
        using *Suc interval-times-first* **by** *blast*
      have *last*: *s*!(*length L*) = *b*+*1*
        using *Suc interval-times-last* **by** *blast*
      have *c1*: *x* = *length* (*take* (*length L* − *1*) *L*) − *1*
        using *Suc* **by** *auto*
      have *c2*: *a* ≤ *b* − *last L*
        using *Suc*(*3*) *b-lastL ba-lastL* **by** *auto*
      have *c3* :*is-composition* (*b* − *last L* − *a* + *1*) (*take* (*length L* − *1*) *L*)
        using *Suc.prems*(*2*) **unfolding** *is-composition-def*
          by (*metis Suc-diff-1 Suc-eq-plus1* ‹*0* < *sum-list* (*take* (*length L* − *1*) *L*)› *add-diff-cancel-right diff-right-commute in-set-member plus-1-eq-Suc pos-preL sum-list-eq*)
      have *c4*: *take* (*length L*) *s* = *interval-times a* (*take* (*length L* − *1*) *L*)
        unfolding *Suc*(*5*) **using** *length-L take-interval-times*
        by (*metis Suc.prems*(*8*) *diff-add diff-le-self*)
      have *c5*: *take* (*length L*) *s* ! *1* ≤ *t* ∧ *t* ≤ *b* − *last L*
      proof−
        have *s*!(*length L*−*1*) = *a* + *sum-list* (*take* (*length L*−*1*) *L*)
          unfolding *Suc*(*5*) *interval-times-def partial-sum-def*
       by (*metis* (*no-types*, *lifting*) *Suc.prems*(*8*) *add.commute add-0 add-mono-thms-linordered-field*(*3*) *le-add-same-cancel2 less-numeral-extra*(*1*) *nth-map-upt ordered-cancel-comm-monoid-diff-class.add-diff-inverse zero-le*)
        then have *part1*: (*s* ! (*length L* − *1*))−*1* ≤ *b* − *last L*
          using *last sum-list-eq*
            by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-leI sum-preL-pos c2 c3 diff-add-inverse2 eq-imp-le is-composition-def order-eq-refl ordered-cancel-comm-monoid-diff-class.add-diff-inve ordered-cancel-comm-monoid-diff-class.diff-add-assoc*)

```
      have part2: take (length L) s ! 1 ≤ t
        using Suc.hyps(2) Suc.prems(4) by auto
      then show ?thesis using ∗ part1 part2
        by linarith
    qed
    have c6: take (length L) s ! 0 = a
      by (simp add: c4 interval-times-first)
    have c7: take (length L) s ! length (take (length L − 1) L) = b − last L + 1
    proof−
      have idx: length (take (length L − 1) L) = length L−1 by simp
      have p1: a + partial-sum L (length L−1) = b − last L + 1
        unfolding partial-sum-def
      by (metis add.assoc c2 c3 is-composition-def ordered-cancel-comm-monoid-diff-class.add-diff-inverse)

      have p2: take (length L) (map (λi. a + partial-sum L i) [0..<length L +
1]) ! (length L − 1)
          = (map (λi. a + partial-sum L i) [0..<length L + 1]) ! (length L − 1)
        by (meson Suc.prems(2) add-gr-0 composition-length-lb diff-less nth-take
zero-less-one)
      have p3: (map (λi. a + partial-sum L i) [0..<length L + 1]) ! (length L
− 1)
          = a + partial-sum L (length L−1)
      proof−
        have fact1: map (λi. a + partial-sum L i) [0..<length L + 1] ! (length
L − 1) =
            a + partial-sum L ([0..<length L + 1] ! (length L − 1))
          using nth-map[of (length L−1) [0..<length L + 1] (λi. a + partial-sum
L i)]
          by simp
        have length L ≥ 0
          using Suc(2) by auto
        then have fact2: ([(0::nat)..<length L + 1] ! (length L − 1)) = length
L −1
          using index-list-index[of length L−1 length L + 1] by simp
        then show ?thesis using fact1 fact2 by argo
      qed
      then have take (length L) s ! (length L−1) = b − last L + 1
        unfolding Suc(5) interval-times-def
        using p1 p2 p3 by argo
      then show ?thesis using idx by argo
    qed
    have c8: length (take (length L) s) = length (take (length L − 1) L) + 1
      using c4 interval-times-length by presburger
    have c9: 1 ≤ length (take (length L − 1) L)
      using length-preL by linarith
    have ih: ∃ i. take (length L) s ! i ≤ t ∧ t ≤ take (length L) s ! (i + 1) − 1
          ∧ 1 ≤ i ∧ i < length (take (length L − 1) L)
      using Suc(1)[of (take (length L − 1) L) a b − last L take (length L) s t,
          OF c1 c2 c3 c4 c5 c6 c7 c8 c9] by blast
```

```
      then obtain i where t-bound: take (length L) s ! i ≤ t ∧ t ≤ take (length
L) s ! (i + 1) − 1
                    and i-bound: 1 ≤ i ∧ i < length (take (length L − 1) L)
        by blast
      have i-bound-L: 1 ≤ i ∧ i < length L
        using i-bound by auto
      then have t-bound-L: s ! i ≤ t ∧ t ≤ s ! (i + 1) − 1
        using t-bound
          by (metis Suc.hyps(2) c1 c9 i-bound le-add-diff-inverse less-diff-conv
nth-take plus-1-eq-Suc)
      then have ?case using i-bound-L t-bound by auto
    } moreover {
      assume ∗: t ≥ s!(length L−1)
      then have ?case
     by (metis Suc.hyps(2) Suc.prems(4) Suc.prems(6) Suc.prems(8) add-diff-cancel-right'
diff-less le-add1 le-add-diff-inverse2 less-numeral-extra(1) order-less-le-trans plus-1-eq-Suc)

    }
    ultimately show ?case by fastforce
  qed
 }
 ultimately show ?thesis
  by (meson less-one verit-comp-simplify1(3))
qed


lemma interval-times-obtain:
 assumes a ≤ b
 assumes is-composition (b − a + 1) L
 assumes s = interval-times a L
 assumes a ≤ t ∧ t ≤ b
 shows ∃ i. s ! i ≤ t ∧ t ≤ s ! (i + 1) − 1 ∧ 0 ≤ i ∧ i < length L
proof−
 {
  assume ∗: (s ! 1) ≤ t
  from interval-times-obtain-aux[OF assms(1−3), of t] ∗ assms(4)
  obtain i where s ! i ≤ t ∧ t ≤ s ! (i + 1) − 1 ∧ 1 ≤ i ∧ i < length L
    by auto
  then have ?thesis by blast
 } moreover {
  assume ∗: t < s!1
  have sfirst: s!0 = a
    using interval-times-first unfolding assms by auto
  have length-L: 0 < length L
    using composition-length-lb[OF assms(2)] using assms by auto
  have t ≤ s ! 1 − 1
    using ∗ by simp
  then have s ! 0 ≤ t ∧ t ≤ s ! 1 − 1 ∧ 0 ≤ (0::nat) ∧ 0 < length L
    using ∗ assms unfolding sfirst using length-L by blast
```

```
    then have ?thesis by auto
  }
  ultimately show ?thesis by force
qed

lemma list-allones:
  assumes ∀ i<length L. L!i = 1
  shows L = map (λi. 1) [0 ..< length L]
  using assms
proof(induct L)
  case Nil
  then show ?case by simp
next
  case (Cons a L)
  then show ?case
    by (metis (no-types, lifting) length-map list-eq-iff-nth-eq map-nth nth-map)
qed

lemma sum-list-constants:
  fixes L::nat list and k::nat
  assumes ∀ i<length L. L ! i = k
  shows sum-list L = k*(length L)
  using assms by(induct L) force+

lemma length-is-composition-allones:
  assumes is-composition-allones n L
  shows length L = n
  using assms unfolding is-composition-allones-def is-composition-def
  by (metis mult-1 sum-list-constants)


lemma partial-sum-allones:
  assumes (∀ i<length L. L ! i = 1)
  assumes i ≤ length L
  shows partial-sum L i = i
  using assms
proof(induct length L arbitrary: i L)
  case 0
  then have i0: i = 0 by auto
  have L-empty: L = [] using 0 by auto
  show ?case using L-empty i0
    unfolding partial-sum-def by simp
next
  case (Suc x)
  then obtain H t where L-is: L = H@[t]
    by (metis length-Suc-conv-rev)
  have L-ones: L = map (λi. 1) [0..<length L]
    using list-allones Suc by blast
  {
```

```
    assume ∗: i = length L
    then have takeall: take i L = L
      using take-all[of L i] by simp
    have ?case unfolding takeall partial-sum-def
      using Suc(3) ∗ sum-list-constants[of L 1] by simp
  } moreover {
    assume ∗: i < length L
    have cond1: x = length H
      using Suc L-is by simp
    have cond2: ∀ i<length H. H ! i = 1
      using Suc(3) unfolding L-is
     by (metis L-is Suc.hyps(2) Suc-lessD Suc-mono butlast-snoc cond1 nth-butlast)

    have cond3: i ≤ length H
      using ∗ L-is by auto
    then have ?case
      using Suc(1)[of H i, OF cond1 cond2 cond3]
      unfolding partial-sum-def L-is by simp
  }
  ultimately show ?case using L-is Suc by fastforce
qed

lemma interval-times-allones:
  assumes a ≤ b
  assumes is-composition-allones (b − a + 1) L
  assumes i < length (interval-times a L)
  shows (interval-times a L)!i = a+i
proof−
  have ∗: map (λi. a + partial-sum L i) [0..<length L + 1] ! i = a + partial-sum
L i
    using assms
   by (metis interval-times-def length-map length-upt nth-map-upt plus-nat.add-0)

  have allones: ∀ i<length L. L!i = 1
    using assms(2) unfolding is-composition-allones-def
    by blast
  have length (interval-times a L) = length L + 1
    using interval-times-length by simp
  then have partial-sum L i = i
    using partial-sum-allones[of L i]
    using allones assms by simp
  then have a + partial-sum L i = a + i
    by auto
  then show ?thesis
    unfolding interval-times-def
    using ∗ by auto
qed

lemma allones-implies-is-composition:
```

assumes *is-composition-allones n L*
shows *is-composition n L*
using *assms* **unfolding** *is-composition-allones-def* **by** *blast*

lemma *allones-implies-is-composition-MLTL*:
assumes *is-composition-MLTL-allones* φ
shows *is-composition-MLTL* φ
using *assms allones-implies-is-composition*
by (*induct* φ) *simp-all*

6 MLTL Decomposition Lemmas

lemma *LP-mltl-nnf*:
fixes φ::*'a mltl-ext* **and** ψ::*'a mltl* **and** *k*::*nat*
assumes ψ-*coformula*: $\psi \in$ *set* (*LP-mltl* φ *k*)
shows $\exists \psi$-*init*. ψ = *convert-nnf* ψ-*init*
proof−
obtain ψ-*init* **where** ψ = *to-mltl* (*convert-nnf-ext* ψ-*init*)
using *assms* **unfolding** *LP-mltl.simps* **by** *auto*
then have ψ = *convert-nnf* (*to-mltl* ψ-*init*)
using *convert-nnf-ext-to-mltl-commute* **by** *metis*
then show *?thesis*
by *blast*
qed

lemma *LP-mltl-element*:
fixes ψ::*'a mltl* **and** φ::*'a mltl-ext*
shows $\psi \in$ *set* (*LP-mltl* φ *k*) $\longleftrightarrow$
($\exists \psi$-*ext* $\in$ *set* (*LP-mltl-aux* (*convert-nnf-ext* φ) *k*).
ψ = *to-mltl* (*convert-nnf-ext* ψ-*ext*))
unfolding *LP-mltl.simps* **by** *auto*

7 Lemmas for MLTL operators that operate over lists of mltl formulas

lemma *pairs-alt*:
shows *set* (*pairs L1* (*h2*#*T2*)) =
set ((*map* (λx. (*x*, *h2*)) *L1*) @ (*pairs L1 T2*))
proof(*induct L1 arbitrary*: *h2 T2*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a L1*)
have *pairs-fact*: *set* (*pairs* (*a*#*L1*) (*h2*#*T2*)) = *set* (*map* (*Pair a*) (*h2* # *T2*) @ *pairs L1* (*h2* # *T2*))
unfolding *pairs.simps* **by** *auto*
have *ih*: *set* (*pairs L1* (*h2* # *T2*)) = *set* (*map* (λx. (*x*, *h2*)) *L1* @ *pairs L1 T2*)
using *Cons.hyps*[*of h2 T2*] **by** *simp*

```
  have ∗: set (pairs (a#L1) (h2#T2)) =
  set (map (Pair a) (h2 # T2)) ∪ set (map (λx. (x, h2)) L1 @ pairs L1 T2)
    using pairs-fact ih by auto
  have ∗∗: set (pairs (a # L1) T2) = set (map (Pair a) T2 @ pairs L1 T2)
    using pairs.simps by simp
  then show ?case using ∗ ∗∗ by auto
qed

lemma list-concat-set-union:
  shows set(A@B) = set A ∪ set B
  by simp

lemma pairs-empty-list:
  shows pairs A [] = []
proof(induct A)
  case Nil
  then show ?case by simp
next
  case (Cons a A)
  then show ?case by auto
qed
```

7.1 Forward Direction Proofs

```
lemma pairs-member-fst-forward:
  assumes List.member (pairs A B) x
  shows List.member A (fst x)
  using assms
proof(induct A)
  case Nil
  then have pairs [] B = [] unfolding pairs.simps by simp
  then show ?case using member-rec(2)
    by (metis Nil)
next
  case (Cons a A)
  {assume fst-x-is-a: fst x = a
    then have ?case
      using Cons member-rec(1) by metis
  } moreover {
    assume fst-x-not-a: fst x ≠ a
    then have ¬(List.member (map (Pair a) B) x)
      using in-set-member by force
    then have List.member (pairs A B) x
      using Cons(2) unfolding pairs.simps List.member-def by auto
    then have ih: List.member A (fst x)
      using Cons.hyps by blast
    then have List.member (a # A) (fst x)
      unfolding List.member-def by simp
    then have ?case
```

```
    using ih by blast
  }
  ultimately show ?case by blast
qed

lemma pairs-member-snd-forward:
  assumes List.member (pairs A B) x
  shows List.member B (snd x)
  using assms
proof(induct B)
  case Nil
  have pairs A [] = []
    using pairs-empty-list by blast
  then show ?case
    by (metis local.Nil member-rec(2))
next
  case (Cons b B)
  have pairs-rec: set (pairs A (b # B)) = set (map (λx. (x, b)) A @ pairs A B)
    using pairs-alt[of A b B] by blast
  {assume snd-x-is-b: snd x = b
    then have ?case
      using Cons member-rec(1) by metis
  } moreover {
    assume snd-x-not-b: snd x ≠ b
    then have ¬(List.member (map (λx. (x, b)) A) x)
      using in-set-member pairs-rec by force
    then have List.member (pairs A B) x
      using Cons(2) unfolding pairs-rec List.member-def by simp
    then have ih: List.member B (snd x)
      using Cons.hyps by blast
    then have List.member (b # B) (snd x)
      unfolding List.member-def by simp
    then have ?case
      using ih by blast
  }
  ultimately show ?case by blast
qed

lemma pairs-member-forward:
  assumes List.member (pairs A B) x
  shows List.member A (fst x) ∧ List.member B (snd x)
  using assms pairs-member-fst-forward pairs-member-snd-forward by blast

lemma And-mltl-list-member-forward:
  assumes List.member (And-mltl-list D-x D-y) ψ
  shows ∃ψ1 ψ2. ψ = And-mltl-ext ψ1 ψ2
  ∧ List.member D-x ψ1 ∧ List.member D-y ψ2
proof−
  obtain x where ψ = And-mltl-ext (fst x) (snd x) ∧ x ∈ set (pairs D-x D-y)
```

using *assms* **unfolding** *And-mltl-list.simps List.member-def* **by** *auto*
then show *?thesis*
using *pairs-member-forward*[*of D-x D-y x*]
by (*simp add*: *in-set-member*)
qed

7.2 Converse Direction Proofs

lemma *pairs-member-converse*:
assumes *List.member A* (*fst x*)
assumes *List.member B* (*snd x*)
shows *List.member* (*pairs A B*) *x*
using *assms*
proof(*induct A*)
case *Nil*
then show *?case* **unfolding** *List.member-def* **by** *simp*
next
case (*Cons a A*)
{assume ∗: *fst x* = *a*
then have *?case* **using** *Cons*
unfolding *pairs.simps List.member-def* **by** *force*
} moreover {
assume ∗: *fst x* ∈ *set A*
then have *List.member* (*pairs A B*) *x*
using *Cons.hyps Cons*(*3*) **unfolding** *List.member-def* **by** *simp*
then have *?case* **unfolding** *pairs.simps List.member-def* **by** *simp*
}
ultimately show *?case* **using** *Cons*(*2*) **unfolding** *List.member-def* **by** *force*
qed

lemma *And-mltl-list-member-converse*:
assumes ∃ *ψ1 ψ2*. *ψ* = *And-mltl-ext ψ1 ψ2*
∧ *List.member D-x ψ1* ∧ *List.member D-y ψ2*
shows *List.member* (*And-mltl-list D-x D-y*) *ψ*
proof−
from *assms* **obtain** *ψ1 ψ2* **where** *ψ* = *And-mltl-ext ψ1 ψ2* ∧ *List.member D-x ψ1* ∧ *List.member D-y ψ2*
by *blast*
then show *?thesis* **using** *pairs-member-converse*
unfolding *And-mltl-list.simps List.member-def* **by** *force*
qed

7.3 Biconditional Lemmas

lemma *pairs-member*:
shows (*List.member A* (*fst x*) ∧ *List.member B* (*snd x*)) ⟷
List.member (*pairs A B*) *x*
using *pairs-member-forward pairs-member-converse* **by** *blast*

lemma *And-mltl-list-member*:
shows ($\exists \psi 1\ \psi 2.\ \psi$ = *And-mltl-ext* $\psi 1\ \psi 2$
$\wedge$ *List.member D-x* $\psi 1 \wedge$ *List.member D-y* $\psi 2$) $\longleftrightarrow$
List.member (*And-mltl-list D-x D-y*) ψ
using *And-mltl-list-member-forward And-mltl-list-member-converse* **by** *blast*

8 MLTL Decomposition Top Level Correctness

fun *wpd-mltl*:: *'a mltl* $\Rightarrow$ *nat*
where *wpd-mltl* $False_m$ = *1*
| *wpd-mltl* $True_m$ = *1*
| *wpd-mltl* ($Prop_m$ (*p*)) = *1*
| *wpd-mltl* ($Not_m\ \varphi$) = *wpd-mltl* φ
| *wpd-mltl* ($\varphi\ And_m\ \psi$) = *max* (*wpd-mltl* φ) (*wpd-mltl* ψ)
| *wpd-mltl* ($\varphi\ Or_m\ \psi$) = *max* (*wpd-mltl* φ) (*wpd-mltl* ψ)
| *wpd-mltl* (G_m[*a*,*b*] φ) = *b* + (*wpd-mltl* φ)
| *wpd-mltl* (F_m[*a*,*b*] φ) = *b* + (*wpd-mltl* φ)
| *wpd-mltl* ($\varphi\ R_m$ [*a*,*b*] ψ) = *b* + (*max* ((*wpd-mltl* φ)) (*wpd-mltl* ψ))
| *wpd-mltl* ($\varphi\ U_m$ [*a*,*b*] ψ) = *b* + (*max* ((*wpd-mltl* φ)) (*wpd-mltl* ψ))

8.1 Helper Lemmas

lemma *wpd-geq-one*:
shows *wpd-mltl* $\varphi \geq$ *1*
by (*induct* φ) *simp-all*

lemma *wpd-convert-nnf*:
fixes φ::*'a mltl*
shows *wpd-mltl* (*convert-nnf* φ) = *wpd-mltl* φ
proof(*induction depth-mltl* φ *arbitrary*: φ *rule*: *less-induct*)
case *less*
have *not*: ($\bigwedge\varphi$. *depth-mltl* φ < *Suc* (*depth-mltl p*) $\Longrightarrow$
wpd-mltl (*convert-nnf* φ) = *wpd-mltl* φ) $\Longrightarrow$
$\varphi = Not_m\ p \Longrightarrow$
wpd-mltl (*convert-nnf* (Not_m *p*)) = *wpd-mltl p* **for** *p*
proof−
assume *ih*: $\bigwedge\varphi$. *depth-mltl* φ < *Suc* (*depth-mltl p*) $\Longrightarrow$
wpd-mltl (*convert-nnf* φ) = *wpd-mltl* φ
assume *notcase*: $\varphi = Not_m\ p$
show *?thesis* **using** *ih notcase less* **by** (*induct p*) *simp-all*
qed
show *?case* **using** *less not* **by** (*cases* φ) *auto*
qed

lemma *convert-nnf-ext-preserves-wpd*:
shows *wpd-mltl* (*to-mltl* (*convert-nnf-ext* φ)) =
wpd-mltl (*to-mltl* φ)
proof(*induction depth-mltl* (*to-mltl* φ) *arbitrary*: φ *rule*: *less-induct*)
case *less*

```
  have not: (⋀φ. depth-mltl (to-mltl φ)
              < Suc (depth-mltl (to-mltl x)) ⟹
              wpd-mltl (to-mltl (convert-nnf-ext φ)) =
              wpd-mltl (to-mltl φ)) ⟹
          φ = Not_c x ⟹
          wpd-mltl (to-mltl (convert-nnf-ext (Not_c x))) =
          wpd-mltl (to-mltl x) for x
  proof−
    assume ih: (⋀φ. depth-mltl (to-mltl φ)
              < Suc (depth-mltl (to-mltl x)) ⟹
              wpd-mltl (to-mltl (convert-nnf-ext φ)) =
              wpd-mltl (to-mltl φ))
    assume shape: φ = Not_c x
    show ?thesis using ih shape less by (induct x) simp-all
  qed
  show ?case using less not
    by (cases φ) auto
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 ⟹
          F1 = Not_m F2 ⟹
          intervals-welldef (convert-nnf (Not_m F2))
    for F2  using less by (cases F2) simp-all
  then show ?case using less by (cases F1) simp-all
qed

lemma is-composition-convert-nnf-ext:
  fixes φ::'a mltl-ext
  assumes intervals-welldef (to-mltl φ)
  assumes is-composition-MLTL φ
  shows is-composition-MLTL (convert-nnf-ext φ)
  using assms
proof(induct depth-mltl (to-mltl φ) arbitrary: φ rule: less-induct)
  case less
  have not-case: (⋀φ. depth-mltl (to-mltl φ)
              < Suc (depth-mltl (to-mltl x4)) ⟹
              intervals-welldef (to-mltl φ) ⟹
              is-composition-MLTL φ ⟹
              is-composition-MLTL (convert-nnf-ext φ)) ⟹
          intervals-welldef (to-mltl x4) ⟹
          is-composition-MLTL x4 ⟹
          φ = Not_c x4 ⟹
```

is-composition-MLTL (*convert-nnf-ext* (*Not*$_c$ *x4*)) **for** *x4*
using *less* **by** (*induct x4*) *simp-all*
show *?case* **using** *less not-case* **by** (*cases* φ) *auto*
qed

lemma *is-composition-allones-convert-nnf-ext*:
fixes φ::*'a mltl-ext*
assumes *intervals-welldef* (*to-mltl* φ)
assumes *is-composition-MLTL-allones* φ
shows *is-composition-MLTL-allones* (*convert-nnf-ext* φ)
using *assms*
proof(*induct depth-mltl* (*to-mltl* φ) *arbitrary*: φ *rule*: *less-induct*)
case *less*
have *not-case*: ($\bigwedge\varphi$. *depth-mltl* (*to-mltl* φ)
$<$ *Suc* (*depth-mltl* (*to-mltl x4*)) $\Longrightarrow$
intervals-welldef (*to-mltl* φ) $\Longrightarrow$
is-composition-MLTL-allones φ $\Longrightarrow$
is-composition-MLTL-allones (*convert-nnf-ext* φ)) $\Longrightarrow$
intervals-welldef (*to-mltl x4*) $\Longrightarrow$
is-composition-MLTL-allones x4 $\Longrightarrow$
φ = *Not*$_c$ *x4* $\Longrightarrow$
is-composition-MLTL-allones (*convert-nnf-ext* (*Not*$_c$ *x4*)) **for** *x4*
using *less* **by** (*induct x4*) *simp-all*
show *?case* **using** *less not-case*
by (*cases* φ) *auto*
qed

function *Ands-mltl-ext*:: *'a mltl-ext list* $\Rightarrow$ *'a mltl-ext*
where *Ands-mltl-ext* [] = *True-mltl-ext*
| *Ands-mltl-ext* (*H*@[*t*]) = (*if* (*length H* = *0*) *then t*
else (*And-mltl-ext* (*Ands-mltl-ext H*) *t*))
using *rev-exhaust* **by** *auto*
termination by (*relation* *measure* (λ*L*. *length L*)) *auto*

lemma *Ands-mltl-semantics*:
assumes *length X* $\geq$ *1*
shows *semantics-mltl-ext* π (*Ands-mltl-ext X*) $\longleftrightarrow$
($\forall$ *x* $\in$ *set X*. *semantics-mltl-ext* π *x*)
using *assms*
proof(*induct length X*$-$*1 arbitrary*: *X*)
case *0*
then obtain *x* **where** *X-is*: *X* = [*x*]
by (*metis butlast-snoc diff-is-0-eq le-antisym length-0-conv length-butlast list.exhaust*
rotate1.simps(*2*) *rotate1-length01 zero-neq-one*)
then show *?case* **unfolding** *X-is*

using *Ands-mltl-ext.simps(2)*[*of* [] *x*] **by** *simp*
next
case (*Suc n*)
then obtain *H t* **where** *X-is*: $X = H@[t]$
by (*metis Ands-mltl-ext.cases One-nat-def Suc-n-not-le-n gen-length-code(1) length-code*)
then have *length-H*: *length H* = *n+1* **using** *Suc* **by** *auto*
then have *cond1*: *n* = *length H* − *1* **by** *simp*
have *cond2*: *length H* ≥ *1* **using** *length-H* **by** *simp*
have *semantics-H*: *semantics-mltl-ext π* (*Ands-mltl-ext H*) =
$(\forall x.\ x \in set\ H \longrightarrow semantics\text{-}mltl\text{-}ext\ \pi\ x)$
using *Suc(1)*[*OF cond1 cond2*] **unfolding** *Ball-def* **by** *simp*
have (*semantics-mltl-ext π* (*Ands-mltl-ext H*) ∧
semantics-mltl-ext π t) ⟷
$(\forall x.\ x \in set\ (H\ @\ [t]) \longrightarrow semantics\text{-}mltl\text{-}ext\ \pi\ x)$
using *semantics-H* **by** *auto*
then have *semantics-mltl-ext π* (*And-mltl-ext* (*Ands-mltl-ext H*) *t*) =
$(\forall x.\ x \in set\ (H\ @\ [t]) \longrightarrow semantics\text{-}mltl\text{-}ext\ \pi\ x)$
unfolding *semantics-mltl-ext-def to-mltl.simps* **by** *simp*
then show *?case* **unfolding** *Ball-def X-is Ands-mltl-ext.simps*
using *length-H* **by** *simp*
qed

lemma *in-Global-mltl-decomp*:
assumes *length D-φ* > *1*
assumes *ψ* ∈ *set* (*Global-mltl-decomp D-φ a n L*)
shows ∃ *X*. ((*ψ* = *Ands-mltl-ext X* ∧
(∀ *x*. *List.member X x* ⟶
(∃ *y* ∈ *set D-φ*. (∃ *k*. *a* ≤ *k* ∧ *k* ≤ (*a*+*n*) ∧ *x* = *Global-mltl-ext k k* [*1*] *y*)))) ∧
(*length X* = *Suc n*))
using *assms*
proof(*induct n arbitrary*: *D-φ ψ a*)
case *0*
then obtain *x* **where** *x-in*: *x* ∈ *set D-φ* **and**
ψ-is: *ψ* = *Global-mltl-ext a a* [*1*] *x*
unfolding *Global-mltl-decomp.simps Global-mltl-list.simps* **by** *auto*
then have *ψ* = *Ands-mltl-ext* [*Global-mltl-ext a a* [*1*] *x*]
using *Ands-mltl-ext.simps(2)*[*of* [] *Global-mltl-ext a a* [*1*] *x*] **by** *auto*
then show *?case*
by (*metis add.right-neutral length-Cons list.size(3) member-rec(1) member-rec(2) order-refl x-in*)
next
case (*Suc x*)
then have *ψ* ∈ *set* (*And-mltl-list* (*Global-mltl-decomp D-φ a x L*)
(*Global-mltl-list D-φ* (*a* + *Suc x*) (*a* + *Suc x*) [*1*]))
unfolding *Global-mltl-decomp.simps* **by** *force*
then obtain *first second* **where** *ψ-is*: *ψ* = *And-mltl-ext first second*
and *first-in*: *first* ∈ *set* (*Global-mltl-decomp D-φ a x L*)

```
      and second-in: second ∈ set (Global-mltl-list D-φ (a + Suc x) (a + Suc x)
[1])
    using And-mltl-list-member by (metis in-set-member)
  from Suc.hyps[OF Suc.prems(1) first-in] obtain X where
      X1: first = Ands-mltl-ext X and
      X2: (∀ xa. List.member X xa ⟶
            (∃ y∈set D-φ. ∃ k≥a. k ≤ a + x ∧ xa = Global-mltl-ext k k [1] y)) and
      X3: length X = (Suc x)
    by blast
  from second-in obtain x-second where
      second-is: second = Global-mltl-ext (a + Suc x) (a + Suc x) [1] x-second
  and x-second-in: x-second ∈ set D-φ by auto
  have prop1: ψ = Ands-mltl-ext (X@[second]) using ψ-is X1
    unfolding Ands-mltl-ext.simps using X3 by auto
  have prop2: (∃ y∈set D-φ. ∃ k≥a. k ≤ a + Suc x ∧ xa = Global-mltl-ext k k [1]
y)
    if prem: List.member (X@[second]) xa for xa
    using X2 second-is
  proof−
    have split: (List.member X xa) ∨ xa = second
      using prem
      by (metis in-set-member member-rec(1) rotate1.simps(2) set-rotate1)
    {assume in-X: List.member X xa
      have ?thesis using X2 in-X by force
    } moreover {
      assume in-second: xa = second
      have ?thesis using in-second second-is
        by (simp add: x-second-in)
    }
    ultimately show ?thesis using split by blast
  qed
  have prop3: length (X@[second]) = Suc (Suc x)
    using X3 by simp
  then show ?case
    using prop1 prop2 prop3 by blast
qed


lemma in-Global-mltl-decomp-exact-forward:
  assumes length D-φ > 1
  assumes ψ ∈ set (Global-mltl-decomp D-φ a n L)
  shows ∃ X. ((ψ = Ands-mltl-ext X ∧
          (∀ i < length X. (∃ y ∈ set D-φ. (X!i) = Global-mltl-ext (a+i) (a+i) [1]
y)))) ∧
           (length X = Suc n)
  using assms
proof(induct n arbitrary: D-φ ψ a)
  case 0
  then obtain x where x-in: x ∈ set D-φ and
```

```
                    ψ-is: ψ = Global-mltl-ext a a [1] x
    unfolding Global-mltl-decomp.simps Global-mltl-list.simps by auto
  then have ψ = Ands-mltl-ext [Global-mltl-ext a a [1] x]
    using Ands-mltl-ext.simps(2)[of [] Global-mltl-ext a a [1] x] by auto
  then show ?case
    using x-in by auto
next
  case (Suc n)
  obtain H t where ψ-is: ψ = And-mltl-ext H t
              and H-in: H ∈ set (Global-mltl-decomp D-φ a n L)
              and t-in: t ∈ set (Global-mltl-list D-φ (a + Suc n) (a + Suc n) [1])
    using Suc(3) unfolding Global-mltl-decomp.simps
    using And-mltl-list-member unfolding List.member-def
    by (metis add-diff-cancel-left' plus-1-eq-Suc)
  obtain x where t-is: t = Global-mltl-ext (a+Suc n) (a+Suc n) [1] x
            and x-in: x ∈ set D-φ
    using t-in unfolding Global-mltl-list.simps by auto
  have ∃ X. (H = Ands-mltl-ext X ∧
      (∀ i<length X. ∃ y∈set D-φ. X ! i = Global-mltl-ext (a + i) (a + i) [1] y)) ∧
      length X = Suc n
    using Suc.hyps[of D-φ H a] Suc.prems H-in by blast
  then obtain X where H-is: H = Ands-mltl-ext X
                  and X-prop: ∀ i<length X. ∃ y∈set D-φ. X ! i = Global-mltl-ext (a
+ i) (a + i) [1] y
                  and length-X: length X = Suc n
    by blast
  have ψ-is: ψ = Ands-mltl-ext (X@[t])
    unfolding Ands-mltl-ext.simps using length-X ψ-is
    by (simp add: H-is)
  have property: ∃ y∈set D-φ. (X@[t]) ! i = Global-mltl-ext (a + i) (a + i) [1] y
    if i-bound: i<length (X@[t]) for i
  proof−
    {
      assume ∗: i < length X
      then have X ! i = (X@[t])!i using length-X
        by (simp add: nth-append)
      then have ?thesis using X-prop length-X ∗ by metis
    } moreover {
      assume ∗: i = length X
      have (X@[t])!i = t
        using length-X ∗
        by (metis nth-append-length)
      then have ?thesis using t-is ∗ length-X
        by (simp add: x-in)
    }
    ultimately show ?thesis using i-bound by fastforce
  qed
  have len: length (X@[t]) = Suc (Suc n)
    using length-X by auto
```

```
  then show ?case
    using ψ-is property len by blast
qed

lemma in-Global-mltl-decomp-exact-converse:
  fixes n::nat and X::'a mltl-ext list
  assumes length D-φ > 1
  assumes ψ = Ands-mltl-ext X
  assumes (∀ i < length X. (∃ y ∈ set D-φ.
          (X!i) = Global-mltl-ext (a+i) (a+i) [1] y))
  assumes length X = n+1
  shows ψ ∈ set (Global-mltl-decomp D-φ a n L)
  using assms
proof(induct n arbitrary: X ψ a)
  case 0
  then have length-X: length X = 1 by auto
  then have ∃ x. X = [x]
   by (metis Suc-eq-plus1 add-cancel-right-left length-Cons list.size(3) neq-Nil-conv
zero-eq-add-iff-both-eq-0 zero-neq-one)
  then obtain x where X-is: X = [x] by blast
  then obtain y where x-is: x = Global-mltl-ext a a [1] y
                  and y-in: y ∈ set D-φ
    using 0 by auto
  then show ?case unfolding 0(2) X-is
    using Ands-mltl-ext.simps(2)[of [] x] by simp
next
  case (Suc n)
  then have length-X: length X = n+2 by simp
  then obtain H t where X-is: X = H@[t]
    by (metis Suc.prems(4) Suc-eq-plus1 length-Suc-conv-rev)
  have length-H: length H = n+1 using length-X X-is by auto
  have ψ-is: ψ = And-mltl-ext (Ands-mltl-ext H) t
    using Suc(3) unfolding X-is Ands-mltl-ext.simps
    using length-H by simp
  have H-prop: ∃ y∈set D-φ. H ! i = Global-mltl-ext (a + i) (a + i) [1] y
    if i-bound: i<length H for i
  proof−
    have index: (H @ [t]) ! i = H!i
      using i-bound by (simp add: nth-append)
    then have ∃ y∈set D-φ. (H @ [t]) ! i = Global-mltl-ext (a + i) (a + i) [1] y
      using i-bound Suc(4) unfolding X-is
     by (metis Suc.prems(4) Suc-eq-plus1 X-is length-H plus-1-eq-Suc trans-less-add2)

    then show ?thesis
      using index by auto
  qed
  then have H-prop: ∀ i<length H.
      ∃ y∈set D-φ. H ! i = Global-mltl-ext (a + i) (a + i) [1] y
    by blast
```

```
  have H-in: Ands-mltl-ext H ∈ set (Global-mltl-decomp D-φ a n L)
    using Suc(1)[OF Suc(2) - H-prop, of (Ands-mltl-ext H)]
    using length-H by blast
  have t-is: ∃ y∈set D-φ. t = Global-mltl-ext (a + n + 1) (a + n + 1) [1] y
    using Suc(4) unfolding X-is using length-X
    by (metis X-is add.assoc length-H less-add-one nth-append-length one-add-one)
  then obtain y where t-is: t = Global-mltl-ext (a + n + 1) (a + n + 1) [1] y
                   and y-in: y ∈ set D-φ
    by blast
  have t-in: t ∈ set (Global-mltl-list D-φ (a + Suc n) (a + Suc n) [1])
    using y-in t-is by simp
  show ?case unfolding ψ-is Global-mltl-decomp.simps
     using t-in H-in And-mltl-list-member[of ψ (Global-mltl-decomp D-φ a n) L
(Global-mltl-list D-φ (a + Suc n) (a + Suc n) [1])]
    unfolding List.member-def ψ-is by auto
qed

lemma case-split-helper:
  assumes x ∈ A ∪ B ∪ C
  assumes x ∈ A ⟹ P x and x ∈ B ⟹ P x and x ∈ C ⟹ P x
  shows P x
  using assms by blast

lemma LP-mltl-aux-intervals-welldef:
  fixes φ ψ::'a mltl-ext
  assumes intervals-welldef (to-mltl φ)
  assumes ψ ∈ set (LP-mltl-aux (convert-nnf-ext φ) k)
  assumes is-composition-MLTL φ
  shows intervals-welldef (to-mltl ψ)
  using assms
proof(induct k arbitrary: φ ψ)
  case 0
  then show ?case unfolding LP-mltl-aux.simps
    by (simp add: convert-nnf-and-convert-nnf-ext nnf-intervals-welldef)
next
  case (Suc k)
  then show ?case
  proof(cases convert-nnf-ext φ)
    case True-mltl-ext
    then show ?thesis using Suc by simp
  next
    case False-mltl-ext
    then show ?thesis using Suc by simp
  next
    case (Prop-mltl-ext p)
    then show ?thesis using Suc by simp
  next
    case (Not-mltl-ext q)
    then have ∃ p. q = Prop-mltl-ext p
```

```
    using convert-nnf-form-Not-Implies-Prop
   by (metis convert-nnf-ext-to-mltl-commute to-mltl.simps(4) to-mltl-prop-bijective)

   then obtain p where q = Prop-mltl-ext p by auto
   then show ?thesis using Suc
    by (simp add: Not-mltl-ext)
 next
   case (And-mltl-ext α β)
   obtain x y where ψ-is: ψ = And-mltl-ext x y
            and x-in: x ∈ set (LP-mltl-aux (convert-nnf-ext α) k)
            and y-in: y ∈ set (LP-mltl-aux (convert-nnf-ext β) k)
    using Suc(3) unfolding And-mltl-ext LP-mltl-aux.simps
    by (meson And-mltl-list-member in-set-member)
   then show ?thesis unfolding ψ-is to-mltl.simps intervals-welldef.simps
    using Suc.hyps x-in y-in
   by (metis And-mltl-ext Suc.prems(1) Suc.prems(3) convert-nnf-ext-to-mltl-commute
intervals-welldef.simps(5) nnf-intervals-welldef is-composition-MLTL.simps(1) is-composition-convert-nnf-ext
to-mltl.simps(5))
 next
   case (Or-mltl-ext α β)
   let ?Dx = LP-mltl-aux (convert-nnf-ext α) k
   let ?Dy = LP-mltl-aux (convert-nnf-ext β) k
   {assume ∗: ψ ∈ set (And-mltl-list ?Dx ?Dy)
    then obtain x y where ψ-is: ψ = And-mltl-ext x y
            and x-in: x ∈ set ?Dx and y-in: y ∈ set ?Dy
      using Suc(3) LP-mltl-aux.simps
      by (meson And-mltl-list-member in-set-member)
   then have ?thesis unfolding Or-mltl-ext
   by (metis Or-mltl-ext Suc.hyps Suc.prems(1) Suc.prems(3) convert-nnf-ext-to-mltl-commute
intervals-welldef.simps(5) intervals-welldef.simps(6) nnf-intervals-welldef is-composition-MLTL.simps(2)
is-composition-convert-nnf-ext to-mltl.simps(5) to-mltl.simps(6))
   } moreover {
    assume ∗: ψ ∈ set (And-mltl-list [Not_c α] ?Dy)
    then obtain y where ψ-is: ψ = And-mltl-ext (Not_c α) y
            and y-in: y ∈ set ?Dy
      using Suc(3)
      using And-mltl-list-member[of ψ ?Dy [Not_c α]] by auto
    have lhs-welldef: intervals-welldef (to-mltl α)
      by (metis Or-mltl-ext Suc.prems(1) convert-nnf-ext-to-mltl-commute inter-
vals-welldef.simps(6) nnf-intervals-welldef to-mltl.simps(6))
    have rhs-welldef: intervals-welldef (to-mltl y)
      using y-in Suc.prems unfolding Or-mltl-ext
        by (metis Or-mltl-ext Suc.hyps convert-nnf-ext-to-mltl-commute inter-
vals-welldef.simps(6) nnf-intervals-welldef is-composition-MLTL.simps(2) is-composition-convert-nnf-ext
to-mltl.simps(6))
    then have ?thesis
      unfolding ψ-is to-mltl.simps intervals-welldef.simps
      using lhs-welldef rhs-welldef by blast
   } moreover {
```

assume $*$: $\psi \in$ *set* (*And-mltl-list ?Dx* [Not_c β])
then obtain x **where** ψ*-is*: ψ = *And-mltl-ext x* (Not_c β)
and *x-in*: $x \in$ *set ?Dx*
using *Suc*(*3*) *And-mltl-list-member*[*of* ψ *?Dx* [Not_c β]]
by (*metis in-set-member member-rec*(*1*) *member-rec*(*2*))
have *lhs-welldef*: *intervals-welldef* (*to-mltl* β)
by (*metis Or-mltl-ext Suc.prems*(*1*) *convert-nnf-ext-to-mltl-commute intervals-welldef.simps*(*6*) *nnf-intervals-welldef to-mltl.simps*(*6*))
have *rhs-welldef*: *intervals-welldef* (*to-mltl x*)
using *x-in Suc.prems* **unfolding** *Or-mltl-ext*
by (*metis Or-mltl-ext Suc.hyps convert-nnf-ext-to-mltl-commute intervals-welldef.simps*(*6*) *nnf-intervals-welldef is-composition-MLTL.simps*(*2*) *is-composition-convert-nnf-ext to-mltl.simps*(*6*))
then have *?thesis*
unfolding ψ*-is to-mltl.simps intervals-welldef.simps*
using *lhs-welldef rhs-welldef* **by** *blast*
}
ultimately show *?thesis*
using *Suc*(*3*) **unfolding** *Or-mltl-ext LP-mltl-aux.simps*
using *list-concat-set-union*
by (*metis UnE*)
next
case (*Future-mltl-ext a b L* α)
let *?D* = *LP-mltl-aux* (*convert-nnf-ext* α) *k*
let *?s* = *interval-times a L*
have *convert-nnf* (*to-mltl* φ) = *Future-mltl a b* (*to-mltl* α)
using *Future-mltl-ext convert-nnf-and-convert-nnf-ext*
by (*simp add*: *convert-nnf-ext-to-mltl-commute*)
then have *a-leq-b*: $a \leq b$
using *Suc* (*2*) *Future-mltl-ext nnf-intervals-welldef*
by *fastforce*
from *is-composition-convert-nnf-ext*[*OF Suc*(*2*) *Suc*(*4*)]
have *is-composition-MLTL* (*convert-nnf-ext* φ)
.
then have *is-comp*: *is-composition* ($b-a+1$) *L*
unfolding *Future-mltl-ext is-composition-MLTL.simps* **by** *blast*
{**assume** $*$: $\psi \in$ *set* (*Future-mltl-list ?D* (*?s ! 0*) (*?s ! 1 − 1*) [*?s ! 1 − ?s ! 0*])
then obtain x **where** ψ*-is*: ψ = *Future-mltl-ext* (*?s ! 0*) (*?s ! 1 − 1*) [*?s ! 1 − ?s ! 0*] x
and *x-in*: $x \in$ *set ?D*
unfolding *Future-mltl-list.simps* **by** *fastforce*
from *is-comp* **have** *welldef*: *?s ! 0* $\leq$ *?s ! 1 −1*
using *interval-times-diff-ge*[*OF a-leq-b is-comp - , of 0 ?s*]
by (*metis a-leq-b add-0 add-le-imp-le-diff gr-zeroI interval-times-first interval-times-last less-iff-succ-less-eq order-less-irrefl*)
have *ih*: *intervals-welldef* (*to-mltl x*)
using *Suc x-in*
by (*metis Future-mltl-ext convert-nnf-ext-to-mltl-commute intervals-welldef.simps*(*7*)

nnf-intervals-welldef is-composition-MLTL.simps(5) *is-composition-convert-nnf-ext to-mltl.simps*(7))
then have *?thesis*
unfolding ψ*-is to-mltl.simps intervals-welldef.simps*
using *welldef ih* **by** *blast*
} moreover {
assume $*$: $\psi \in$ *set* (*concat* (*map* (λi. *And-mltl-list*
[*Global-mltl-ext* (*?s* ! 0)
(*?s* ! $i - 1$) [*?s*!i−*?s*!0] (Not_c α)]
(*Future-mltl-list ?D* (*?s* ! i) (*?s* ! $(i + 1) - 1$)
[*?s* ! $(i + 1)$ − *?s* ! i]))
[1..<*length L*]))
then obtain i **where** ψ*-is*: $\psi \in$ *set* ((*And-mltl-list*
[*Global-mltl-ext* (*?s* ! 0)
(*?s* ! $i - 1$) [*?s*!i−*?s*!0] (Not_c α)]
(*Future-mltl-list ?D* (*?s* ! i) (*?s* ! $(i + 1) - 1$)
[*?s* ! $(i + 1)$ − *?s* ! i])
))
and *i-in*: $i \in \{1..<length\ L\}$
by *force*
then obtain x **where** ψ*-is*: ψ = ((*And-mltl-ext*
(*Global-mltl-ext* (*?s* ! 0)
(*?s* ! $i - 1$) [*?s*!i−*?s*!0] (Not_c α))
(*Future-mltl-ext* (*?s* ! i) (*?s* ! $(i + 1) - 1$)
[*?s* ! $(i + 1)$ − *?s* ! i] x)))
and *x-in*: $x \in$ *set ?D*
by *auto*
from *is-comp* **have** *welldef1*: *interval-times a L* ! 0 ≤ *interval-times a L* ! i − 1
using *i-in*
using *interval-times-diff-ge-general*[*OF a-leq-b is-comp - , of i 0 ?s*]
by *force*
have *welldef2*: *interval-times a L* ! i ≤ *interval-times a L* ! $(i + 1)$ − 1
using *i-in*
by (*metis a-leq-b add.commute add-le-imp-le-diff atLeastLessThan-iff interval-times-diff-ge is-comp less-eq-Suc-le plus-1-eq-Suc*)

have *ih1*: *intervals-welldef* (*to-mltl* α)
using *Suc x-in*
by (*metis* ‹*convert-nnf* (*to-mltl* φ) = *Future-mltl a b* (*to-mltl* α)› *intervals-welldef.simps*(7) *nnf-intervals-welldef*)
have *ih2*: *intervals-welldef* (*to-mltl x*)
using *Suc*
by (*metis Future-mltl-ext* ‹*is-composition-MLTL* (*convert-nnf-ext* φ)› *ih1 is-composition-MLTL.simps*(5) *x-in*)
have *?thesis* **unfolding** ψ*-is to-mltl.simps intervals-welldef.simps*
using *ih1 ih2 welldef1 welldef2*
by *auto*
}

ultimately show *?thesis*
using *Suc(3)* **unfolding** *Future-mltl-ext LP-mltl-aux.simps*
using *list-concat-set-union*
by (*metis* (*no-types*, *lifting*) *Un-iff*)
next
case (*Global-mltl-ext a b L α*)
let *?D-φ* = *LP-mltl-aux* (*convert-nnf-ext α*) *k*
have *nnf-φ*: *convert-nnf* (*to-mltl φ*) = *Global-mltl a b* (*to-mltl α*)
using *Global-mltl-ext convert-nnf-and-convert-nnf-ext*
by (*simp add*: *convert-nnf-ext-to-mltl-commute*)
then have *a-leq-b*: *a* ≤ *b*
using *Suc* (*2*) *Global-mltl-ext nnf-intervals-welldef*
by *fastforce*
have *α-composition*: *is-composition-MLTL α*
using *Suc(4) Global-mltl-ext Suc.prems(1) is-composition-convert-nnf-ext* **by** *fastforce*
have *L-composition*: *is-composition* (*b*−*a*+*1*) *L*
by (*metis Global-mltl-ext Suc.prems(1) Suc.prems(3) is-composition-MLTL.simps(3) is-composition-convert-nnf-ext*)
{assume ∗: *length ?D-φ* ≤ *1*
then have *ψ*: *ψ* = *Global-mltl-ext a b L α*
using *Suc(3)*
unfolding *Global-mltl-ext LP-mltl-aux.simps*
by *simp*
have *ih1*: *intervals-welldef* (*to-mltl α*)
using *Suc nnf-φ*
by (*metis intervals-welldef.simps(8) nnf-intervals-welldef*)
then have *?thesis*
using *a-leq-b* **unfolding** *ψ to-mltl.simps*
intervals-welldef.simps **by** *auto*
} moreover {assume ∗: *length ?D-φ* > *1*
then have *ψ-in*: *ψ* ∈ *set* (*Global-mltl-decomp ?D-φ a* (*b* − *a*) *L*)
using *Suc(3)*
unfolding *Global-mltl-ext LP-mltl-aux.simps*
by *simp*
then obtain *X* **where** *ψ-is*: *ψ* = *Ands-mltl-ext X* **and**
X-fact: (∀ *x* ∈ *set X*.
(∃ *y*∈*set* (*LP-mltl-aux* (*convert-nnf-ext α*) *k*).
∃ *k*≥*a*. *k* ≤ *a* + (*b* − *a*) ∧ *x* = *Global-mltl-ext k k* [*1*] *y*))
and *length-X*: *length X* = *Suc* (*b* − *a*)
using *in-Global-mltl-decomp*[*OF* ∗ *ψ-in*]
unfolding *List.member-def* **by** *blast*
have *X-ih*: *intervals-welldef* (*to-mltl x*)
if *x-in*: *x* ∈ *set X* **for** *x*
proof−
obtain *y k* **where** *y-in*: *y* ∈ *set ?D-φ*
and *k-bound*: *a* ≤ *k* ∧ *k* ≤ *b*
and *x-is*: *x* = *Global-mltl-ext k k* [*1*] *y*
using *X-fact a-leq-b x-in* **by** *fastforce*

```
      show ?thesis using y-in Suc
        unfolding x-is to-mltl.simps intervals-welldef.simps
      by (metis Global-mltl-ext intervals-welldef.simps(8) is-composition-MLTL.simps(3)
is-composition-convert-nnf-ext nnf-φ nnf-intervals-welldef order-refl)
    qed
    have ?thesis
      using ψ-is X-ih length-X
    proof(induct b−a arbitrary: b a ψ X)
      case 0
      then obtain x where X-is: X = [x]
        by (metis length-0-conv length-Suc-conv)
      have ψ = x
        using Ands-mltl-ext.simps(2) 0
        by (metis X-is append-self-conv2 length-0-conv)
      then show ?case using 0(3)[of x] unfolding X-is by auto
    next
      case (Suc n)
      then have length X = n + 2 by linarith
     then obtain H t where X-is: X = H@[t] and length-H: length H = length
X−1
        by (metis Suc.prems(3) diff-Suc-1 length-Suc-conv-rev)
      have ψ-is: ψ = And-mltl-ext (Ands-mltl-ext H) t
        using Suc(3) unfolding X-is Ands-mltl-ext.simps using length-H
      by (metis One-nat-def Suc.hyps(2) Suc.prems(3) diff-Suc-1′ nat.distinct(1))

      have t-ih: intervals-welldef (to-mltl t)
        using X-is Suc by force
      have (⋀x. x ∈ set H ⟹ intervals-welldef (to-mltl x))
        using Suc.prems unfolding X-is by auto
      then have H-ih: intervals-welldef (to-mltl (Ands-mltl-ext H))
        using Suc.hyps(1)[of - - Ands-mltl-ext H H]
        by (metis Suc.hyps(2) Suc.prems(3) diff-Suc-1 length-H)
      show ?case unfolding ψ-is to-mltl.simps
        using t-ih H-ih by simp
    qed
  }
  ultimately show ?thesis
    by linarith
next
  case (Until-mltl-ext α a b L β)
  let ?D-β = LP-mltl-aux (convert-nnf-ext β) k
  let ?s = interval-times a L
  have a-leq-b: a ≤ b using Suc(2)
    by (metis Until-mltl-ext convert-nnf-ext-to-mltl-commute intervals-welldef.simps(9)
to-mltl.simps(9) nnf-intervals-welldef)
  have composition: is-composition-MLTL (Until-mltl-ext α a b L β)
      using Suc(4) Until-mltl-ext
      by (metis Suc.prems(1) is-composition-convert-nnf-ext)
  have interval-composition: is-composition (b − a + 1) L
```

```
    using composition by simp
  have length-L: 0 < length L
    using interval-composition
    by (meson add-gr-0 composition-length-lb less-numeral-extra(1))
  have α-ih: intervals-welldef (to-mltl α)
    using Suc Until-mltl-ext convert-nnf-ext-to-mltl-commute
   by (metis  intervals-welldef.simps(9) to-mltl.simps(9) nnf-intervals-welldef)
  have β-ih: intervals-welldef (to-mltl β)
    using Suc(2) Until-mltl-ext
  by (metis convert-nnf-ext-to-mltl-commute intervals-welldef.simps(9) to-mltl.simps(9)
nnf-intervals-welldef)
  {assume ∗: ψ ∈ set (Until-mltl-list α ?D-β (?s ! 0) (?s ! 1 − 1) [?s ! 1 − ?s
! 0])
   then obtain x where ψ-is: ψ = Until-mltl-ext α (?s!0) (?s!1−1) [?s!1−?s!0]
x
                  and x-in: x ∈ set (?D-β)
    by auto
   have fact1: interval-times a L ! 0 ≤ interval-times a L ! 1 − 1
    unfolding is-composition-def
   using interval-times-diff-ge[OF a-leq-b interval-composition length-L, of ?s]
    by auto
   have x-ih: intervals-welldef (to-mltl x)
    using x-in Suc.hyps[of β x] Suc.prems
    using β-ih composition is-composition-MLTL.simps(6) by blast
   have ?thesis unfolding ψ-is unfolding to-mltl.simps
    unfolding intervals-welldef.simps
    using fact1 α-ih x-ih by blast
  } moreover {
   assume ∗: ψ ∈ set (concat
          (map (λi. And-mltl-list
                  [Global-mltl-ext
                    (?s ! 0) (?s ! i − 1) [?s!i − ?s!0] (And-mltl-ext α (Not_c
β))]
                  (Until-mltl-list α ?D-β (?s ! i) (?s ! (i + 1) − 1)
                    [?s ! (i + 1) − ?s ! i]))
            [1..<length L]))
   then obtain i x where
  ψ-is: ψ = And-mltl-ext (Global-mltl-ext (?s!0) (?s!i−1) [?s!i − ?s!0] (And-mltl-ext
α (Not_c β)))
        (Until-mltl-ext α (?s!i) (?s!(i+1)−1) [(?s!(i+1)) − (?s!i)] x)
    and i-bound: 1 ≤ i ∧ i < length L
    and x-in: x ∈ set ?D-β
    by auto
   have fact1: interval-times a L ! 0 ≤ interval-times a L ! i − 1
    using i-bound a-leq-b
    using interval-times-diff-ge-general[OF a-leq-b interval-composition, of i 0
?s]
    by force
   have fact2: interval-times a L ! i ≤ interval-times a L ! (i + 1) − 1
```

```
      using i-bound
      using interval-times-diff-ge[OF a-leq-b interval-composition, of i ?s]
      by auto
    have x-ih: intervals-welldef (to-mltl x)
      using Suc.hyps β-ih composition is-composition-MLTL.simps(6) x-in by
blast
    have ?thesis unfolding ψ-is to-mltl.simps
      unfolding intervals-welldef.simps
      using fact1 fact2 α-ih β-ih x-ih by blast
  }
  ultimately show ?thesis using Suc(3) list-concat-set-union
    unfolding Until-mltl-ext LP-mltl-aux.simps
    by (metis (mono-tags, lifting) UnE)
next
  case (Release-mltl-ext α a b L β)
  let ?D = LP-mltl-aux (convert-nnf-ext α) k
  let ?s = interval-times a L
  have α-ih: intervals-welldef (to-mltl α)
    using Suc(2) Release-mltl-ext convert-nnf-ext-to-mltl-commute
   by (metis intervals-welldef.simps(10) to-mltl.simps(10) nnf-intervals-welldef)
  have β-ih: intervals-welldef (to-mltl β)
    using Suc(2) Release-mltl-ext convert-nnf-ext-to-mltl-commute
   by (metis intervals-welldef.simps(10) to-mltl.simps(10) nnf-intervals-welldef)
  have a-leq-b: a ≤ b using Suc(2) Release-mltl-ext
  by (metis convert-nnf-ext-to-mltl-commute intervals-welldef.simps(10) to-mltl.simps(10)
nnf-intervals-welldef)
  have composition: is-composition-MLTL (Release-mltl-ext α a b L β)
    using Suc.prems(3) Release-mltl-ext
    by (metis Suc.prems(1) is-composition-convert-nnf-ext)
  then have composition-L: is-composition (b−a+1) L
      and composition-α: is-composition-MLTL α
      and composition-β: is-composition-MLTL β
    unfolding is-composition-MLTL.simps by simp-all
  have length-L: length L > 0
    using composition-length-lb composition-L by auto
  have sfirst: ?s!0 = a
    using interval-times-first by simp
  have slast: ?s!(length L) = b+1
    using interval-times-last[OF a-leq-b composition-L] by blast
  let ?front = set [Global-mltl-ext a b L (And-mltl-ext (Not_c α) β)]
  let ?middle = set (Mighty-Release-mltl-list ?D β (?s ! 0) (?s ! 1 − 1)
          [?s ! 1 − ?s ! 0])
  let ?back = set (concat (map (λi. And-mltl-list
                  [Global-mltl-ext
                    (?s ! 0)
                    (?s ! i − 1) [?s!i − ?s!0] (And-mltl-ext (Not_c α) β)]
                  (Mighty-Release-mltl-list ?D β (?s ! i)
                    (?s ! (i + 1) − 1) [?s ! (i + 1) − ?s ! i]))
            [1..<length L]))
```

have *split*: $\psi \in$ *?front* $\cup$ *?middle* $\cup$ *?back*
using *Suc*(3) **unfolding** *Release-mltl-ext LP-mltl-aux.simps*
using *list-concat-set-union*
by (*metis append.assoc*)
{
assume $*$: $\psi \in$ *?front*
then have ψ-*is*: ψ = *Global-mltl-ext a b L* (*And-mltl-ext* (*Not*$_c$ α) β)
by *auto*
have *?thesis* **unfolding** ψ-*is to-mltl.simps intervals-welldef.simps*
using α-*ih* β-*ih a-leq-b* **by** *blast*
} moreover {
assume $*$: $\psi \in$ *?middle*
then obtain *x* **where** ψ-*is*: ψ = *Mighty-Release-mltl-ext x* β
(*interval-times a L* ! *0*) (*interval-times a L* ! *1* − *1*)
[*interval-times a L* ! *1* − *interval-times a L* ! *0*]
and *x-in*: *x* ∈ *set ?D*
by *auto*
have *x-ih*: *intervals-welldef* (*to-mltl x*)
using *Suc*(*1*)[*OF* α-*ih x-in composition-*α] **by** *blast*
have *welldef*: *interval-times a L* ! *0* ≤ *interval-times a L* ! *1* − *1*
using *interval-times-diff-ge*[*OF a-leq-b composition-L*, *of 0 ?s*]
using *length-L* **by** *auto*
then have *?thesis* **unfolding** ψ-*is to-mltl.simps Mighty-Release-mltl-ext.simps intervals-welldef.simps*
using *x-ih* α-*ih* β-*ih* **by** *blast*
} moreover {
assume $*$: $\psi \in$ *?back*
then obtain *i x* **where** ψ-*is*: ψ = *And-mltl-ext*
(*Global-mltl-ext*
(*interval-times a L* ! *0*)
(*interval-times a L* ! *i* − *1*) [*?s*!*i* − *?s*!*0*] (*And-mltl-ext* (*Not*$_c$ α) β))
(*Mighty-Release-mltl-ext x* β
(*interval-times a L* ! *i*)
(*interval-times a L* ! (*i* + *1*) − *1*)
[*interval-times a L* ! (*i* + *1*) −
interval-times a L ! *i*])
and *x-in*: *x* ∈ *set ?D*
and *i-bound*: *1* ≤ *i* ∧ *i* < *length L*
by *auto*
have *lb*: *a* < *?s*!*i*
using *interval-times-diff-ge-general*[*OF a-leq-b composition-L*, *of i 0 ?s*]
using *sfirst i-bound* **by** *simp*
have *welldef*: (*interval-times a L* ! *i*) < (*interval-times a L* ! (*i* + *1*))
using *interval-times-diff-ge*[*OF a-leq-b composition-L*, *of i ?s*]
using *i-bound* **by** *simp*
have *ub*: *?s*!(*i*+*1*) ≤ *b*+*1*
using *slast i-bound*
using *interval-times-diff-ge-general*[*OF a-leq-b composition-L*, *of length L*

i+1 ?s]
by (*metis Orderings.order-eq-iff less-iff-succ-less-eq order-le-imp-less-or-eq order-less-imp-le*)
have *x-ih*: *intervals-welldef* (*to-mltl x*)
using *Suc*(*1*)
using α*-ih composition-*α *x-in* **by** *blast*
have *?thesis* **unfolding** ψ*-is to-mltl.simps intervals-welldef.simps Mighty-Release-mltl-ext.simps*
using *x-ih* α*-ih* β*-ih ub lb welldef*
by (*simp add*: *add-le-imp-le-diff sfirst*)
}
ultimately show *?thesis*
using *Suc*(*3*) **unfolding** *Release-mltl-ext LP-mltl-aux.simps*
using *split* **by** *blast*
qed
qed

lemma *LP-mltl-aux-wpd*:
assumes $\exists\,\varphi$*-init.* φ = *convert-nnf-ext* φ*-init*
assumes *intervals-welldef* (*to-mltl* φ)
assumes $\psi \in$ *set* (*LP-mltl-aux* φ *k*)
assumes *is-composition-MLTL* φ
shows *wpd-mltl* (*to-mltl* ψ) $\leq$ *wpd-mltl* (*to-mltl* φ)
using *assms*
proof(*induct k arbitrary*: φ ψ)
case *0*
then show *?case* **by** *auto*
next
case (*Suc k*)
then show *?case*
proof(*cases* φ)
case *True-mltl-ext*
then show *?thesis* **using** *Suc* **by** *auto*
next
case *False-mltl-ext*
then show *?thesis* **using** *Suc* **by** *auto*
next
case (*Prop-mltl-ext p*)
then show *?thesis* **using** *Suc* **by** *auto*
next
case (*Not-mltl-ext q*)
then have $\exists\,p.$ *q* = *Prop-mltl-ext p*
using *convert-nnf-form-Not-Implies-Prop Suc*
by (*metis convert-nnf-ext-to-mltl-commute to-mltl.simps*(*4*) *to-mltl-prop-bijective*)

then obtain *p* **where** *q* = *Prop-mltl-ext p* **by** *blast*
then show *?thesis*
using *Not-mltl-ext Suc.prems*(*3*) **by** *fastforce*
next

case (*And-mltl-ext* α β)
obtain x y **where** ψ*-is*: ψ = *And-mltl-ext* x y
and *x-in*: $x \in$ *set* (*LP-mltl-aux* α k)
and *y-in*: $y \in$ *set* (*LP-mltl-aux* β k)
using *Suc* **unfolding** *And-mltl-ext LP-mltl-aux.simps*
by (*metis And-mltl-list-member convert-nnf-ext.simps*(4) *convert-nnf-ext-convert-nnf-ext in-set-member mltl-ext.inject*(3))
have α*-nnf*: $\exists \varphi$*-init*. α = *convert-nnf-ext* φ*-init*
using *Suc*(2) **unfolding** *And-mltl-ext*
by (*metis convert-nnf-ext.simps*(4) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(3))

have β*-nnf*: $\exists \varphi$*-init*. β = *convert-nnf-ext* φ*-init*
using *Suc*(2) **unfolding** *And-mltl-ext*
by (*metis convert-nnf-ext.simps*(4) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(3))

have α*-welldef*: *intervals-welldef* (*to-mltl* α) **and**
β*-welldef*: *intervals-welldef* (*to-mltl* β)
using *Suc*(3) **unfolding** *And-mltl-ext* **by** *simp-all*
have α*-composition*: *is-composition-MLTL* α **and**
β*-composition*: *is-composition-MLTL* β
using *Suc*(5) **unfolding** *And-mltl-ext is-composition-MLTL.simps* **by** *simp-all*
have *x-ih*: *wpd-mltl* (*to-mltl* x) $\leq$ *wpd-mltl* (*to-mltl* α)
using *Suc.hyps*[*of* α x, *OF* α*-nnf* α*-welldef x-in* α*-composition*] **by** *blast*
have *y-ih*: *wpd-mltl* (*to-mltl* y) $\leq$ *wpd-mltl* (*to-mltl* β)
using *Suc.hyps*[*of* β y, *OF* β*-nnf* β*-welldef y-in* β*-composition*] **by** *blast*
show *?thesis*
unfolding *And-mltl-ext* ψ*-is to-mltl.simps wpd-mltl.simps*
using *x-ih y-ih* **by** *linarith*
next
case (*Or-mltl-ext* α β)
let *?Dx* = *LP-mltl-aux* α k
let *?Dy* = *LP-mltl-aux* β k
have α*-nnf*: $\exists \varphi$*-init*. α = *convert-nnf-ext* φ*-init*
using *Suc*(2) **unfolding** *Or-mltl-ext*
by (*metis convert-nnf-ext.simps*(5) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(4))

have β*-nnf*: $\exists \varphi$*-init*. β = *convert-nnf-ext* φ*-init*
using *Suc*(2) **unfolding** *Or-mltl-ext*
by (*metis convert-nnf-ext.simps*(5) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(4))

have α*-welldef*: *intervals-welldef* (*to-mltl* α) **and**
β*-welldef*: *intervals-welldef* (*to-mltl* β)
using *Suc*(3) **unfolding** *Or-mltl-ext* **by** *simp-all*
have α*-composition*: *is-composition-MLTL* α **and**
β*-composition*: *is-composition-MLTL* β
using *Suc*(5) **unfolding** *Or-mltl-ext is-composition-MLTL.simps* **by** *simp-all*
{
assume $*$: $\psi \in$ *set* (*And-mltl-list ?Dx ?Dy*)
then obtain x y **where** ψ*-is*: ψ = *And-mltl-ext* x y

```
          and x-in: x ∈ set ?Dx and y-in: y ∈ set ?Dy
      using And-mltl-list-member[of ψ ?Dx ?Dy]
      by (metis in-set-member)
    have x-ih: wpd-mltl (to-mltl x) ≤ wpd-mltl (to-mltl α)
      using Suc.hyps[of α x, OF α-nnf α-welldef x-in α-composition] by blast
    have y-ih: wpd-mltl (to-mltl y) ≤ wpd-mltl (to-mltl β)
      using Suc.hyps[of β y, OF β-nnf β-welldef y-in β-composition] by blast

    have ?thesis
      unfolding Or-mltl-ext ψ-is to-mltl.simps wpd-mltl.simps
      using x-ih y-ih by linarith
  } moreover {
    assume ∗: ψ ∈ set (And-mltl-list [Not_c α] ?Dy)
    then obtain y where ψ-is: ψ = And-mltl-ext (Not_c α) y
                    and y-in: y ∈ set ?Dy
      using And-mltl-list-member[of ψ [Not_c α] ?Dy]
      by auto
    have y-ih: wpd-mltl (to-mltl y) ≤ wpd-mltl (to-mltl β)
      using Suc.hyps[of β y, OF β-nnf β-welldef y-in β-composition] by blast

    have ?thesis
      unfolding Or-mltl-ext ψ-is to-mltl.simps wpd-mltl.simps
      using y-ih by auto
  } moreover {
    assume ∗: ψ ∈ set (And-mltl-list ?Dx [Not_c β])
    then obtain x where ψ-is: ψ = And-mltl-ext x (Not_c β)
                    and x-in: x ∈ set ?Dx
      using And-mltl-list-member[of ψ ?Dx [Not_c β]]
      by (metis in-set-member member-rec(1) member-rec(2))
    have x-ih: wpd-mltl (to-mltl x) ≤ wpd-mltl (to-mltl α)
     using Suc.hyps[of α x, OF α-nnf α-welldef x-in α-composition] by blast
    have ?thesis
      unfolding Or-mltl-ext ψ-is to-mltl.simps wpd-mltl.simps
      using x-ih by auto
  }
  ultimately show ?thesis
    using Suc unfolding Or-mltl-ext LP-mltl-aux.simps
    using list-concat-set-union
    by (metis UnE α-nnf β-nnf convert-nnf-ext-convert-nnf-ext)
next
  case (Future-mltl-ext a b L α)
  let ?D = LP-mltl-aux α k
  let ?s = interval-times a L
  let ?front = (Future-mltl-list ?D (?s ! 0) (?s ! 1 − 1) [?s ! 1 − ?s ! 0])
  let ?back = (concat (map (λi. And-mltl-list
                        [Global-mltl-ext (?s ! 0)
                          (?s ! i − 1) [?s!i − ?s!0] (Not_c α)]
                        (Future-mltl-list ?D (?s ! i) (?s ! (i + 1) − 1)
                          [?s ! (i + 1) − ?s ! i]))
```

```
            [1..<length L]))
  have a-leq-b: a ≤ b using Suc(3)
    unfolding Future-mltl-ext to-mltl.simps intervals-welldef.simps
    by blast
  have composition-L: is-composition (b−a+1) L and
       composition-α: is-composition-MLTL α using Suc(5)
    unfolding Future-mltl-ext is-composition-MLTL.simps by simp-all
  have α-nnf: ∃φ-init. α = convert-nnf-ext φ-init
    using Suc(2) unfolding Future-mltl-ext
  by (metis convert-nnf-ext.simps(6) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(5))

  have α-welldef: intervals-welldef (to-mltl α)
    using Suc(3) unfolding Future-mltl-ext by simp
  have nnf: convert-nnf-ext α = α
    using α-nnf convert-nnf-ext-convert-nnf-ext by metis
  have slast: interval-times a L ! (length L) = b+1
     using interval-times-last[OF a-leq-b composition-L] by blast
  then have split: ψ ∈ (set ?front) ∪ (set ?back)
    using Suc(4) unfolding Future-mltl-ext LP-mltl-aux.simps nnf
    using list-concat-set-union[of ?front ?back] by metis
  {
    assume ∗: ψ ∈ set ?front
    then obtain x where ψ-is: ψ = Future-mltl-ext (?s ! 0) (?s ! 1 − 1) [?s ! 1
− ?s ! 0] x
                  and x-in: x ∈ set ?D
      unfolding Future-mltl-list.simps by fastforce
    have length-s: 1 < length ?s using ψ-is
       by (metis One-nat-def add.commute add-gr-0 add-less-cancel-right com-
position-L composition-length-lb interval-times-length plus-1-eq-Suc zero-less-one)

    then have length-L: 1 ≤ length L
      unfolding interval-times-def
      by (simp add: less-eq-iff-succ-less)
    have interval-times a L ! 1 ≤ interval-times a L ! (length L)
      using interval-times-diff-ge-general[OF a-leq-b composition-L, of length L 1
?s]
      using length-L by force
    then have bound: interval-times a L ! 1 − 1 ≤ b
      using slast by auto
    have ih: wpd-mltl (to-mltl x) ≤ wpd-mltl (to-mltl α)
      using Suc(1)[OF α-nnf α-welldef x-in composition-α] by blast
    have ?thesis
      unfolding ψ-is Future-mltl-ext to-mltl.simps wpd-mltl.simps
      using bound ih by simp
  } moreover {
    assume ∗: ψ ∈ set ?back
    then obtain i where ψ-is: ψ ∈ set ((And-mltl-list
                    [Global-mltl-ext (?s ! 0)
                      (?s ! i − 1) [?s!i − ?s!0] (Not_c α)]
```

```
                    (Future-mltl-list ?D (?s ! i) (?s ! (i + 1) - 1)
                      [?s ! (i + 1) - ?s ! i])
              ))
      and i-in: i ∈ {1..<length L}
      by force
    then obtain x where ψ-is: ψ = ((And-mltl-ext
                    (Global-mltl-ext (?s ! 0)
                      (?s ! i - 1) [?s!i - ?s!0] (Not_c α))
                    (Future-mltl-ext (?s ! i) (?s ! (i + 1) - 1)
                      [?s ! (i + 1) - ?s ! i] x)))
      and x-in: x ∈ set ?D
      by auto
    have ih: wpd-mltl (to-mltl x) ≤ wpd-mltl (to-mltl α)
      using Suc.hyps(1)[OF α-nnf α-welldef x-in composition-α] by blast
    have bound: interval-times a L ! i < interval-times a L ! (i + 1)
      using interval-times-diff-ge[OF a-leq-b composition-L, of i ?s]
      using i-in by simp
    have (interval-times a L ! (i + 1) - 1) ≤ b using slast
      using interval-times-diff-ge-general[OF a-leq-b composition-L, of length L
i+1 ?s] i-in
    by (metis Suc-eq-plus1 atLeastLessThan-iff le-Suc-eq le-diff-conv linorder-not-less
order-less-imp-le verit-comp-simplify1(2))
    then have ?thesis
      unfolding ψ-is Future-mltl-ext to-mltl.simps wpd-mltl.simps
      using ih bound by linarith
  }
  ultimately show ?thesis using split by blast
 next
  case (Global-mltl-ext a b L α)
  let ?D-α = LP-mltl-aux α k
  have a-leq-b: a ≤ b and α-welldef: intervals-welldef (to-mltl α)
    using Suc(3)
    unfolding Global-mltl-ext to-mltl.simps intervals-welldef.simps
    by simp-all
  have composition-α: is-composition-MLTL α using Suc(5)
    unfolding Global-mltl-ext is-composition-MLTL.simps by simp-all
  have α-nnf: ∃φ-init. α = convert-nnf-ext φ-init
    using Suc(2) unfolding Global-mltl-ext
  by (metis convert-nnf-ext.simps(7) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(6))

  have α-welldef: intervals-welldef (to-mltl α)
    using Suc(3) unfolding Global-mltl-ext by simp
  have nnf: convert-nnf-ext α = α
    using α-nnf convert-nnf-ext-convert-nnf-ext by metis
  {
    assume *: length ?D-α ≤ 1
    then have ψ-is: ψ = Global-mltl-ext a b L α
      using Suc unfolding Global-mltl-ext LP-mltl-aux.simps
      using nnf by fastforce
```

```
    have ?thesis unfolding ψ-is Global-mltl-ext by simp
  } moreover {
    assume ∗: length ?D-α > 1
    then have ψ-in: ψ ∈ set (Global-mltl-decomp ?D-α a (b − a) L)
      using Suc nnf unfolding Global-mltl-ext LP-mltl-aux.simps
      by simp
    then obtain X where ψ-is: ψ = Ands-mltl-ext X
                  and X-fact: ∀ i<length X. ∃ y∈set (LP-mltl-aux α k).
                                X ! i = Global-mltl-ext (a + i) (a + i) [1] y
                  and length-X: length X = Suc (b − a)
      using in-Global-mltl-decomp-exact-forward[OF ∗ ψ-in] nnf a-leq-b
      unfolding List.member-def by blast
    have X-ih: wpd-mltl (to-mltl (X!i)) ≤ b+wpd-mltl (to-mltl α)
      if i-bound: i < length X for i
    proof−
      obtain x where x-in: x ∈ set ?D-α
                   and Xi-is: X!i = Global-mltl-ext (a+i) (a+i) [1] x
        using X-fact a-leq-b i-bound by blast
      have wpd-mltl (to-mltl x) ≤ wpd-mltl (to-mltl α)
        using Suc.hyps[OF α-nnf α-welldef x-in composition-α] by simp
      then show ?thesis unfolding Xi-is to-mltl.simps wpd-mltl.simps
        using a-leq-b length-X i-bound by auto
    qed
    have ?thesis
      unfolding ψ-is Global-mltl-ext to-mltl.simps wpd-mltl.simps
      using X-ih length-X X-fact Suc(1)
    proof(induct b−a arbitrary:X a b)
      case 0
      then have length X = 1
        by simp
      then obtain x where X-is: X = [x]
        by (metis One-nat-def Suc-length-conv length-0-conv)
      show ?case using 0(2)[of 0] unfolding X-is
        using Ands-mltl-ext.simps(2)
     by (metis X-is ‹length X = 1› length-0-conv less-one nth-Cons' self-append-conv2)

    next
      case (Suc n)
      then have length-X: length X = n + 2 by linarith
     then obtain H t where X-is: X = H@[t] and length-H: length H = length
X−1
        by (metis Suc.prems(2) diff-Suc-1 length-Suc-conv-rev)
      have Ands: Ands-mltl-ext X = And-mltl-ext (Ands-mltl-ext H) t
        unfolding X-is Ands-mltl-ext.simps using length-H length-X by simp
      have t-bound: (wpd-mltl (to-mltl t)) ≤ b + wpd-mltl (to-mltl α)
        using Suc(3)[of length X−1] unfolding X-is by auto
      have cond1: n = b − 1 − a using Suc by auto
      have cond2: wpd-mltl (to-mltl (H ! i))
                  ≤ b + wpd-mltl (to-mltl α)−1
```

```
      if i-bound: i < length H for i
    proof−
      have Hi-is: H!i = X!i using X-is i-bound
        by (simp add: nth-append)
      have ∃ y∈set (LP-mltl-aux α k). X ! i = Global-mltl-ext (a + i) (a + i)
[1] y
        using Suc(3)[of i] Suc(5) i-bound
      by (metis Suc.prems(2) add-diff-cancel-left' length-H less-Suc-eq plus-1-eq-Suc)

      then obtain y where Xi-is: X ! i = Global-mltl-ext (a + i) (a + i) [1] y
                      and y-in: y ∈ set (LP-mltl-aux α k)
        by auto
      have ih: wpd-mltl (to-mltl (X ! i)) ≤ b + wpd-mltl (to-mltl α)
        using i-bound Suc(3)[of i] X-is by auto
      have bound: a+i < b
        using i-bound length-H length-X
        by (simp add: Suc.prems(2))
      have wpd-mltl (to-mltl y) ≤ wpd-mltl (to-mltl α)
        using Suc(6)[OF α-nnf α-welldef y-in composition-α] by blast
      then show ?thesis unfolding Hi-is Xi-is to-mltl.simps wpd-mltl.simps
        using bound by simp
    qed
    have cond3: length H = Suc (b − 1 − a)
      using length-H length-X Suc.hyps(2) by simp
    have cond4: ∃ y∈set (LP-mltl-aux α k). H ! i = Global-mltl-ext (a + i) (a
+ i) [1] y
      if i-bound: i<length H for i
    proof−
      have ∃ y∈set (LP-mltl-aux α k). X ! i = Global-mltl-ext (a + i) (a + i)
[1] y
        using Suc(5) i-bound length-H by auto
      then obtain y where y-in: y∈set (LP-mltl-aux α k) and
                          Xi-is: X ! i = Global-mltl-ext (a + i) (a + i) [1] y
        by blast
      then have Hi-is: H!i = X!i using i-bound length-H
        by (metis X-is nth-append)
      then show ?thesis unfolding Xi-is using y-in by blast
    qed
    have ih: wpd-mltl (to-mltl (Ands-mltl-ext H))
  ≤ b − 1 + wpd-mltl (to-mltl α)
     using Suc.hyps(1)[of b−1 a H, OF cond1 - cond3] cond2 cond4 Suc.prems(4)
      by force
     show ?case unfolding Ands wpd-mltl.simps to-mltl.simps
      using t-bound ih by simp
   qed
  }
  ultimately show ?thesis by linarith
 next
  case (Until-mltl-ext α a b L β)
```

let *?D-α = LP-mltl-aux α k*
let *?D-β = LP-mltl-aux β k*
let *?s = interval-times a L*
have *a-leq-b*: $a \leq b$ **and** *α-welldef*: *intervals-welldef* (*to-mltl α*)
and *β-weldef*: *intervals-welldef* (*to-mltl α*)
using *Suc(3)*
unfolding *Until-mltl-ext to-mltl.simps intervals-welldef.simps*
by *simp-all*
have *composition-α*: *is-composition-MLTL α* **and**
composition-β: *is-composition-MLTL β* **and**
composition-L: *is-composition* (*b−a+1*) *L* **using** *Suc(5)*
unfolding *Until-mltl-ext is-composition-MLTL.simps* **by** *simp-all*
have *α-nnf*: ∃ *φ-init*. *α = convert-nnf-ext φ-init*
using *Suc(2)* **unfolding** *Until-mltl-ext*
by (*metis convert-nnf-ext.simps(8) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(7)*)

have *β-nnf*: ∃ *φ-init*. *β = convert-nnf-ext φ-init*
using *Suc(2)* **unfolding** *Until-mltl-ext*
by (*metis convert-nnf-ext.simps(8) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(7)*)
have *α-welldef*: *intervals-welldef* (*to-mltl α*) **and**
β-welldef: *intervals-welldef* (*to-mltl β*)
using *Suc(3)* **unfolding** *Until-mltl-ext* **by** *simp-all*
have *convert-α*: *convert-nnf-ext α = α*
by (*metis α-nnf convert-nnf-ext-convert-nnf-ext*)
have *convert-β*: *convert-nnf-ext β = β*
by (*metis Suc.prems(1) Until-mltl-ext convert-nnf-ext.simps(8) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(7)*)
have *slast*: *interval-times a L* ! (*length L*) = *b+1*
using *interval-times-last*[*OF a-leq-b composition-L*] **by** *blast*
let *?front* = (*Until-mltl-list α ?D-β* (*?s* ! *0*) (*?s* ! *1* − *1*) [*?s* ! *1* − *?s* ! *0*])
let *?back* = (*concat* (*map* (λ*i*. *And-mltl-list*
[*Global-mltl-ext*
(*?s* ! *0*) (*?s* ! *i* − *1*) [*?s*!*i* − *?s*!*0*] (*And-mltl-ext α* (Not_c *β*))]
(*Until-mltl-list α ?D-β* (*?s* ! *i*) (*?s* ! (*i* + *1*) − *1*)
[*?s* ! (*i* + *1*) − *?s* ! *i*])) [*1*..<*length L*]))
have *split*: *ψ* ∈ (*set ?front*) ∪ (*set ?back*)
using *Suc(4)* **unfolding** *Until-mltl-ext LP-mltl-aux.simps*
using *convert-α convert-β list-concat-set-union* **by** *metis*
{
assume ∗: *ψ* ∈ *set ?front*
then obtain *y* **where** *ψ-is*: *ψ = Until-mltl-ext α* (*interval-times a L* ! *0*)
(*interval-times a L* ! *1* − *1*) [*interval-times a L* ! *1* − *interval-times a L* ! *0*] *y*
and *y-in*: *y* ∈ *set ?D-β*
by *auto*
have *length-s*: *1* < *length ?s* **using** *ψ-is*
by (*metis One-nat-def add.commute add-gr-0 add-less-cancel-right composition-L composition-length-lb interval-times-length plus-1-eq-Suc zero-less-one*)

then have *length-L*: $1 \leq$ *length L*
unfolding *interval-times-def*
by (*simp add*: *less-eq-iff-succ-less*)
have *interval-times a L* ! *1* $\leq$ *interval-times a L* ! (*length L*)
using *interval-times-diff-ge-general*[*OF a-leq-b composition-L*, *of length L 1 ?s*]
using *length-L* **by** *force*
then have *bound*: *interval-times a L* ! *1* − *1* $\leq$ *b*
using *slast* **by** *auto*
have β*-ih*: *wpd-mltl* (*to-mltl y*) $\leq$ *wpd-mltl* (*to-mltl* β)
using *Suc.hyps*(*1*)[*OF* β*-nnf* β*-welldef y-in composition-*β] **by** *blast*
have *?thesis*
unfolding ψ*-is Until-mltl-ext to-mltl.simps wpd-mltl.simps*
using β*-ih bound* **by** *linarith*
} moreover {
assume $*$: $\psi \in$ *set ?back*
then obtain *i y* **where**
ψ*-is*: ψ = *And-mltl-ext* (*Global-mltl-ext* (*?s*!*0*) (*?s*!*i*−*1*) [*?s*!*i* − *?s*!*0*] (*And-mltl-ext* α (*Not*$_c$ β)))
(*Until-mltl-ext* α (*?s*!*i*) (*?s*!(*i*+*1*)−*1*) [(*?s*!(*i*+*1*)) − (*?s*!*i*)] *y*)
and *i-bound*: *1* $\leq$ *i* $\wedge$ *i* < *length L*
and *y-in*: *y* $\in$ *set ?D-*β
by *auto*
have *bound1*: *interval-times a L* ! *i* < *interval-times a L* ! (*i*+*1*)
using *interval-times-diff-ge*[*OF a-leq-b composition-L*, *of i ?s*]
using *i-bound* **by** *blast*
have *interval-times a L* ! (*i* + *1*) $\leq$ *interval-times a L* ! (*length L*)
using *interval-times-diff-ge-general*[*OF a-leq-b composition-L*, *of length L i+1 ?s*]
using *i-bound* **by** (*metis less-iff-succ-less-eq order-le-less*)
then have *bound2*: *interval-times a L* ! (*i*+*1*) $\leq$ *b*+*1*
using *slast* **by** *simp*
have β*-ih*: *wpd-mltl* (*to-mltl y*) $\leq$ *wpd-mltl* (*to-mltl* β)
using *Suc.hyps*(*1*)[*OF* β*-nnf* β*-welldef y-in composition-*β] **by** *blast*
have *interval-times a L* ! *i* > *interval-times a L* ! *0*
using *i-bound interval-times-diff-ge-general*[*OF a-leq-b composition-L*, *of i 0 ?s*]
by *auto*
then have *interval-times a L* ! *i* > *0*
unfolding *interval-times-def* **by** *simp*
then have *b* > *interval-times a L* ! *i* − *1*
using *bound1 bound2* **by** *simp*
then have *case1*: (*interval-times a L* ! *i* − *1* +
max (*wpd-mltl* (*to-mltl* α))
(*wpd-mltl* (*to-mltl* β))) $\leq$
b + *max* (*wpd-mltl* (*to-mltl* α))
(*wpd-mltl* (*to-mltl* β))
using *bound1 bound2* β*-ih* **by** *linarith*

```
    have case2: (interval-times a L ! (i + 1) − 1 +
    max (wpd-mltl (to-mltl α))
     (wpd-mltl (to-mltl y))) ≤
         b + max (wpd-mltl (to-mltl α))
         (wpd-mltl (to-mltl β))
      using bound1 bound2 β-ih by linarith
    have ?thesis
      unfolding Until-mltl-ext ψ-is to-mltl.simps wpd-mltl.simps
      using case1 case2
      by presburger
   }
   ultimately show ?thesis using split by blast
  next
   case (Release-mltl-ext α a b L β)
   let ?D = LP-mltl-aux α k
   let ?s = interval-times a L
   have a-leq-b: a ≤ b and α-welldef: intervals-welldef (to-mltl α)
                    and β-weldef: intervals-welldef (to-mltl α)
    using Suc(3)
    unfolding Release-mltl-ext to-mltl.simps intervals-welldef.simps
     by simp-all
   have composition-α: is-composition-MLTL α and
       composition-β: is-composition-MLTL β and
       composition-L: is-composition (b−a+1) L using Suc(5)
    unfolding Release-mltl-ext is-composition-MLTL.simps by simp-all
   have α-nnf: ∃φ-init. α = convert-nnf-ext φ-init
    using Suc(2) unfolding Release-mltl-ext
   by (metis convert-nnf-ext.simps(9) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(8))

   have β-nnf: ∃φ-init. α = convert-nnf-ext φ-init
    using Suc(2) unfolding Release-mltl-ext
   by (metis convert-nnf-ext.simps(9) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(8))
   have α-welldef: intervals-welldef (to-mltl α) and
       β-welldef: intervals-welldef (to-mltl α)
    using Suc(3) unfolding Release-mltl-ext by simp-all
   have convert-α: convert-nnf-ext α = α
    by (metis α-nnf convert-nnf-ext-convert-nnf-ext)
   have convert-β: convert-nnf-ext β = β
   by (metis Suc.prems(1) Release-mltl-ext convert-nnf-ext.simps(9) convert-nnf-ext-convert-nnf-ext
mltl-ext.inject(8))
   have slast: interval-times a L ! (length L) = b+1
    using interval-times-last[OF a-leq-b composition-L] by blast
   have sfirst: ?s!0 = a
    using interval-times-first by blast
   have length-L: length L > 0
    using composition-length-lb composition-L by simp
   let ?front = set [Global-mltl-ext a b L (And-mltl-ext (Not_c α) β)]
   let ?middle = set (Mighty-Release-mltl-list ?D β (?s ! 0) (?s ! 1 − 1)
            [?s ! 1 − ?s ! 0])
```

```
  let ?back = set (concat
          (map (λi. And-mltl-list
                  [Global-mltl-ext
                    (?s ! 0)
                    (?s ! i − 1) [?s!i − ?s!0] (And-mltl-ext (Not_c α) β)]
                  (Mighty-Release-mltl-list ?D β (?s ! i)
                    (?s ! (i + 1) − 1) [?s ! (i + 1) − ?s ! i]))
            [1..<length L]))
  have split: ψ ∈ ?front ∪ ?middle ∪ ?back
    using Suc(4) unfolding Release-mltl-ext LP-mltl-aux.simps
    using list-concat-set-union
    by (metis append.assoc convert-α)
  {
    assume ∗: ψ ∈ ?front
    then have ψ-is: ψ = Global-mltl-ext a b L (And-mltl-ext (Not_c α) β)
      by simp
    have ?thesis unfolding Release-mltl-ext ψ-is to-mltl.simps wpd-mltl.simps
      by linarith
  } moreover {
    assume ∗: ψ ∈ ?middle
    then obtain x where ψ-is: ψ = Mighty-Release-mltl-ext x β (interval-times
a L ! 0)
        (interval-times a L ! 1 − 1)
        [interval-times a L ! 1 − interval-times a L ! 0]
          and x-in: x ∈ set ?D
      by auto
    have ub: interval-times a L ! 1 − 1 ≤ b
      using interval-times-diff-ge-general[OF a-leq-b composition-L, of length L 1
?s]
      using slast length-L
    by (metis diff-add-inverse2 diff-le-self dual-order.strict-iff-order dual-order.trans
less-eq-iff-succ-less zero-less-diff)
    have x-ih: wpd-mltl (to-mltl x) ≤ wpd-mltl (to-mltl α)
      using Suc(1)[OF α-nnf α-welldef x-in composition-α]
     by blast
  then have ?thesis unfolding ψ-is Release-mltl-ext to-mltl.simps wpd-mltl.simps
Mighty-Release-mltl-ext.simps
      using ub by auto
  } moreover {
    assume ∗: ψ ∈ ?back
    then obtain x i where ψ-is: ψ = And-mltl-ext
                  (Global-mltl-ext
                    (interval-times a L ! 0)
                   (interval-times a L ! i − 1) [?s!i − ?s!0] (And-mltl-ext (Not_c
α) β))
                  (Mighty-Release-mltl-ext x β
                    (interval-times a L ! i)
                    (interval-times a L ! (i + 1) − 1)
                    [interval-times a L ! (i + 1) −
```

```
                    interval-times a L ! i])
                and x-in: x ∈ set ?D
                and i-bound: 1 ≤ i ∧ i < length L
      by auto
    have x-ih: wpd-mltl (to-mltl x) ≤ wpd-mltl (to-mltl α)
      using Suc(1)[OF α-nnf α-welldef x-in composition-α] by blast
    have lb: a < ?s!i
      using interval-times-diff-ge-general sfirst
       by (smt (verit, ccfv-SIG) a-leq-b composition-L i-bound less-or-eq-imp-le
order-less-le-trans zero-less-one)
    have welldef: ?s!i < ?s!(i+1)
      using interval-times-diff-ge[OF a-leq-b composition-L]
      using i-bound length-L by blast
    have ub: ?s!(i+1) ≤ b+1
       using interval-times-diff-ge-general[OF a-leq-b composition-L, of length L
i+1 ?s]
      using i-bound slast
       by (metis less-iff-succ-less-eq order-le-imp-less-or-eq order-less-imp-le or-
der-refl)
     have ?thesis unfolding Release-mltl-ext ψ-is to-mltl.simps wpd-mltl.simps
Mighty-Release-mltl-ext.simps
      using lb welldef ub x-ih by auto
   }
   ultimately show ?thesis
     using split by blast
  qed
qed

lemma And-mltl-list-nonempty:
  assumes A ≠ [] and B ≠ []
  shows And-mltl-list A B ≠ []
proof−
  have length A > 0
    using assms by blast
  then obtain ha Ta where A: A = ha#Ta
    using list.exhaust by auto
  have length B > 0
    using assms by blast
  then obtain hb Tb where B: B = hb#Tb
    using list.exhaust by auto
  show ?thesis
    using assms unfolding And-mltl-list.simps A B pairs.simps
    by blast
qed

lemma Global-mltl-decomp-nonempty:
  assumes D ≠ []
  shows Global-mltl-decomp D a n L ≠ []
  using assms
```

proof(*induct n*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case* **unfolding** *Global-mltl-decomp.simps Global-mltl-list.simps*
using *And-mltl-list-nonempty* **by** *auto*
qed

lemma *LP-mltl-aux-nonempty*:
assumes $\exists \varphi\text{-init}.\ \varphi = \text{convert-nnf-ext}\ \varphi\text{-init}$
assumes *intervals-welldef* (*to-mltl* φ)
assumes *is-composition-MLTL* φ
shows *LP-mltl-aux* φ $k \neq []$
using *assms*
proof(*induct k arbitrary*: φ)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case*
proof(*cases* φ)
case *True-mltl-ext*
then show *?thesis* **by** *simp*
next
case *False-mltl-ext*
then show *?thesis* **by** *simp*
next
case (*Prop-mltl-ext p*)
then show *?thesis* **by** *simp*
next
case (*Not-mltl-ext q*)
then have $\exists p.\ q = \text{Prop-mltl-ext}\ p$
using *convert-nnf-form-Not-Implies-Prop Suc*
by (*metis convert-nnf-ext-to-mltl-commute to-mltl.simps*(*4*) *to-mltl-prop-bijective*)

then obtain *p* **where** $q = \text{Prop-mltl-ext}\ p$ **by** *blast*
then show *?thesis*
unfolding *Not-mltl-ext* **by** *simp*
next
case (*And-mltl-ext* α β)
have $\alpha\text{-nnf}$: $\exists \varphi\text{-init}.\ \alpha = \text{convert-nnf-ext}\ \varphi\text{-init}$
using *Suc*(*2*) **unfolding** *And-mltl-ext*
by (*metis convert-nnf-ext.simps*(*4*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*3*))

have $\beta\text{-nnf}$: $\exists \varphi\text{-init}.\ \beta = \text{convert-nnf-ext}\ \varphi\text{-init}$
using *Suc*(*2*) **unfolding** *And-mltl-ext*
by (*metis convert-nnf-ext.simps*(*4*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*3*))
have $\alpha\text{-welldef}$: *intervals-welldef* (*to-mltl* α) **and**

β-welldef: *intervals-welldef* (*to-mltl* *β*)
using *Suc*(*3*) **unfolding** *And-mltl-ext* **by** *simp-all*
have *α-composition*: *is-composition-MLTL* *α* **and**
β-composition: *is-composition-MLTL* *β*
using *Suc*(*4*) **unfolding** *And-mltl-ext is-composition-MLTL.simps*
by *simp-all*
have *α-ih*: *LP-mltl-aux* *α* *k* ≠ []
using *Suc*(*1*)[*OF* *α-nnf* *α-welldef* *α-composition*] **by** *simp*
have *β-ih*: *LP-mltl-aux* *β* *k* ≠ []
using *Suc*(*1*)[*OF* *β-nnf* *β-welldef* *β-composition*] **by** *simp*
show *?thesis*
unfolding *And-mltl-ext LP-mltl-aux.simps And-mltl-list.simps*
using *pairs.simps*(*2*) *α-ih* *β-ih*
by (*metis* (*no-types*, *lifting*) *α-nnf* *β-nnf append-is-Nil-conv convert-nnf-ext-convert-nnf-ext list.map-disc-iff pairs.elims*)
next
case (*Or-mltl-ext* *α* *β*)
have *α-nnf*: ∃*φ-init*. *α* = *convert-nnf-ext* *φ-init*
using *Suc*(*2*) **unfolding** *Or-mltl-ext*
by (*metis convert-nnf-ext.simps*(*5*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*4*))

have *β-nnf*: ∃*φ-init*. *β* = *convert-nnf-ext* *φ-init*
using *Suc*(*2*) **unfolding** *Or-mltl-ext*
by (*metis convert-nnf-ext.simps*(*5*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*4*))
have *α-welldef*: *intervals-welldef* (*to-mltl* *α*) **and**
β-welldef: *intervals-welldef* (*to-mltl* *β*)
using *Suc*(*3*) **unfolding** *Or-mltl-ext* **by** *simp-all*
have *α-composition*: *is-composition-MLTL* *α* **and**
β-composition: *is-composition-MLTL* *β*
using *Suc*(*4*) **unfolding** *Or-mltl-ext is-composition-MLTL.simps*
by *simp-all*
have *α-ih*: *LP-mltl-aux* *α* *k* ≠ []
using *Suc*(*1*)[*OF* *α-nnf* *α-welldef* *α-composition*] **by** *simp*
have *β-ih*: *LP-mltl-aux* *β* *k* ≠ []
using *Suc*(*1*)[*OF* *β-nnf* *β-welldef* *β-composition*] **by** *simp*
then show *?thesis*
unfolding *Or-mltl-ext LP-mltl-aux.simps And-mltl-list.simps*
by (*metis* (*no-types*, *lifting*) *α-ih* *α-nnf concat.simps*(*1*) *concat-eq-append-conv convert-nnf-ext-convert-nnf-ext list.map-disc-iff not-Cons-self2 pairs.elims*)
next
case (*Future-mltl-ext* *a* *b* *L* *α*)
have *α-nnf*: ∃*φ-init*. *α* = *convert-nnf-ext* *φ-init*
using *Suc*(*2*) **unfolding** *Future-mltl-ext*
by (*metis convert-nnf-ext.simps*(*6*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*5*))

have *α-welldef*: *intervals-welldef* (*to-mltl* *α*)
using *Suc*(*3*) **unfolding** *Future-mltl-ext* **by** *simp-all*
have *α-composition*: *is-composition-MLTL* *α*
using *Suc*(*4*) **unfolding** *Future-mltl-ext is-composition-MLTL.simps*

```
    by simp-all
  have α-ih: LP-mltl-aux α k ≠ []
    using Suc(1)[OF α-nnf α-welldef α-composition] by simp
  then show ?thesis
    unfolding Future-mltl-ext LP-mltl-aux.simps And-mltl-list.simps
     by (metis (no-types, lifting) Future-mltl-list.elims α-nnf append-is-Nil-conv
convert-nnf-ext-convert-nnf-ext map-is-Nil-conv)
 next
  case (Global-mltl-ext a b L α)
  have α-nnf: ∃φ-init. α = convert-nnf-ext φ-init
    using Suc(2) unfolding Global-mltl-ext
   by (metis convert-nnf-ext.simps(7) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(6))

  then have α-convert: convert-nnf-ext α = α
    using convert-nnf-ext-convert-nnf-ext by metis
  have α-welldef: intervals-welldef (to-mltl α)
    using Suc(3) unfolding Global-mltl-ext by simp-all
   have α-composition: is-composition-MLTL α
    using Suc(4) unfolding Global-mltl-ext is-composition-MLTL.simps
    by simp-all
  have α-ih: LP-mltl-aux α k ≠ []
    using Suc(1)[OF α-nnf α-welldef α-composition] by simp
  let ?D = LP-mltl-aux α k
  {
    assume ∗: length ?D ≤ 1
    then have ?thesis unfolding Global-mltl-ext LP-mltl-aux.simps
      using α-ih α-convert by simp
  } moreover {
    assume ∗: length ?D > 1
    have D-is: LP-mltl-aux φ (Suc k) = Global-mltl-decomp ?D a (b − a) L
      unfolding Global-mltl-ext LP-mltl-aux.simps
      using ∗ α-convert by auto
    have ?thesis unfolding D-is
      using Global-mltl-decomp-nonempty α-ih by blast
  }
  ultimately show ?thesis by linarith
 next
  case (Until-mltl-ext α a b L β)
  have α-nnf: ∃φ-init. α = convert-nnf-ext φ-init
    using Suc(2) unfolding Until-mltl-ext
   by (metis convert-nnf-ext.simps(8) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(7))

  have β-nnf: ∃φ-init. β = convert-nnf-ext φ-init
    using Suc(2) unfolding Until-mltl-ext
   by (metis convert-nnf-ext.simps(8) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(7))
  have α-welldef: intervals-welldef (to-mltl α) and
      β-welldef: intervals-welldef (to-mltl β) and
      a-leq-b: a ≤ b
    using Suc(3) unfolding Until-mltl-ext by simp-all
```

have α-*composition*: *is-composition-MLTL* α **and**
β-*composition*: *is-composition-MLTL* β **and**
L-composition: *is-composition* $(b-a+1)$ L
using *Suc*(*4*) **unfolding** *Until-mltl-ext is-composition-MLTL.simps*
by *simp-all*
have α-*ih*: *LP-mltl-aux* α $k \neq []$
using *Suc*(*1*)[*OF* α-*nnf* α-*welldef* α-*composition*] **by** *simp*
have β-*ih*: *LP-mltl-aux* β $k \neq []$
using *Suc*(*1*)[*OF* β-*nnf* β-*welldef* β-*composition*] **by** *simp*
show *?thesis* **unfolding** *Until-mltl-ext LP-mltl-aux.simps*
using α-*ih* β-*ih*
by (*metis* (*no-types*, *lifting*) *Until-mltl-list.elims* β-*nnf append-is-Nil-conv convert-nnf-ext-convert-nnf-ext map-is-Nil-conv*)
next
case (*Release-mltl-ext* α a b L β)
show *?thesis* **unfolding** *LP-mltl-aux.simps Release-mltl-ext*
by (*meson append-is-Nil-conv not-Cons-self2*)
qed
qed

8.2 Union Theorem

Forward Direction **lemma** *exist-first*:
fixes *lb* *i*::*nat*
assumes *lowerbound*: $lb \leq i$ **and** *iprop*: $(P\ i)$
shows $\exists j.\ (lb \leq j \wedge j \leq i \wedge (P\ j)$
$\wedge\ (\forall l.\ (lb \leq l \wedge l < j) \longrightarrow \neg(P\ l)))$
using *lowerbound iprop*
proof(*induct i*−*lb arbitrary*: *i rule*: *less-induct*)
case *less*
{
assume $*$: $\forall\, l{\geq}lb.\ l < i \longrightarrow \neg(P\ l)$
then have *?case*
using *less* **by** *blast*
} moreover {
assume $*$: $\exists\, i'{\geq}lb.\ i' < i \wedge (P\ i')$
then obtain i' **where** $lb \leq i' \wedge i' < i \wedge P\ i'$
by *blast*
then have *?case*
using *less.hyps*(*1*)[*of* i'] **by** *fastforce*
}
ultimately show *?case* **by** *blast*
qed

lemma *exist-bound-split*:
fixes a m b::*nat*
assumes $a \leq b$
assumes $\exists\, i.\ a \leq i \wedge i \leq b \wedge P\ i$

shows $(\exists\, i.\ a \leq i \wedge i \leq m-1 \wedge P\ i) \vee$
$(\exists\, i.\ m \leq i \wedge i \leq b \wedge P\ i \wedge \neg(\exists\, j.\ a \leq j \wedge j < m \wedge P\ j))$
using *assms* **by** *fastforce*

lemma *Global-mltl-ext-obtain*:
fixes D::$'a$ *mltl-ext list* **and** π::$'a$ *set list*
and α::$'a$ *mltl-ext* **and** $a\ b\ k$::*nat*
assumes *a-leq-b*: $a \leq b$
assumes *length-π*: *length* $\pi \geq b +$ *wpd-mltl* (*to-mltl* α)
assumes *semantics*: *semantics-mltl-ext* π (*Global-mltl-ext* $a\ b\ L\ \alpha$)
assumes *ih*: $\bigwedge$*trace*. *semantics-mltl-ext trace* $\alpha \Longrightarrow$
wpd-mltl (*to-mltl* α) $\leq$ *length trace* $\Longrightarrow$
$\exists\, x{\in}$*set D*. *semantics-mltl-ext trace x*
shows $\exists\, X.$ (*length* $X = b{-}a{+}1$) $\wedge$
$(\forall\, i{<}$*length* $X.$ $(X!i \in$ *set* $D) \wedge$ *semantics-mltl-ext* (*drop* $(a{+}i)\ \pi$) $(X!i))$
proof−
have *semantics*: $\bigwedge i.\ a \leq i \wedge i \leq b \Longrightarrow$ *semantics-mltl-ext* (*drop* $i\ \pi$) α
using *semantics length-π a-leq-b*
unfolding *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by (*metis add-diff-cancel-left' wpd-geq-one diff-add-zero le-less-Suc-eq le-trans less-add-Suc1 not-one-le-zero*)
have *ih*: $\exists\, x{\in}$*set D*. *semantics-mltl-ext* (*drop* $i\ \pi$) x
if *i-bound*: $a \leq i \wedge i \leq b$ **for** i
proof−
have *cond1*: *semantics-mltl-ext* (*drop* $i\ \pi$) α
using *semantics*[*of i*] *i-bound* **by** *blast*
have *cond2*: *wpd-mltl* (*to-mltl* α) $\leq$ *length* (*drop* $i\ \pi$)
using *length-π a-leq-b i-bound* **by** *auto*
show *?thesis*
using *ih*[*OF cond1 cond2*] **by** *blast*
qed
show *?thesis* **using** *ih a-leq-b*
proof(*induct* $b{-}a$ *arbitrary*: $a\ b$)
case *0*
then have *aeqb*: $a = b$ **by** *simp*
then obtain x **where** *semantics-x*: *semantics-mltl-ext* (*drop* $a\ \pi$) x
and *x-in*: $x \in$ *set D*
using *0*(*2*)[*of a*] **by** *blast*
let *?X* = $[x]$
have *length-X*: *length ?X* $= b - a + 1$ **using** *aeqb* **by** *simp*
have *?X* ! $i \in$ *set D* $\wedge$ *semantics-mltl-ext* (*drop* $(a{+}i)\ \pi$) (*?X* ! i)
if *i-bound*: $i{<}$*length ?X* **for** i
using *semantics-x that x-in* **by** *force*
then show *?case* **using** *length-X* **by** *blast*
next
case (*Suc n*)
then have *n-eq*: $n = b - 1 - a$ **by** *simp*
have $\exists\, X.$ *length* $X = b - 1 - a + 1 \wedge$
$(\forall\, i{<}$*length* $X.$

```
        X ! i ∈ set D ∧ semantics-mltl-ext (drop (a + i) π) (X ! i))
      using Suc(1)[OF n-eq] unfolding Bex-def
       using Suc.hyps(2) Suc.prems(1) diff-diff-left diff-le-self plus-1-eq-Suc by
fastforce
    then obtain X where length-X: length X = b−a and
     X-prop: ∀ i<length X. X ! i ∈ set D ∧ semantics-mltl-ext (drop (a + i) π) (X
! i)
      by (metis Suc.hyps(2) Suc-eq-plus1 n-eq)
    obtain x where x-in: x ∈ set D
    and semantics-x: semantics-mltl-ext (drop b π) x
      using Suc(3)[of b] unfolding Bex-def using Suc(4) by blast
    let ?L = X@[x]
    have length-L: length ?L = b − a + 1
      using length-X by simp
    have ?L ! i ∈ set D ∧ semantics-mltl-ext (drop (a + i) π) (?L ! i)
      if i-bound: i < length ?L for i
    proof−
      {
       assume ∗: i < b−a
       have ?thesis
         using X-prop length-X
         by (metis ∗ nth-append)
      } moreover {
       assume ∗: i = b−a
       then have x-is: (X @ [x]) ! i = x
         using length-L by (metis length-X nth-append-length)
       have ?thesis unfolding x-is
         using x-in Suc semantics-x unfolding ∗ by simp
      }
      ultimately show ?thesis using i-bound length-L by fastforce
    qed
    then show ?case using length-L by blast
  qed
qed


lemma Release-semantics-split:
  assumes (∀ i. a ≤ i ∧ i ≤ b ⟶ semantics-mltl (drop i π) (to-mltl β)) ∨
    (∃ j≥a. j ≤ b − 1 ∧ semantics-mltl (drop j π) (to-mltl α) ∧
          (∀ k. a ≤ k ∧ k ≤ j ⟶
               semantics-mltl (drop k π) (to-mltl β)))
  shows ((∀ i. a ≤ i ∧ i ≤ b ⟶ semantics-mltl (drop i π) (to-mltl β))
        ∧(∀ i. a ≤ i ∧ i ≤ b ⟶ semantics-mltl (drop i π) (Not_m (to-mltl α))))
       ∨ (∃ j≥a. j ≤ b ∧
           semantics-mltl (drop j π) (to-mltl α) ∧
           (∀ k. a ≤ k ∧ k ≤ j ⟶
                semantics-mltl (drop k π) (to-mltl β)))
proof−
  {assume ∗: (∀ i. a ≤ i ∧ i ≤ b ⟶ semantics-mltl (drop i π) (to-mltl β)) ∧
```

```
        ¬(∃j≥a. j ≤ b − 1 ∧ semantics-mltl (drop j π) (to-mltl α) ∧
          (∀k. a ≤ k ∧ k ≤ j ⟶
              semantics-mltl (drop k π) (to-mltl β)))
    then have semantics: ∀j. a ≤ j ∧ j ≤ b−1 ⟶ ¬semantics-mltl (drop j π)
(to-mltl α) ∨
         ¬(∀k. a ≤ k ∧ k ≤ j ⟶
              semantics-mltl (drop k π) (to-mltl β))
     by blast
   then have ¬semantics-mltl (drop j π) (to-mltl α)
     if j-bound: a ≤ j ∧ j ≤ b−1 for j
   proof−
     have semantics-mltl (drop k π) (to-mltl β)
       if k-bound:  a ≤ k ∧ k ≤ j for k
       using k-bound j-bound ∗ by auto
     then show ?thesis using semantics j-bound by blast
   qed
   then have ?thesis using ∗
     by (metis dual-order.trans semantics-mltl.simps(4))
  } moreover {
   assume (∀i. a ≤ i ∧ i ≤ b ⟶ semantics-mltl (drop i π) (to-mltl β)) ∧
        (∃j≥a. j ≤ b − 1 ∧ semantics-mltl (drop j π) (to-mltl α) ∧
          (∀k. a ≤ k ∧ k ≤ j ⟶
              semantics-mltl (drop k π) (to-mltl β)))
   then have ?thesis
     by (meson diff-le-self le-trans)
  } moreover {
   assume (∃j≥a. j ≤ b − 1 ∧ semantics-mltl (drop j π) (to-mltl α) ∧
          (∀k. a ≤ k ∧ k ≤ j ⟶
              semantics-mltl (drop k π) (to-mltl β)))
   then have ?thesis
     by (meson diff-le-self le-trans)
  }
  ultimately show ?thesis using assms
   by blast
qed


theorem LP-mltl-aux-language-union-forward:
   fixes φ::'a mltl-ext and k::nat and π::'a set list
  assumes intervals-welldef: intervals-welldef (to-mltl φ)
  assumes is-nnf: ∃φ-init. φ = convert-nnf-ext φ-init
  assumes composition: is-composition-MLTL φ
  assumes D-is: D = LP-mltl-aux φ k
  assumes semantics: semantics-mltl-ext π φ
  assumes trace-length: length π ≥ wpd-mltl (to-mltl φ)
  shows ∃ψ ∈ set D. semantics-mltl-ext π ψ
  using assms
proof(induct k arbitrary: φ D π)
  case 0
```

then show *?case* **by** *auto*
next
case (*Suc k*)
then show *?case*
proof(*cases* φ)
case *True-mltl-ext*
then show *?thesis* **using** *Suc* **by** *simp*
next
case *False-mltl-ext*
then show *?thesis* **using** *Suc* **by** *simp*
next
case (*Prop-mltl-ext x3*)
then show *?thesis* **using** *Suc* **by** *simp*
next
case (*Not-mltl-ext x4*)
then have $\exists$ *p*. *x4* = *Prop-mltl-ext p*
using *convert-nnf-form-Not-Implies-Prop Suc*(*3*)
by (*metis convert-nnf-ext-to-mltl-commute to-mltl.simps*(*4*) *to-mltl-prop-bijective*)

then show *?thesis* **using** *Suc*
by (*metis LP-mltl-aux.simps*(*5*) *ListMem-iff Not-mltl-ext elem*)
next
case (*And-mltl-ext* α β)
have α-*welldef*: *intervals-welldef* (*to-mltl* α) **and**
β-*welldef*: *intervals-welldef* (*to-mltl* β)
using *Suc*(*2*) **unfolding** *And-mltl-ext* **by** *simp-all*
have α-*nnf*: $\exists$ φ-*init*. α = *convert-nnf-ext* φ-*init*
using *Suc*(*3*) **unfolding** *And-mltl-ext*
by (*metis convert-nnf-ext.simps*(*4*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*3*))
have β-*nnf*: $\exists$ φ-*init*. β = *convert-nnf-ext* φ-*init*
by (*metis And-mltl-ext Suc.prems*(*2*) *convert-nnf-ext.simps*(*4*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*3*))
have α-*composition*: *is-composition-MLTL* α **and**
β-*composition*: *is-composition-MLTL* β
using *Suc*(*4*) **unfolding** *And-mltl-ext is-composition-MLTL.simps*
by *simp-all*
have α-*semantics*: *semantics-mltl-ext* π α **and**
β-*semantics*: *semantics-mltl-ext* π β
using *Suc*(*6*) **unfolding** *And-mltl-ext semantics-mltl-ext-def*
by *simp-all*
have α-*wpd*: *wpd-mltl* (*to-mltl* α) $\leq$ *length* π **and**
β-*wpd*: *wpd-mltl* (*to-mltl* β) $\leq$ *length* π
using *Suc*(*7*) **unfolding** *And-mltl-ext to-mltl.simps wpd-mltl.simps*
by *simp-all*
have α-*ih*: $\exists$ *xa*$\in$*set* (*LP-mltl-aux* α *k*). *semantics-mltl-ext* π *xa*
using *Suc*(*1*)[*OF* α-*welldef* α-*nnf* α-*composition* - α-*semantics* α-*wpd*] **by** *blast*
have β-*ih*: $\exists$ *xb*$\in$*set* (*LP-mltl-aux* β *k*). *semantics-mltl-ext* π *xb*
using *Suc*(*1*)[*OF* β-*welldef* β-*nnf* β-*composition* - β-*semantics* β-*wpd*] **by**

blast
then obtain *xa* **where** *xa-in*: *xa* ∈ *set* (*LP-mltl-aux* α *k*) **and** *xa-semantics*: *semantics-mltl-ext* π *xa*
using α*-ih* **by** *blast*
then obtain *xb* **where** *xb-in*: *xb* ∈ *set* (*LP-mltl-aux* β *k*) **and** *xb-semantics*: *semantics-mltl-ext* π *xb*
using β*-ih* **by** *blast*
have *xab-in*: *And-mltl-ext xa xb* ∈ *set D*
unfolding *Suc*(*5*) *And-mltl-ext LP-mltl-aux.simps*
using *xa-in xb-in And-mltl-list-member*
by (*metis* α*-nnf* β*-nnf convert-nnf-ext-convert-nnf-ext in-set-member*)
have *xab-semantics*: *semantics-mltl-ext* π (*And-mltl-ext xa xb*)
using *xa-semantics xb-semantics* **unfolding** *semantics-mltl-ext-def*
by *simp*
show *?thesis* **using** *xab-in xab-semantics* **by** *blast*
next
case (*Or-mltl-ext* α β)
have α*-welldef*: *intervals-welldef* (*to-mltl* α) **and**
β*-welldef*: *intervals-welldef* (*to-mltl* β)
using *Suc*(*2*) **unfolding** *Or-mltl-ext* **by** *simp-all*
have α*-nnf*: ∃ φ*-init*. α = *convert-nnf-ext* φ*-init*
using *Suc*(*3*) **unfolding** *Or-mltl-ext*
by (*metis convert-nnf-ext.simps*(*5*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*4*))

have β*-nnf*: ∃ φ*-init*. β = *convert-nnf-ext* φ*-init*
by (*metis Or-mltl-ext Suc.prems*(*2*) *convert-nnf-ext.simps*(*5*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*4*))
have α*-composition*: *is-composition-MLTL* α **and**
β*-composition*: *is-composition-MLTL* β
using *Suc*(*4*) **unfolding** *Or-mltl-ext is-composition-MLTL.simps*
by *simp-all*
have α*-wpd*: *wpd-mltl* (*to-mltl* α) ≤ *length* π **and**
β*-wpd*: *wpd-mltl* (*to-mltl* β) ≤ *length* π
using *Suc*(*7*) **unfolding** *Or-mltl-ext to-mltl.simps wpd-mltl.simps*
by *simp-all*
have αβ*-semantics*: *semantics-mltl-ext* π α ∨ *semantics-mltl-ext* π β
using *Suc*(*6*) **unfolding** *Or-mltl-ext semantics-mltl-ext-def*
by *simp*
let *?D-*α = *LP-mltl-aux* α *k* **and** *?D-*β = *LP-mltl-aux* β *k*
{
assume ∗: *semantics-mltl-ext* π α ∧ ¬*semantics-mltl-ext* π β
have α*-ih*: ∃ *xa*∈*set* (*LP-mltl-aux* α *k*). *semantics-mltl-ext* π *xa*
using ∗ *Suc*(*1*)[*OF* α*-welldef* α*-nnf* α*-composition - -* α*-wpd*] **by** *blast*
then obtain *xa* **where** *xa-in*: *xa* ∈ *set ?D-*α **and** *xa-semantics*: *semantics-mltl-ext* π *xa*
using α*-ih* **by** *blast*
let *?*ψ = *And-mltl-ext xa* (*Not*$_c$ β)
have *xa*β*-in*: *?*ψ ∈ *set* (*And-mltl-list ?D-*α [*Not*$_c$ β])
using *xa-in And-mltl-list-member* **unfolding** *List.member-def*

by (*metis list.set-intros(1)*)
then have *xaβ-in*: *?ψ* ∈ *set D*
unfolding *Suc(5) Or-mltl-ext LP-mltl-aux.simps*
using *list-concat-set-union*
[*of And-mltl-list ?D-α ?D-β* @ *And-mltl-list* [Not_c *α*] *?D-β*
And-mltl-list (*LP-mltl-aux α k*) [Not_c *β*]]
by (*metis UnCI α-nnf β-nnf append-assoc convert-nnf-ext-convert-nnf-ext*)
have *xaβ-semantics*: *semantics-mltl-ext π ?ψ* **using** ∗ *xa-semantics*
unfolding *semantics-mltl-ext-def semantics-mltl.simps to-mltl.simps*
by *simp*
have *?thesis* **using** *xaβ-in xaβ-semantics* **by** *blast*
} moreover {
assume ∗: ¬*semantics-mltl-ext π α* ∧ *semantics-mltl-ext π β*
have *β-ih*: ∃ *xb*∈*set* (*LP-mltl-aux β k*). *semantics-mltl-ext π xb*
using ∗ *Suc(1)*[*OF β-welldef β-nnf β-composition - - β-wpd*] **by** *blast*
then obtain *xb* **where** *xa-in*: *xb* ∈ *set ?D-β* **and** *xa-semantics*: *semantics-mltl-ext π xb*
using *β-ih* **by** *blast*
let *?ψ* = *And-mltl-ext* (Not_c *α*) *xb*
have *αxb-in*: *?ψ* ∈ *set* (*And-mltl-list* [Not_c *α*] *?D-β*)
using *xa-in And-mltl-list-member* **unfolding** *List.member-def*
by (*metis list.set-intros(1)*)
then have *αxb-in*: *?ψ* ∈ *set* (*And-mltl-list ?D-α ?D-β* @ *And-mltl-list* [Not_c *α*] *?D-β*)
using *list-concat-set-union*[*of And-mltl-list ?D-α ?D-β And-mltl-list* [Not_c *α*] *?D-β*]
by *blast*
then have *αxb-in*: *?ψ* ∈ *set D*
unfolding *Suc(5) Or-mltl-ext LP-mltl-aux.simps*
using *list-concat-set-union*
[*of And-mltl-list ?D-α ?D-β* @ *And-mltl-list* [Not_c *α*] *?D-β*
And-mltl-list (*LP-mltl-aux α k*) [Not_c *β*]]
by (*metis UnCI α-nnf β-nnf append-assoc convert-nnf-ext-convert-nnf-ext*)
have *αxb-semantics*: *semantics-mltl-ext π ?ψ* **using** ∗ *xa-semantics*
unfolding *semantics-mltl-ext-def semantics-mltl.simps to-mltl.simps*
by *simp*
have *?thesis* **using** *αxb-in αxb-semantics* **by** *blast*
} moreover {
assume ∗: *semantics-mltl-ext π α* ∧ *semantics-mltl-ext π β*
have *α-ih*: ∃ *xa*∈*set* (*LP-mltl-aux α k*). *semantics-mltl-ext π xa*
using ∗ *Suc(1)*[*OF α-welldef α-nnf α-composition - - α-wpd*] **by** *blast*
have *β-ih*: ∃ *xb*∈*set* (*LP-mltl-aux β k*). *semantics-mltl-ext π xb*
using ∗ *Suc(1)*[*OF β-welldef β-nnf β-composition - - β-wpd*] **by** *blast*
then obtain *xa* **where** *xa-in*: *xa* ∈ *set* (*LP-mltl-aux α k*) **and** *xa-semantics*: *semantics-mltl-ext π xa*
using *α-ih* **by** *blast*
then obtain *xb* **where** *xb-in*: *xb* ∈ *set* (*LP-mltl-aux β k*) **and** *xb-semantics*: *semantics-mltl-ext π xb*
using *β-ih* **by** *blast*

```
    have xab-in: And-mltl-ext xa xb ∈ set D
      unfolding Suc(5) Or-mltl-ext LP-mltl-aux.simps
      using xa-in xb-in And-mltl-list-member list-concat-set-union
      unfolding List.member-def
      by (metis UnCI α-nnf β-nnf convert-nnf-ext-convert-nnf-ext)
    have xab-semantics: semantics-mltl-ext π (And-mltl-ext xa xb)
      using xa-semantics xb-semantics unfolding semantics-mltl-ext-def
      by simp
    have ?thesis using xab-in xab-semantics by blast
   }
   ultimately show ?thesis using αβ-semantics by blast
  next
   case (Future-mltl-ext a b L α)
   have α-welldef: intervals-welldef (to-mltl α)
    using Suc(2) unfolding Future-mltl-ext by auto
   have α-nnf: ∃ φ-init. α = convert-nnf-ext φ-init
    using Suc(3) unfolding Future-mltl-ext
   by (metis convert-nnf-ext.simps(6) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(5))
   have α-composition: is-composition-MLTL α
    using Suc(4) unfolding Future-mltl-ext is-composition-MLTL.simps by blast
   have α-wpd: b + wpd-mltl (to-mltl α) ≤ length π
    using Suc(7) unfolding Future-mltl-ext to-mltl.simps wpd-mltl.simps
    by simp
   have a-leq-b: a ≤ b and length-π-geq-b: b < length π and length-π-ge-a: a <
length π
    and semantics: ∃ i. (a ≤ i ∧ i ≤ b) ∧ semantics-mltl (drop i π) (to-mltl α)
    using Suc(6) α-wpd
   unfolding Future-mltl-ext semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps
    using wpd-geq-one[of (to-mltl α)]
    by simp-all
   have composition-L: is-composition (b − a + 1) L
    using Suc(4) unfolding Future-mltl-ext is-composition-MLTL.simps by blast
   then have s0: (interval-times a L ! 0) = a
    using interval-times-first by auto
   have slast: interval-times a L ! (length L) = b+1
    using interval-times-last[OF a-leq-b composition-L] by blast
   have length-L: length L ≥ 0
    using composition-L composition-length-lb by blast
   let ?s = interval-times a L
   let ?D-α = LP-mltl-aux α k
   let ?decomp = (concat(map (λi. And-mltl-list
                        [Global-mltl-ext (?s ! 0)
                          (?s ! i − 1) [?s!i − ?s!0] (Not_c α)]
                        (Future-mltl-list ?D-α (?s ! i) (?s ! (i + 1) − 1)
                          [?s ! (i + 1) − ?s ! i]))
                [1..<length L]))
   {
    assume ∗: ∃ i. (a ≤ i ∧ i ≤ (?s!1−1)) ∧ semantics-mltl (drop i π) (to-mltl
α)
```

then obtain *i* **where** *i-bounds*: $a \leq i \wedge i \leq (?s!1-1)$ **and**
semantics: *semantics-mltl* (*drop i* π) (*to-mltl* α) **by** *blast*
have *length-s*: *length ?s* $\geq$ *2*
using *i-bounds*
by (*metis a-leq-b add-less-same-cancel2 antisym-conv3 interval-times-first interval-times-length less-eq-iff-succ-less less-iff-succ-less-eq less-nat-zero-code one-add-one slast verit-comp-simplify1*(*1*))
have *dropi-length*: *wpd-mltl* (*to-mltl* α) $\leq$ *length* (*drop i* π)
proof −
have *1* $\leq$ *length L*
using *length-s* **unfolding** *interval-times-def* **by** *simp*
then have *interval-times a L ! 1* $\leq$ *interval-times a L ! length L*
using *interval-times-diff-ge-general*[*OF a-leq-b composition-L, of length L 1 ?s*]
by *fastforce*
then have *interval-times a L ! 1 − 1* $\leq$ *b*
using *slast* **by** *auto*
then show *?thesis*
using α*-wpd i-bounds* **by** *force*
qed
have $\exists$ *x*$\in$*set* (*LP-mltl-aux* α *k*). *semantics-mltl-ext* (*drop i* π) *x*
using *Suc*(*1*)[*OF* α*-welldef* α*-nnf* α*-composition, of ?D-*α *drop i* π] *semantics*
using *semantics-mltl-ext-def* α*-wpd dropi-length* **by** *blast*
then obtain *x* **where** *x-in*: *x*$\in$*set* (*LP-mltl-aux* α *k*) **and**
x-semantics: *semantics-mltl-ext* (*drop i* π) *x*
by *blast*
let *?*ψ = *Future-mltl-ext* (*?s*!*0*) (*?s*!*1*−*1*) [*?s*!*1 − ?s*!*0*] *x*
have ψ*-in*: *?*ψ $\in$ *set* (*Future-mltl-list ?D-*α (*?s*!*0*) (*?s*!*1*−*1*) [*?s*!*1 − ?s*!*0*])
unfolding *Future-mltl-list.simps* **using** *x-in* **by** *simp*
then have ψ*-in*: *?*ψ $\in$ *set* ((*Future-mltl-list ?D-*α (*?s*!*0*) (*?s*!*1*−*1*) [*?s*!*1 − ?s*!*0*]) @
(*concat*
(*map* (λ*i. And-mltl-list*
[*Global-mltl-ext* (*?s ! 0*) (*?s ! i − 1*) [*?s*!*i − ?s*!*0*] (Not_c α)]
(*Future-mltl-list ?D-*α (*?s ! i*) (*?s ! (i + 1) − 1*)
[*?s ! (i + 1) − ?s ! i*]))
[*1..<length L*])))
by *force*
have ψ*-semantics*: *semantics-mltl-ext* π *?*ψ
using *x-semantics* **unfolding** *s0 semantics-mltl-ext-def*
unfolding *semantics-mltl.simps to-mltl.simps*
using *i-bounds length-*π*-geq-b length-*π*-ge-a* **by** *auto*
have *?thesis* **unfolding** *Suc*(*5*) *Future-mltl-ext LP-mltl-aux.simps*
using ψ*-in* ψ*-semantics*
proof −
have *convert-nnf-ext* α = α
by (*metis* (*full-types*) α*-nnf convert-nnf-ext-convert-nnf-ext*)
then have *Future-mltl-ext* (*interval-times a L ! 0*)
(*interval-times a L ! 1 − 1*) [*interval-times a L ! 1 − interval-times a L ! 0*] *x* $\in$

set (*Future-mltl-list* (*LP-mltl-aux* (*convert-nnf-ext* α) *k*)
(*interval-times a L* ! *0*) (*interval-times a L* ! *1* − *1*)
[*interval-times a L* ! *1* − *interval-times a L* ! *0*] @
concat (*map* (λ*n*. *And-mltl-list* [*Global-mltl-ext*
(*interval-times a L* ! *0*) (*interval-times a L* ! *n* − *1*) [*?s*!*n* − *?s*!*0*] (*Not*$_c$ α)]
(*Future-mltl-list* (*LP-mltl-aux* (*convert-nnf-ext* α) *k*)
(*interval-times a L* ! *n*) (*interval-times a L* ! (*n* + *1*) − *1*)
[*interval-times a L* ! (*n* + *1*) − *interval-times a L* ! *n*])) [*1*..<*length L*]))
using ψ-*in* **by** *presburger*
then show $\exists\, m{\in}$*set* (*let ms* = *LP-mltl-aux* (*convert-nnf-ext* α) *k*; *ns* = *interval-times a L in Future-mltl-list ms* (*ns* ! *0*) (*ns* ! *1* − *1*) [*ns* ! *1* − *ns* ! *0*] @ *concat* (*map* (λ*n*. *And-mltl-list* [*Global-mltl-ext* (*ns* ! *0*) (*ns* ! *n* − *1*) [*ns*!*n* − *ns*!*0*] (*Not*$_c$ α)] (*Future-mltl-list ms* (*ns* ! *n*) (*ns* ! (*n* + *1*) − *1*) [*ns* ! (*n* + *1*) − *ns* ! *n*])) [*1*..<*length L*])). *semantics-mltl-ext* π *m*
by (*meson* ψ-*semantics*)
qed
} moreover {
assume ∗: $\exists\, i.$ ((*?s*!*1*) ≤ *i* ∧ *i* ≤ *b*) ∧ *semantics-mltl* (*drop i* π) (*to-mltl* α) ∧
¬($\exists\, i.$ (*a* ≤ *i* ∧ *i* ≤ (*?s*!*1*−*1*)) ∧ *semantics-mltl* (*drop i* π) (*to-mltl* α))
obtain *t*′ **where** *t*′-*facts*: ((*?s*!*1*) ≤ *t*′ ∧ *t*′ ≤ *b*) ∧ *semantics-mltl* (*drop t*′ π) (*to-mltl* α)
using ∗ **by** *blast*
then have $\exists\, j.$ (*interval-times a L* ! *1* ≤ *j* ∧ *j* ≤ *t*′) ∧
semantics-mltl (*drop j* π) (*to-mltl* α) ∧
(∀ *l*. (*interval-times a L* ! *1* ≤ *l* ∧ *l* < *j*) ⟶
¬ *semantics-mltl* (*drop l* π) (*to-mltl* α))
using *exist-first*[*of* (*?s*!*1*) *t*′ $\lambda i.$ *semantics-mltl* (*drop i* π) (*to-mltl* α)]
by *simp*
then obtain *t* **where**
t-bounds: (*interval-times a L* ! *1* ≤ *t* ∧ *t* ≤ *t*′) **and**
t-semantics: *semantics-mltl* (*drop t* π) (*to-mltl* α) **and**
t-minimal: (∀ *l*. (*interval-times a L* ! *1* ≤ *l* ∧ *l* < *t*) ⟶
¬ *semantics-mltl* (*drop l* π) (*to-mltl* α)) **by** *auto*
have *dropt-length*: *wpd-mltl* (*to-mltl* α) ≤ *length* (*drop t* π)
proof−
have *t*′ ≤ *b*
using *t*′-*facts* **by** *blast*
then show *?thesis*
using α-*wpd t-bounds* **by** *auto*
qed
have $\exists\, i.$ *interval-times a L* ! *i* ≤ *t* ∧
t ≤ *interval-times a L* ! (*i* + *1*) − *1* ∧ *1* ≤ *i* ∧ *i* < *length L*
using *interval-times-obtain-aux*[*of a b L ?s t*]
using *a-leq-b composition-L t-bounds t-semantics*
using *le-trans t*′-*facts* **by** *blast*
then obtain *i* **where** *t-bound*: *interval-times a L* ! *i* ≤ *t* ∧ *t* ≤ *interval-times a L* ! (*i* + *1*) − *1*
and *i-bound*: *1* ≤ *i* ∧ *i* < *length L*

by *blast*
have $\exists x{\in}set\ (LP\text{-}mltl\text{-}aux\ \alpha\ k).\ semantics\text{-}mltl\text{-}ext\ (drop\ t\ \pi)\ x$
using $Suc(1)[OF\ \alpha\text{-}welldef\ \alpha\text{-}nnf\ \alpha\text{-}composition,\ of\ ?D\text{-}\alpha\ drop\ t\ \pi]$
using *semantics-mltl-ext-def t-semantics dropt-length* **by** *blast*
then obtain *x* **where** *x-in*: $x{\in}set\ (LP\text{-}mltl\text{-}aux\ \alpha\ k)$ **and**
x-semantics: $semantics\text{-}mltl\text{-}ext\ (drop\ t\ \pi)\ x$
by *blast*
let $?\psi = And\text{-}mltl\text{-}ext$
$(Global\text{-}mltl\text{-}ext\ (?s\ !\ 0)\ (?s\ !\ i - 1)\ [?s!i - ?s!0]\ (Not_c\ \alpha))$
$(Future\text{-}mltl\text{-}ext\ (?s\ !\ i)\ (?s\ !\ (i + 1) - 1)\ [?s\ !\ (i + 1) - ?s\ !\ i]\ x)$
have $?\psi \in set\ ?decomp$
proof−
have $?\psi \in set\ (And\text{-}mltl\text{-}list$
$[Global\text{-}mltl\text{-}ext\ (?s\ !\ 0)$
$(?s\ !\ i - 1)\ [?s!i - ?s!0]\ (Not_c\ \alpha)]$
$(Future\text{-}mltl\text{-}list\ ?D\text{-}\alpha\ (?s\ !\ i)\ (?s\ !\ (i + 1) - 1)$
$[?s\ !\ (i + 1) - ?s\ !\ i]))$
using *x-in* **unfolding** *Future-mltl-list.simps* **by** *auto*
then have $?\psi \in set\ ((map\ (\lambda i.\ And\text{-}mltl\text{-}list$
$[Global\text{-}mltl\text{-}ext$
$(interval\text{-}times\ a\ L\ !\ 0)$
$(interval\text{-}times\ a\ L\ !\ i - 1)\ [?s!i - ?s!0]\ (Not_c\ \alpha)]$
$(Future\text{-}mltl\text{-}list\ (LP\text{-}mltl\text{-}aux\ \alpha\ k)$
$(interval\text{-}times\ a\ L\ !\ i)$
$(interval\text{-}times\ a\ L\ !\ (i + 1) - 1)$
$[interval\text{-}times\ a\ L\ !\ (i + 1) -$
$interval\text{-}times\ a\ L\ !\ i]))$
$[1..<length\ L])!(i-1))$ **using** *i-bound* **by** *auto*
then show *?thesis*
using *set-concat i-bound* **by** *fastforce*
qed
then have $\psi\text{-}in$: $?\psi \in set\ (Future\text{-}mltl\text{-}list\ ?D\text{-}\alpha\ (?s\ !\ 0)\ (?s\ !\ 1 - 1)\ [?s\ !\ 1 - ?s\ !\ 0]$ @
$concat(map\ (\lambda i.\ And\text{-}mltl\text{-}list$
$[Global\text{-}mltl\text{-}ext\ (?s\ !\ 0)$
$(?s\ !\ i - 1)\ [?s!i - ?s!0]\ (Not_c\ \alpha)]$
$(Future\text{-}mltl\text{-}list\ ?D\text{-}\alpha\ (?s\ !\ i)\ (?s\ !\ (i + 1) - 1)$
$[?s\ !\ (i + 1) - ?s\ !\ i]))$
$[1..<length\ L]))$
by *simp*
have $\psi\text{-}semantics$: $semantics\text{-}mltl\text{-}ext\ \pi\ ?\psi$
proof−
have *bound*: $interval\text{-}times\ a\ L\ !\ 0 \leq interval\text{-}times\ a\ L\ !\ i - 1$
using $interval\text{-}times\text{-}diff\text{-}ge\text{-}general[OF\ a\text{-}leq\text{-}b\ composition\text{-}L,\ of\ -\ 0]$ *length-L i-bound*
by (*simp add*: *add-le-imp-le-diff less-iff-succ-less-eq*)
have *not-semantics*: $\neg\ semantics\text{-}mltl\ (drop\ ia\ \pi)\ (to\text{-}mltl\ \alpha)$
if *ia-bound*: $(interval\text{-}times\ a\ L\ !\ 0 \leq ia \wedge ia \leq interval\text{-}times\ a\ L\ !\ i - 1)$ **for** *ia*

proof−
{
assume *ia-location*: $ia \leq$ *interval-times a L* ! *1* − *1*
have *?thesis* **using** ∗ *ia-bound*
using *ia-location s0* **by** *auto*
} moreover {
assume *ia-location*: $ia >$ *interval-times a L* ! *1* − *1*
have *interval-times a L* ! $i - 1 <$ *interval-times a L* ! i
using *interval-times-diff-ge*[*OF a-leq-b composition-L*, *of i*−*1 ?s*]
using *i-bound* **by** *fastforce*
then have $ia < t$
using *t-bound ia-bound* **by** *auto*
then have *ia-cond*: *interval-times a L* ! $1 \leq ia \wedge ia < t$
using *ia-location* **by** *simp*
then have *?thesis* **using** *t-minimal* **by** *blast*
}
ultimately show *?thesis* **by** *linarith*
qed
then have *global-not*: *semantics-mltl-ext* π
(*Global-mltl-ext* (*interval-times a L* ! *0*) (*interval-times a L* ! $i - 1$) [*?s*!i − *?s*!*0*] ($Not_c\ \alpha$))
unfolding *semantics-mltl-ext-def semantics-mltl.simps to-mltl.simps*
using *bound not-semantics* **by** *blast*
have *future*: *semantics-mltl-ext* π (*Future-mltl-ext* (*interval-times a L* ! i)
(*interval-times a L* ! $(i + 1) - 1$) [*interval-times a L* ! $(i + 1)$ − *interval-times a L* ! i] x)
proof−
have *interval-times a L* ! $i \leq b$
using *interval-times-diff-ge-general*[*OF a-leq-b composition-L*, *of length L i ?s*]
unfolding *slast* **using** *i-bound* **by** *auto*
then have *trace-length*: *interval-times a L* ! $i <$ *length* π
using *length-π-geq-b* **by** *auto*
have *semantics*: ($\exists$ *ia*. (*interval-times a L* ! $i \leq ia \wedge$
$ia \leq$ *interval-times a L* ! $(i + 1) - 1) \wedge$
semantics-mltl (*drop ia* π) (*to-mltl* x))
using *x-semantics t-bound semantics-mltl-ext-def*
by *auto*
have *interval-times a L* ! $i \leq$ *interval-times a L* ! $(i + 1) - 1$
using *interval-times-diff-ge*[*OF a-leq-b composition-L*, *of i ?s*]
using *i-bound* **by** *simp*
then show *?thesis* **unfolding** *semantics-mltl-ext-def semantics-mltl.simps to-mltl.simps*
using *trace-length semantics* **by** *blast*
qed
show *?thesis* **using** *global-not future*
unfolding *semantics-mltl-ext-def semantics-mltl.simps* **by** *simp*
qed
have *?thesis*

unfolding *Suc*(*5*) *Future-mltl-ext LP-mltl-aux.simps*
using ψ*-in* ψ*-semantics*
proof −
have *convert-nnf-ext* α = α
by (*metis* α*-nnf convert-nnf-ext-convert-nnf-ext*)
then have *And-mltl-ext* (*Global-mltl-ext* (*interval-times a L* ! *0*) (*interval-times a L* ! *i* − *1*) [*?s*!*i* − *?s*!*0*] (*Not*$_c$ α))
(*Future-mltl-ext* (*interval-times a L* ! *i*) (*interval-times a L* ! (*i* + *1*) − *1*)
[*interval-times a L* ! (*i* + *1*) − *interval-times a L* ! *i*] *x*) ∈
set (*Future-mltl-list* (*LP-mltl-aux* (*convert-nnf-ext* α) *k*) (*interval-times a L* ! *0*)
(*interval-times a L* ! *1* − *1*)
[*interval-times a L* ! *1* − *interval-times a L* ! *0*]
@ *concat* (*map* (λ*n*. *And-mltl-list* [*Global-mltl-ext* (*interval-times a L* ! *0*) (*interval-times a L* ! *n* − *1*) [*?s*!*n* − *?s*!*0*] (*Not*$_c$ α)]
(*Future-mltl-list* (*LP-mltl-aux* (*convert-nnf-ext* α) *k*) (*interval-times a L* ! *n*) (*interval-times a L* ! (*n* + *1*) − *1*) [*interval-times a L* ! (*n* + *1*) − *interval-times a L* ! *n*]))
[*1*..<*length L*]))
using ψ*-in* **by** *presburger*
then show ∃ *m*∈*set* (*let ms* = *LP-mltl-aux* (*convert-nnf-ext* α) *k*;
ns = *interval-times a L in Future-mltl-list ms* (*ns* ! *0*) (*ns* ! *1* − *1*)
[*ns* ! *1* − *ns* ! *0*] @ *concat* (*map* (λ*n*. *And-mltl-list*
[*Global-mltl-ext* (*ns* ! *0*) (*ns* ! *n* − *1*) [*ns*!*n* − *ns*!*0*] (*Not*$_c$ α)] (*Future-mltl-list ms*
(*ns* ! *n*) (*ns* ! (*n* + *1*) − *1*) [*ns* ! (*n* + *1*) − *ns* ! *n*])) [*1*..<*length L*])). *semantics-mltl-ext* π *m*
by (*meson* ψ*-semantics*)
qed
}
ultimately show *?thesis* **using** *semantics* **by** *force*
next
case (*Global-mltl-ext a b L* α)
have α*-welldef*: *intervals-welldef* (*to-mltl* α)
using *Suc*(*2*) **unfolding** *Global-mltl-ext* **by** *auto*
have α*-nnf*: ∃ φ*-init*. α = *convert-nnf-ext* φ*-init*
using *Suc*(*3*) **unfolding** *Global-mltl-ext*
by (*metis convert-nnf-ext.simps*(*7*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*6*))
have α*-composition*: *is-composition-MLTL* α
using *Suc*(*4*) **unfolding** *Global-mltl-ext is-composition-MLTL.simps* **by** *blast*
have α*-wpd*: *b* + *wpd-mltl* (*to-mltl* α) ≤ *length* π
using *Suc*(*7*) **unfolding** *Global-mltl-ext to-mltl.simps wpd-mltl.simps*
by *simp*
have *a-leq-b*: *a* ≤ *b*
using *Suc*(*6*) α*-wpd* **unfolding** *Global-mltl-ext semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *blast*
have *length-*π*-geq-b*: *b* < *length* π
and *semantics*: ∀ *i*. *a* ≤ *i* ∧ *i* ≤ *b* ⟶ *semantics-mltl* (*drop i* π) (*to-mltl* α)
using *Suc*(*6*) α*-wpd* **unfolding** *Global-mltl-ext semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *wpd-geq-one*[*of* (*to-mltl* α)] **by** *auto*

let *?D-α = LP-mltl-aux α k*
{
assume ∗: *length ?D-α ≤ 1*
let *?ψ = Global-mltl-ext a b L α*
have *semantics*: *semantics-mltl π (to-mltl ?ψ)*
using *Suc(6)* **unfolding** *Global-mltl-ext semantics-mltl-ext-def*
by *blast*
have *ψ-in*: *?ψ ∈ set D* **using** *Suc(5)* ∗
unfolding *Global-mltl-ext LP-mltl-aux.simps*
by (*metis* (*full-types*) *α-nnf convert-nnf-ext-convert-nnf-ext list.set-intros(1)*)

have *?thesis*
using *semantics ψ-in Global-mltl-ext Suc.prems(5)* **by** *auto*
} moreover {
assume ∗: *length ?D-α > 1*
then have *D-is*: *D = Global-mltl-decomp ?D-α a (b − a) L*
using *Suc(5)* ∗ **unfolding** *Global-mltl-ext LP-mltl-aux.simps*
by (*metis* (*full-types*) *α-nnf convert-nnf-ext-convert-nnf-ext leD*)
have *semantics-global*: *semantics-mltl-ext π (Global-mltl-ext a b L α)*
using *Suc(6)* **unfolding** *Global-mltl-ext* **by** *blast*
have *length-π*: *length π ≥ b + wpd-mltl (to-mltl α)*
using *Suc(6) α-wpd* **unfolding** *Global-mltl-ext semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *wpd-geq-one*[*of (to-mltl α)*] **by** *blast*
have *ih*: ⋀*trace. semantics-mltl-ext trace α ⟹*
wpd-mltl (to-mltl α) ≤ length trace ⟹
∃ a∈set (LP-mltl-aux α k). semantics-mltl-ext trace a
using *Suc(1)*[*OF α-welldef α-nnf α-composition, of ?D-α*] **by** *blast*
have *∃ X. length X = b − a + 1 ∧*
(∀ i<length X. X ! i ∈ set (LP-mltl-aux α k) ∧
semantics-mltl-ext (drop (a+i) π) (X ! i))
using *Global-mltl-ext-obtain*[*OF a-leq-b length-π semantics-global ih*]
by *blast*
then obtain *Y* **where** *length-Y*: *length Y = b − a + 1*
and *Y-prop*: *∀ i<length Y. Y!i ∈ set ?D-α ∧*
semantics-mltl-ext (drop (a+i) π) (Y ! i)
by *blast*
let *?X = map (λi. Global-mltl-ext (a+i) (a+i) [1] (Y!i)) [0..<length Y]*
let *?ψ = Ands-mltl-ext ?X*
have *cond1*: *?ψ = ?ψ* **by** *auto*
have *length-X*: *length ?X = b−a+1*
using *length-Y* **by** *simp*
have *cond2*: *∀ i<length ?X.*
∃ y∈set ?D-α. ?X ! i = Global-mltl-ext (a + i) (a + i) [1] y
using *Y-prop* **by** *simp*
have *ψ-in*: *?ψ ∈ set D*
using *in-Global-mltl-decomp-exact-converse*[*OF ∗ cond1 cond2 length-X*]
unfolding *D-is* **by** *blast*
have *ψ-semantics*: *semantics-mltl-ext π ?ψ*

```
    proof−
      have cond1: length ?X ≥ 1 using length-X by simp
      have semantics-mltl-ext π (?X!i)
        if i-bound: i < length ?X for i
      proof−
        have Xi-is: ?X!i = Global-mltl-ext (a + i) (a + i) [1] (Y ! i)
          using i-bound by auto
        show ?thesis unfolding Xi-is
          using Y-prop i-bound unfolding semantics-mltl-ext-def
          unfolding semantics-mltl.simps by auto
      qed
      then have (∀ x∈set ?X. semantics-mltl-ext π x)
        by auto
      then show ?thesis
        using Ands-mltl-semantics[of ?X π, OF cond1] by blast
    qed
    have ?thesis using D-is ψ-in ψ-semantics by blast
  }
  ultimately show ?thesis by linarith
next
  case (Until-mltl-ext α a b L β)
  have α-welldef: intervals-welldef (to-mltl α)
   and β-welldef: intervals-welldef (to-mltl β)
    using Suc(2) unfolding Until-mltl-ext by simp-all
  have α-nnf: ∃ φ-init. α = convert-nnf-ext φ-init
    using Suc(3) unfolding Until-mltl-ext
  by (metis convert-nnf-ext.simps(8) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(7))
  have β-nnf: ∃ φ-init. β = convert-nnf-ext φ-init
    using Suc(3) unfolding Until-mltl-ext
  by (metis convert-nnf-ext.simps(8) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(7))
  have α-convert: convert-nnf-ext α = α
    using α-nnf convert-nnf-ext-convert-nnf-ext by metis
  have β-convert: convert-nnf-ext β = β
    using β-nnf convert-nnf-ext-convert-nnf-ext by metis
  have α-composition: is-composition-MLTL α
   and β-composition: is-composition-MLTL β
   and L-composition: is-composition (b−a+1) L
    using Suc(4) unfolding Until-mltl-ext is-composition-MLTL.simps
    by simp-all
  have α-wpd: b + wpd-mltl (to-mltl α)−1 ≤ length π
   and β-wpd: b + wpd-mltl (to-mltl β) ≤ length π
    using Suc(7) unfolding Until-mltl-ext to-mltl.simps wpd-mltl.simps
    by simp-all
  have a-leq-b: a ≤ b and length-π-ge-b: b < length π
  and semantics: (∃ i. (a ≤ i ∧ i ≤ b) ∧
      semantics-mltl (drop i π) (to-mltl β) ∧
      (∀ j. a ≤ j ∧ j < i ⟶
          semantics-mltl (drop j π) (to-mltl α)))
   using Suc(6) α-wpd unfolding Until-mltl-ext semantics-mltl-ext-def to-mltl.simps
```

semantics-mltl.simps
using *wpd-geq-one*[*of to-mltl* β] β*-wpd*
by *simp-all*
let *?D-*β = *LP-mltl-aux* β *k*
let *?s* = *interval-times a L*
have *sfirst*: *?s*!*0* = *a*
using *interval-times-first* **by** *auto*
have *slast*: *?s*!(*length L*) = *b*+*1*
using *interval-times-last*[*OF a-leq-b L-composition*] **by** *auto*
have *length-L*: *length L* ≥ *1*
using *composition-length-lb*[*OF L-composition*] **by** *linarith*
have *s-second-lb*: *a* ≤ *interval-times a L* ! *1* − *1*
using *sfirst interval-times-diff-ge*[*OF a-leq-b L-composition*, *of 0 ?s*]
using *length-L* **by** *force*
have *s-second-ub*: *interval-times a L* ! *1* − *1* ≤ *b*
using *slast length-L*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L 1 ?s*]
by *force*
let *?front* = (*Until-mltl-list* α *?D-*β (*?s* ! *0*) (*?s* ! *1* − *1*) [*?s* ! *1* − *?s* ! *0*])
let *?back* = (*concat* (*map* (λ*i*. *And-mltl-list*
[*Global-mltl-ext*
(*?s* ! *0*) (*?s* ! *i* − *1*) [*?s*!*i* − *?s*!*0*] (*And-mltl-ext* α (Not_c β))]
(*Until-mltl-list* α *?D-*β (*?s* ! *i*) (*?s* ! (*i* + *1*) − *1*)
[*?s* ! (*i* + *1*) − *?s* ! *i*])) [*1*..<*length L*]))
have *D-union*: *set D* = (*set ?front*) ∪ (*set ?back*)
unfolding *Suc*(*5*) *Until-mltl-ext LP-mltl-aux.simps*
using α*-convert* β*-convert list-concat-set-union* **by** *metis*
let *?P* = λ*i*. *semantics-mltl* (*drop i* π) (*to-mltl* β) ∧
(∀*j*. *a* ≤ *j* ∧ *j* < *i* ⟶ *semantics-mltl* (*drop j* π) (*to-mltl* α))
{
assume ∗: ∃ *i*. (*a* ≤ *i*) ∧ (*i* ≤ (*?s*!*1*)−*1*) ∧ *?P i*
then obtain *i* **where** *i-bound*: (*a* ≤ *i* ∧ *i* ≤ (*?s*!*1*)−*1*) **and**
semantics: *semantics-mltl* (*drop i* π) (*to-mltl* β) ∧
(∀*j*. *a* ≤ *j* ∧ *j* < *i* ⟶ *semantics-mltl* (*drop j* π) (*to-mltl* α))
by *blast*
have *semantics-dropi*: *semantics-mltl-ext* (*drop i* π) β
using *semantics* **unfolding** *semantics-mltl-ext-def* **by** *blast*
have *length-dropi*: *wpd-mltl* (*to-mltl* β) ≤ *length* (*drop i* π)
using β*-wpd length-*π*-ge-b i-bound a-leq-b s-second-ub* **by** *auto*
obtain *x* **where** *x-semantics*: *semantics-mltl-ext* (*drop i* π) *x*
and *x-in*: *x* ∈ *set ?D-*β
using *Suc*(*1*)[*OF* β*-welldef* β*-nnf* β*-composition - semantics-dropi length-dropi*, *of ?D-*β]
by *blast*
let *?*ψ = (*Until-mltl-ext* α *a* ((*?s*!*1*)−*1*) [(*?s*!*1*) − *a*] *x*)
have ψ*-semantics*: *semantics-mltl-ext* π *?*ψ
using *semantics length-*π*-ge-b a-leq-b i-bound x-semantics*

unfolding *semantics-mltl-ext-def semantics-mltl.simps to-mltl.simps*
by *auto*
have $?\psi \in$ *set ?front*
using *x-in* **unfolding** *Until-mltl-list.simps sfirst* **by** *auto*
then have *ψ-in*: $?\psi \in$ *set D*
unfolding *D-union* **by** *blast*
have *?thesis*
using *ψ-semantics ψ-in* **by** *blast*
} moreover {
assume $*$: $\exists\, i.\ ((?s!1) \leq i) \wedge (i \leq b) \wedge\ ?P\ i \wedge$
$\neg(\exists\, j.\ a \leq j \wedge j < (?s!1) \wedge\ ?P\ j)$
then obtain t' **where** *t′-bound*: $((?s!1) \leq t') \wedge (t' \leq b)$ **and**
semantics: *?P t′* **and** *not-semantics*: $\neg(\exists\, j.\ a \leq j \wedge j < (?s!1) \wedge\ ?P\ j)$
by *blast*
have $\exists\, j{\geq}$*interval-times a L ! 1*. $j \leq t' \wedge$
?P j $\wedge$ $(\forall\, l.$ *interval-times a L ! 1* $\leq l \wedge l < j \longrightarrow \neg$ *?P l*)
proof−
have *cond1*: *interval-times a L ! 1* $\leq t'$
using *t′-bound* **by** *auto*
show *?thesis*
using *exist-first*[*of ?s!1 t′ ?P*, *OF cond1 semantics*] **by** *blast*
qed
then obtain t **where**
t-bound: *interval-times a L ! 1* $\leq t \wedge t \leq t'$ **and**
t-semantics: *?P t* **and**
t-minimal: $\forall\, l.$ *interval-times a L ! 1* $\leq l \wedge l < t \longrightarrow \neg$ *?P l*
by *blast*
have $\exists\, i.$ *interval-times a L ! i* $\leq t \wedge$
$t \leq$ *interval-times a L ! (i + 1) − 1* $\wedge$ *1* $\leq i \wedge i <$ *length L*
using *interval-times-obtain-aux*[*OF a-leq-b L-composition*, *of ?s t*]
using *t-bound t′-bound* **by** *simp*
then obtain i **where** *t-bound*: *interval-times a L ! i* $\leq t$
$\wedge\ t \leq$ *interval-times a L ! (i + 1) − 1*
and *i-bound*: *1* $\leq i \wedge i <$ *length L*
by *blast*
have *bound1*: *interval-times a L ! i < interval-times a L ! (i+1)*
using *interval-times-diff-ge*[*OF a-leq-b L-composition*, *of i ?s*]
using *i-bound* **by** *blast*
have *bound2*: $a \leq$ *interval-times a L ! i − 1*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i 0 ?s*]
using *i-bound sfirst* **by** *simp*
have *positive-i*: *interval-times a L ! i > 0*
using *i-bound sfirst*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i 0 ?s*]
by *auto*
have *global-α*: *semantics-mltl-ext π (Global-mltl-ext a (?s ! i − 1) [?s!i − ?s!0] α)*
proof−
have *semantics-mltl (drop ia π) (to-mltl α)*

if *ia-bound*: $a \leq ia \wedge ia \leq$ *interval-times a L* ! $i - 1$ **for** *ia*
proof−
have $a \leq ia \wedge ia < t$
using *ia-bound t-bound positive-i* **by** *auto*
then show *?thesis*
using *t-semantics* **by** *blast*
qed
then show *?thesis*
using *bound2*
unfolding *semantics-mltl-ext-def semantics-mltl.simps to-mltl.simps*
by *blast*
qed
have *global-not-β*: *semantics-mltl-ext π* (*Global-mltl-ext a* (*?s* ! $i - 1$) [*?s*!i − *?s*!*0*] (Not_c β))
proof−
have ¬ *semantics-mltl* (*drop ia π*) (*to-mltl β*)
if *ia-bound*: $a \leq ia \wedge ia \leq$ *interval-times a L* ! $i - 1$ **for** *ia*
proof−
have *globally*: $(\forall j.\ a \leq j \wedge j < ia \longrightarrow$
semantics-mltl (*drop j π*) (*to-mltl α*))
using *global-α* **unfolding** *semantics-mltl-ext-def semantics-mltl.simps to-mltl.simps*
using *length-π-ge-b a-leq-b*
using *antisym dual-order.trans that* **by** *auto*
have $a \leq ia \wedge ia < t$
using *ia-bound t-bound positive-i* **by** *auto*
then show *?thesis*
using *t-minimal globally*
by (*meson linorder-le-less-linear not-semantics*)
qed
then show *?thesis*
unfolding *semantics-mltl-ext-def semantics-mltl.simps to-mltl.simps*
using *bound2* **by** *blast*
qed
let *?ψ1* = *Global-mltl-ext* (*?s* ! *0*) (*?s* ! $i - 1$) [*?s*!i − *?s*!*0*] (*And-mltl-ext α* (Not_c β))
have *ψ1-semantics*: *semantics-mltl-ext π ?ψ1*
proof−
have *p1*: *semantics-mltl π* (*Global-mltl* (*?s* ! *0*) (*?s* ! $i - 1$) (*to-mltl α*))
using *global-α* **unfolding** *semantics-mltl-ext-def to-mltl.simps sfirst* **by** *blast*
have *p2*: *semantics-mltl π* (*Global-mltl* (*?s* ! *0*) (*?s* ! $i - 1$) (Not_m (*to-mltl β*)))
using *global-not-β* **unfolding** *semantics-mltl-ext-def to-mltl.simps sfirst* **by** *blast*
show *?thesis* **unfolding** *semantics-mltl-ext-def to-mltl.simps*
using *p1 p2 global-and-distribute* **by** *auto*
qed
have *interval-times a L* ! $(i + 1) \leq$ *?s*!(*length L*)

using *interval-times-diff-ge-general*[*OF a-leq-b L-composition, of length L i+1 ?s*]
using *i-bound*
by (*metis le-eq-less-or-eq less-iff-succ-less-eq*)
then have *interval-times a L ! (i + 1)−1 ≤ b*
using *slast* **by** *auto*
then have $t \leq b$
using *t-bound* **by** *simp*
then have *wpd-mltl* (*to-mltl* β) ≤ *length* (*drop t* π)
using β*-wpd* **by** *simp*
then obtain *x* **where** *x-semantics*: *semantics-mltl-ext* (*drop t* π) *x*
and *x-in*: $x \in$ *set ?D-*β
using *t-semantics*
using *Suc*(*1*)[*OF* β*-welldef* β*-nnf* β*-composition, of ?D-*β (*drop t* π)]
unfolding *semantics-mltl-ext-def* **by** *blast*
let *?ψ2 = Until-mltl-ext* α (*?s ! i*) (*?s ! (i + 1) − 1*) [*?s ! (i + 1) − ?s ! i*] *x*
have ψ*2-semantics*: *semantics-mltl-ext* π *?ψ2*
proof−
have (∀ *j. interval-times a L ! i* ≤ *j* ∧ *j < t* ⟶
semantics-mltl (*drop j* π) (*to-mltl* α))
using *t-minimal not-semantics*
by (*metis bound2 diff-less dual-order.strict-trans1 dual-order.strict-trans2 less-numeral-extra*(*1*) *nless-le positive-i t-semantics*)
then have *semantics-mltl* (*drop t* π) (*to-mltl x*) ∧
(∀ *j. interval-times a L ! i* ≤ *j* ∧ *j < t* ⟶
semantics-mltl (*drop j* π) (*to-mltl* α))
using *x-semantics* **unfolding** *semantics-mltl-ext-def* **by** *blast*
then have (∃ *ia.* (*interval-times a L ! i* ≤ *ia* ∧
ia ≤ *interval-times a L ! (i + 1) − 1*) ∧
semantics-mltl (*drop ia* π) (*to-mltl x*) ∧
(∀ *j. interval-times a L ! i* ≤ *j* ∧ *j < ia* ⟶
semantics-mltl (*drop j* π) (*to-mltl* α)))
using *t-bound* **by** *blast*
then show *?thesis*
unfolding *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *bound1*
by (*smt* (*verit*) ‹*interval-times a L ! (i + 1) − 1* ≤ *b*› *le-antisym le-neq-implies-less le-trans length-*π*-ge-b less-or-eq-imp-le*)
qed
let *?ψ = And-mltl-ext ?ψ1 ?ψ2*
have ψ*-semantics*: *semantics-mltl-ext* π *?ψ*
using ψ*1-semantics* ψ*2-semantics* **unfolding** *semantics-mltl-ext-def* **by** *simp*
have *?ψ* ∈ *set ?back*
using *x-in i-bound*
unfolding *Until-mltl-list.simps* **by** *auto*
then have ψ*-in*: *?ψ* ∈ *set D*
using *D-union* **by** *blast*

have *?thesis* **using** ψ*-semantics* ψ*-in* **by** *auto*
}
ultimately show *?thesis*
using *exist-bound-split*[*OF a-leq-b, of ?P ?s*!*1*] *semantics* **by** *blast*
next
case (*Release-mltl-ext* α *a b L* β)
have α*-welldef*: *intervals-welldef* (*to-mltl* α)
and β*-welldef*: *intervals-welldef* (*to-mltl* β)
using *Suc*(*2*) **unfolding** *Release-mltl-ext* **by** *simp-all*
have α*-nnf*: $\exists\,\varphi$*-init*. α = *convert-nnf-ext* φ*-init*
using *Suc*(*3*) **unfolding** *Release-mltl-ext*
by (*metis convert-nnf-ext.simps*(*9*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*8*))
have β*-nnf*: $\exists\,\varphi$*-init*. β = *convert-nnf-ext* φ*-init*
using *Suc*(*3*) **unfolding** *Release-mltl-ext*
by (*metis convert-nnf-ext.simps*(*9*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*8*))
have α*-convert*: *convert-nnf-ext* α = α
using α*-nnf convert-nnf-ext-convert-nnf-ext* **by** *metis*
have β*-convert*: *convert-nnf-ext* β = β
using β*-nnf convert-nnf-ext-convert-nnf-ext* **by** *metis*
have α*-composition*: *is-composition-MLTL* α
and β*-composition*: *is-composition-MLTL* β
and *L-composition*: *is-composition* (*b*−*a*+*1*) *L*
using *Suc*(*4*) **unfolding** *Release-mltl-ext is-composition-MLTL.simps*
by *simp-all*
have α*-wpd*: *b* + *wpd-mltl* (*to-mltl* α) ≤ *length* π
and β*-wpd*: *b* + *wpd-mltl* (*to-mltl* β) ≤ *length* π
using *Suc*(*7*) **unfolding** *Release-mltl-ext to-mltl.simps wpd-mltl.simps*
by *simp-all*
have *length-*π*-ge-b*: *b* < *length* π
using *wpd-geq-one*[*of to-mltl* β] β*-wpd*
by *auto*
have *a-leq-b*: *a* ≤ *b*
using *Suc*(*6*) α*-wpd* **unfolding** *Release-mltl-ext semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *blast*
have *semantics*: (∀ *i*. *a* ≤ *i* ∧ *i* ≤ *b* ⟶
semantics-mltl (*drop i* π) (*to-mltl* β)) ∨
(∃ *j*≥*a*. *j* ≤ *b* − *1* ∧
semantics-mltl (*drop j* π) (*to-mltl* α) ∧
(∀ *k*. *a* ≤ *k* ∧ *k* ≤ *j* ⟶
semantics-mltl (*drop k* π) (*to-mltl* β)))
using *Suc*(*6*) **unfolding** *Release-mltl-ext semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *length-*π*-ge-b* **by** *auto*
let *?D* = *LP-mltl-aux* α *k*
let *?s* = *interval-times a L*
have *sfirst*: *?s*!*0* = *a*
using *interval-times-first* **by** *auto*
have *slast*: *?s*!(*length L*) = *b*+*1*

```
    using interval-times-last[OF a-leq-b L-composition] by auto
  let ?front = set [Global-mltl-ext a b L (And-mltl-ext (Not_c α) β)]
  let ?middle = set (Mighty-Release-mltl-list ?D β (?s ! 0) (?s ! 1 − 1)
        [?s ! 1 − ?s ! 0])
  let ?back = set (concat (map (λi. And-mltl-list
                  [Global-mltl-ext
                   (?s ! 0) (?s ! i − 1) [?s!i − ?s!0] (And-mltl-ext (Not_c α)
β)]
                  (Mighty-Release-mltl-list ?D β (?s ! i)
                   (?s ! (i + 1) − 1) [?s ! (i + 1) − ?s ! i]))
            [1..<length L]))
  let ?P = λj. (semantics-mltl (drop j π) (to-mltl α) ∧
      (∀ k. a ≤ k ∧ k ≤ j ⟶
          semantics-mltl (drop k π) (to-mltl β)))
  have D-is: set D = ?front ∪ ?middle ∪ ?back
    unfolding Suc(5) Release-mltl-ext LP-mltl-aux.simps
    using α-convert list-concat-set-union
    by (metis append-assoc)
  {
    assume ∗: (∀ i. a ≤ i ∧ i ≤ b ⟶ semantics-mltl (drop i π) (to-mltl β))
        ∧(∀ i. a ≤ i ∧ i ≤ b ⟶ semantics-mltl (drop i π) (Not_m (to-mltl
α)))
    let ?ψ = Global-mltl-ext a b L (And-mltl-ext (Not_c α) β)
    have ψ-in: ?ψ ∈ set D
      using D-is by auto
    have semantics-mltl-ext π ?ψ
      unfolding semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps
      using a-leq-b ∗ by auto
    then have ?thesis using ψ-in by blast
  } moreover {
    assume ∗: ∃ i. a ≤ i ∧ i ≤ b ∧ ?P i
    then obtain t' where t'-semantics: ?P t'
            and t'-bound: a ≤ t' ∧ t' ≤ b
      by blast
    then obtain t where t-semantics: ?P t
            and t-bound: a ≤ t ∧ t ≤ t'
            and t-minimal: ∀ j. (a ≤ j ∧ j < t) ⟶ ¬ ?P j
      using exist-first[of a t' ?P] by blast
    have globally-notα: ∀ i. (a ≤ i ∧ i < t) ⟶
        ¬ (semantics-mltl-ext (drop i π) α)
      using t-minimal t-semantics unfolding semantics-mltl-ext-def by auto
    have α-semantics: semantics-mltl-ext (drop t π) α
      using t-semantics unfolding semantics-mltl-ext-def by blast
    have globally-β: ∀ i. (a ≤ i ∧ i ≤ t) ⟶ (semantics-mltl-ext (drop i π) β)
      using t-semantics unfolding semantics-mltl-ext-def by blast
    obtain i where t-bound: ?s!i ≤ t ∧ t ≤ ?s!(i+1)−1
          and i-bound: 0 ≤ i ∧ i < length L
      using interval-times-obtain[OF a-leq-b L-composition, of ?s t]
      using t-bound t'-bound by auto
```

```
    have lb: a ≤ ?s!i
      using i-bound sfirst interval-times-diff-ge-general[OF a-leq-b L-composition,
of i 0 ?s]
      by force
    have welldef: ?s!i < ?s!(i+1)
      using i-bound
      using interval-times-diff-ge[OF a-leq-b L-composition, of i ?s]
      by blast
    have ub: ?s!(i+1) ≤ b+1
      using i-bound slast interval-times-diff-ge-general[OF a-leq-b L-composition,
of length L i+1 ?s]
       by (metis Orderings.order-eq-iff less-iff-succ-less-eq order-le-imp-less-or-eq
order-less-imp-le)
    have wpd-mltl (to-mltl α) ≤ length (drop t π)
       using α-wpd t-bound i-bound sfirst welldef ub by auto
    then obtain x where x-semantics: semantics-mltl-ext (drop t π) x
                    and x-in: x ∈ set (LP-mltl-aux α k)
      using Suc(1)[OF α-welldef α-nnf α-composition - α-semantics, of ?D]
      by blast
    {
      assume i-bound: i = 0
        let ?ψ = Mighty-Release-mltl-ext x β a (interval-times a L ! 1 − 1)
[interval-times a L ! 1 − a]
      have ψ-in: ?ψ ∈ ?middle using x-in unfolding sfirst by auto
      then have ψ-in: ?ψ ∈ set D using D-is by blast
      have semantics-mltl-ext π ?ψ
      proof−
        have sem1: (∀ i. a ≤ i ∧ i ≤ interval-times a L ! 1 − 1 ⟶
         semantics-mltl (drop i π) (to-mltl β)) ∨
    (∃ j≥a. j ≤ interval-times a L ! 1 − 1 − 1 ∧
          semantics-mltl (drop j π) (to-mltl x) ∧
          (∀ k. a ≤ k ∧ k ≤ j ⟶
               semantics-mltl (drop k π) (to-mltl β)))
        proof−
          {
            assume t-loc: t = ?s ! (i + 1) − 1
            then have ?thesis
              using globally-β
              by (simp add: i-bound t-semantics)
          } moreover {
            assume t-loc: ?s ! i ≤ t ∧ t ≤ ?s ! (i + 1) − 1 −1
            then have ?thesis
              using t-semantics i-bound globally-β
           by (metis add-cancel-right-left semantics-mltl-ext-def sfirst x-semantics)

          }
          ultimately show ?thesis using t-bound by fastforce
        qed
        have sem2: (∃ i. (a ≤ i ∧ i ≤ interval-times a L ! 1 − 1) ∧
```

```
        semantics-mltl (drop i π) (to-mltl x))
         using x-semantics t-bound ub lb welldef unfolding semantics-mltl-ext-def
          using i-bound sfirst by auto
       show ?thesis unfolding Mighty-Release-mltl-ext.simps semantics-mltl-ext-def
to-mltl.simps semantics-mltl.simps
          using welldef i-bound sem1 sem2 length-π-ge-b a-leq-b by auto
       qed
       then have ?thesis
        using ψ-in by auto
      } moreover {
       assume i-bound: 0 < i ∧ i < length L
       have lb: a < ?s!i
        using i-bound sfirst interval-times-diff-ge-general[OF a-leq-b L-composition,
of i 0 ?s]
         by force
       let ?ψ = And-mltl-ext
                (Global-mltl-ext
                  a (interval-times a L ! i − 1) [?s!i − ?s!0] (And-mltl-ext (Not_c
α) β))
                (Mighty-Release-mltl-ext x β
                  (interval-times a L ! i) (interval-times a L ! (i + 1) − 1)
                  [interval-times a L ! (i + 1) − interval-times a L ! i])
       have ?ψ ∈ ?back
        using x-in i-bound sfirst by auto
       then have ψ-in: ?ψ ∈ set D using D-is by blast
       have semantics-mltl-ext π ?ψ
       proof−
        have p1: (∀ ia. a ≤ ia ∧ ia ≤ interval-times a L ! i − 1 ⟶
          ¬ semantics-mltl (drop ia π) (to-mltl α) ∧
          semantics-mltl (drop ia π) (to-mltl β))
          using globally-notα globally-β t-bound lb ub welldef
          unfolding semantics-mltl-ext-def by auto
        have p2: (∀ ia. interval-times a L ! i ≤ ia ∧
          ia ≤ interval-times a L ! (i + 1) − 1 ⟶
          semantics-mltl (drop ia π) (to-mltl β)) ∨
     (∃ j≥interval-times a L ! i.
          j ≤ interval-times a L ! (i + 1) − 1 − 1 ∧
          semantics-mltl (drop j π) (to-mltl x) ∧
          (∀ k. interval-times a L ! i ≤ k ∧ k ≤ j ⟶
              semantics-mltl (drop k π) (to-mltl β)))
          proof−
           {
            assume t-loc: t = interval-times a L ! (i + 1) − 1
            then have ?thesis
             using globally-β t-bound ub lb welldef
             by (metis le-trans less-or-eq-imp-le t-semantics)
           } moreover {
            assume t-loc: t ≤ interval-times a L ! (i + 1) − 1−1
            then have ?thesis
```

using *x-semantics globally-β t-bound ub lb welldef*
by (*meson le-trans less-imp-le-nat semantics-mltl-ext-def*)
}
ultimately show *?thesis* **using** *t-bound* **by** *fastforce*
qed
have *p3*: (∃ *ia*. (*interval-times a L* ! *i* ≤ *ia* ∧
ia ≤ *interval-times a L* ! (*i* + *1*) − *1*) ∧
semantics-mltl (*drop ia π*) (*to-mltl x*))
using *x-semantics i-bound lb ub welldef*
unfolding *semantics-mltl-ext-def*
using *t-bound* **by** *auto*
have *tracelen*: *interval-times a L* ! *i* < *length π*
using *length-π-ge-b ub welldef* **by** *simp*
then show *?thesis* **unfolding** *semantics-mltl-ext-def to-mltl.simps Mighty-Release-mltl-ext.simps semantics-mltl.simps*
using *lb ub welldef p1 p2 p3* **by** *auto*
qed
then have *?thesis*
using *ψ-in* **by** *auto*
}
ultimately have *?thesis* **using** *i-bound* **by** *blast*
}
ultimately show *?thesis* **using** *semantics Release-semantics-split*
by *blast*
qed
qed

Converse Direction **lemma** *LP-mltl-aux-language-union-converse*:
fixes *φ*::*'a mltl-ext* **and** *k*::*nat* **and** *π*::*'a set list*
assumes *intervals-welldef*: *intervals-welldef* (*to-mltl φ*)
assumes *is-nnf*: ∃ *φ-init*. *φ* = *convert-nnf-ext φ-init*
assumes *composition*: *is-composition-MLTL φ*
assumes *trace-length*: *length π* ≥ *wpd-mltl* (*to-mltl φ*)
assumes *D-is*: *D* = *LP-mltl-aux φ k*
assumes ∃ *ψ* ∈ *set D*. *semantics-mltl-ext π ψ*
shows *semantics-mltl-ext π φ*
using *assms*
proof(*induct k arbitrary*: *D φ π*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case*
proof(*cases φ*)
case *True-mltl-ext*
then show *?thesis* **unfolding** *semantics-mltl-ext-def* **by** *simp*
next
case *False-mltl-ext*
then show *?thesis* **using** *assms* **unfolding** *semantics-mltl-ext-def*

by (*metis LP-mltl-aux.simps*(3) *Suc.prems*(5) *Suc.prems*(6) *empty-iff empty-set semantics-mltl-ext-def set-ConsD*)
next
case (*Prop-mltl-ext p*)
then show *?thesis* **using** *Suc*
unfolding *semantics-mltl-ext-def* **by** *simp*
next
case (*Not-mltl-ext q*)
then have ∃ *p*. *q* = *Prop-mltl-ext p*
using *convert-nnf-form-Not-Implies-Prop Suc*
by (*metis convert-nnf-ext-to-mltl-commute to-mltl.simps*(4) *to-mltl-prop-bijective*)

then obtain *p* **where** *q* = *Prop-mltl-ext p* **by** *blast*
then show *?thesis*
using *Not-mltl-ext Suc* **by** *simp*
next
case (*And-mltl-ext* α β)
have α*-welldef*: *intervals-welldef* (*to-mltl* α) **and**
β*-welldef*: *intervals-welldef* (*to-mltl* β)
using *Suc*(2) **unfolding** *And-mltl-ext* **by** *simp-all*
have α*-nnf*: ∃ φ*-init*. α = *convert-nnf-ext* φ*-init*
using *Suc*(3) **unfolding** *And-mltl-ext*
by (*metis convert-nnf-ext.simps*(4) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(3))

have β*-nnf*: ∃ φ*-init*. β = *convert-nnf-ext* φ*-init*
using *Suc*(3) **unfolding** *And-mltl-ext*
by (*metis convert-nnf-ext.simps*(4) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(3))

have α*-composition*: *is-composition-MLTL* α **and**
β*-composition*: *is-composition-MLTL* β
using *Suc*(4) **unfolding** *And-mltl-ext is-composition-MLTL.simps* **by** *simp-all*
have α*-convert*: *convert-nnf-ext* α = α
using α*-nnf* β*-nnf convert-nnf-ext-convert-nnf-ext* **by** *metis*
have β*-convert*: *convert-nnf-ext* β = β
using α*-nnf* β*-nnf convert-nnf-ext-convert-nnf-ext* **by** *metis*
have α*-wpd*: *length* π ≥ *wpd-mltl* (*to-mltl* α) **and**
β*-wpd*: *length* π ≥ *wpd-mltl* (*to-mltl* β)
using *Suc*(5) **unfolding** *And-mltl-ext to-mltl.simps wpd-mltl.simps*
by *simp-all*
obtain ψ **where** ψ*-in*: ψ ∈ *set D*
and ψ*-semantics*: *semantics-mltl-ext* π ψ
using *Suc*(7) **by** *blast*
let *?Da* = *LP-mltl-aux* α *k*
let *?Db* = *LP-mltl-aux* β *k*
obtain *x y* **where** ψ*-is*: ψ = *And-mltl-ext x y*
and *x-in*: *x* ∈ *set ?Da*
and *y-in*: *y* ∈ *set ?Db*
using ψ*-in* **unfolding** *Suc*(6) *And-mltl-ext LP-mltl-aux.simps*
using *And-mltl-list-member* **unfolding** *List.member-def*

```
    using α-convert β-convert by metis
  have x-semantics: semantics-mltl-ext π x and
       y-semantics: semantics-mltl-ext π y
    using ψ-semantics unfolding semantics-mltl-ext-def ψ-is to-mltl.simps
    by simp-all
  have α-ih: semantics-mltl-ext π α
    using Suc(1)[OF α-welldef α-nnf α-composition α-wpd, of ?Da]
    using x-in x-semantics by blast
  have β-ih: semantics-mltl-ext π β
    using Suc(1)[OF β-welldef β-nnf β-composition β-wpd, of ?Db]
    using y-in y-semantics by blast
  show ?thesis
    using α-ih β-ih unfolding And-mltl-ext semantics-mltl-ext-def by auto
next
  case (Or-mltl-ext α β)
  have α-welldef: intervals-welldef (to-mltl α) and
       β-welldef: intervals-welldef (to-mltl β)
    using Suc(2) unfolding Or-mltl-ext by simp-all
  have α-nnf: ∃ φ-init. α = convert-nnf-ext φ-init
    using Suc(3) unfolding Or-mltl-ext
  by (metis convert-nnf-ext.simps(5) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(4))

  have β-nnf: ∃ φ-init. β = convert-nnf-ext φ-init
    using Suc(3) unfolding Or-mltl-ext
  by (metis convert-nnf-ext.simps(5) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(4))

  have α-composition: is-composition-MLTL α and
       β-composition: is-composition-MLTL β
   using Suc(4) unfolding Or-mltl-ext is-composition-MLTL.simps by simp-all
  have α-convert: convert-nnf-ext α = α
    using α-nnf β-nnf convert-nnf-ext-convert-nnf-ext by metis
  have β-convert: convert-nnf-ext β = β
    using α-nnf β-nnf convert-nnf-ext-convert-nnf-ext by metis
  have α-wpd: length π ≥ wpd-mltl (to-mltl α) and
       β-wpd: length π ≥ wpd-mltl (to-mltl β)
    using Suc(5) unfolding Or-mltl-ext to-mltl.simps wpd-mltl.simps
    by simp-all
  obtain ψ where ψ-in: ψ ∈ set D
           and ψ-semantics: semantics-mltl-ext π ψ
    using Suc(7) by blast
  let ?Da = LP-mltl-aux α k
  let ?Db = LP-mltl-aux β k
  let ?front = And-mltl-list ?Da ?Db
  let ?middle = And-mltl-list [Not_c α] ?Db
  let ?back = And-mltl-list ?Da [Not_c β]
  have cases: ψ ∈ (set ?front) ∪ (set ?middle) ∪ (set ?back)
    using Suc(6) unfolding Or-mltl-ext LP-mltl-aux.simps using ψ-in
    by (metis α-convert β-convert boolean-algebra-cancel.sup1 set-append)
  {
```

assume $*$: $\psi \in$ *set ?front*
obtain *x y* **where** ψ*-is*: $\psi =$ *And-mltl-ext x y*
and *x-in*: $x \in$ *set ?Da*
and *y-in*: $y \in$ *set ?Db*
using ψ*-in* $*$ **unfolding** *Or-mltl-ext LP-mltl-aux.simps*
using *And-mltl-list-member* **unfolding** *List.member-def*
using α*-convert* β*-convert* **by** *metis*
have *x-semantics*: *semantics-mltl-ext* π *x* **and**
y-semantics: *semantics-mltl-ext* π *y*
using ψ*-semantics* **unfolding** *semantics-mltl-ext-def* ψ*-is to-mltl.simps*
by *simp-all*
have α*-ih*: *semantics-mltl-ext* π α
using *Suc(1)*[*OF* α*-welldef* α*-nnf* α*-composition* α*-wpd*, *of ?Da*]
using *x-in x-semantics* **by** *blast*
have β*-ih*: *semantics-mltl-ext* π β
using *Suc(1)*[*OF* β*-welldef* β*-nnf* β*-composition* β*-wpd*, *of ?Db*]
using *y-in y-semantics* **by** *blast*
have *?thesis*
using α*-ih* β*-ih* **unfolding** *Or-mltl-ext semantics-mltl-ext-def* **by** *auto*
} moreover {
assume $*$: $\psi \in$ *set ?middle*
obtain *y* **where** ψ*-is*: $\psi =$ *And-mltl-ext* (Not_c α) *y*
and *y-in*: $y \in$ *set ?Db*
using ψ*-in* $*$ **unfolding** *Or-mltl-ext LP-mltl-aux.simps*
using *And-mltl-list-member* **unfolding** *List.member-def*
using α*-convert* β*-convert* **by** *auto*
have *x-semantics*: *semantics-mltl-ext* π (Not_c α) **and**
y-semantics: *semantics-mltl-ext* π *y*
using ψ*-semantics* **unfolding** *semantics-mltl-ext-def* ψ*-is to-mltl.simps*
by *simp-all*
have β*-ih*: *semantics-mltl-ext* π β
using *Suc(1)*[*OF* β*-welldef* β*-nnf* β*-composition* β*-wpd*, *of ?Db*]
using *y-in y-semantics* **by** *blast*
have *?thesis*
using β*-ih* **unfolding** *Or-mltl-ext semantics-mltl-ext-def* **by** *auto*
} moreover {
assume $*$: $\psi \in$ *set ?back*
obtain *x* **where** ψ*-is*: $\psi =$ *And-mltl-ext x* (Not_c β)
and *x-in*: $x \in$ *set ?Da*
using ψ*-in* $*$ **unfolding** *Or-mltl-ext LP-mltl-aux.simps*
using *And-mltl-list-member* **unfolding** *List.member-def*
using α*-convert* β*-convert*
by (*metis empty-iff empty-set set-ConsD*)
have *x-semantics*: *semantics-mltl-ext* π *x* **and**
y-semantics: *semantics-mltl-ext* π (Not_c β)
using ψ*-semantics* **unfolding** *semantics-mltl-ext-def* ψ*-is to-mltl.simps*
by *simp-all*
have α*-ih*: *semantics-mltl-ext* π α
using *Suc(1)*[*OF* α*-welldef* α*-nnf* α*-composition* α*-wpd*, *of ?Da*]

```
      using x-in x-semantics by blast
    have ?thesis
      using α-ih unfolding Or-mltl-ext semantics-mltl-ext-def by auto
  }
  ultimately show ?thesis using cases by blast
next
  case (Future-mltl-ext a b L α)
  have α-welldef: intervals-welldef (to-mltl α) and
       a-leq-b: a ≤ b
    using Suc(2) unfolding Future-mltl-ext by simp-all
  have α-nnf: ∃φ-init. α = convert-nnf-ext φ-init
    using Suc(3) unfolding Future-mltl-ext
  by (metis convert-nnf-ext.simps(6) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(5))

  have α-composition: is-composition-MLTL α and
       L-composition: is-composition (b−a+1) L
       using Suc(4) unfolding Future-mltl-ext is-composition-MLTL.simps by
simp-all
  have α-convert: convert-nnf-ext α = α
    using α-nnf convert-nnf-ext-convert-nnf-ext by metis
  have α-wpd: length π ≥ b + wpd-mltl (to-mltl α)
    using Suc(5) unfolding Future-mltl-ext to-mltl.simps wpd-mltl.simps
    by simp-all
  then have length-π-ge-b: length π > b
    using wpd-geq-one[of to-mltl α] by auto
  obtain ψ where ψ-in: ψ ∈ set D
            and ψ-semantics: semantics-mltl-ext π ψ
    using Suc(7) by blast
  let ?D = LP-mltl-aux α k
  let ?s = interval-times a L
  have length-L: 1 ≤ length L
    using composition-length-lb[OF L-composition] a-leq-b by linarith
  have sfirst: ?s!0 = a
    using interval-times-first by simp
  have slast: ?s!(length L) = b+1
    using interval-times-last[OF a-leq-b L-composition] by blast
  let ?front = (Future-mltl-list ?D (?s ! 0) (?s ! 1 − 1) [?s ! 1 − ?s ! 0])
  let ?back = (concat (map (λi. And-mltl-list
                        [Global-mltl-ext (?s ! 0) (?s ! i − 1) [?s!i − ?s!0] (Not_c α)]
                        (Future-mltl-list ?D (?s ! i) (?s ! (i + 1) − 1) [?s ! (i + 1)
− ?s ! i]))
              [1..<length L]))
  have cases: ψ ∈ (set ?front) ∪ (set ?back)
    using ψ-in using Suc(6) unfolding Future-mltl-ext LP-mltl-aux.simps
    using list-concat-set-union[of ?front ?back] α-convert by metis
  {
    assume ∗: ψ ∈ set ?front
    then obtain x where ψ-is: ψ = Future-mltl-ext (?s ! 0) (?s ! 1 − 1) [?s ! 1
− ?s ! 0] x
```

and *x-in*: $x \in$ *set ?D*
unfolding *Future-mltl-list.simps* **by** *fastforce*
obtain l **where** *x-semantics*: *semantics-mltl* (*drop* l π) (*to-mltl* x) **and**
l-bound: $a \leq l \wedge l \leq$ *interval-times* a L ! $1 - 1$
using ψ*-semantics*
unfolding ψ*-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps sfirst*
by *blast*
have *bound*: *interval-times* a L ! $1 - 1 \leq b$
using *slast length-L l-bound*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L 1 ?s*]
by *force*
then have *wpd-mltl* (*to-mltl* α) $\leq$ *length* (*drop* l π)
using α*-wpd l-bound* **by** *auto*
then have α*-ih*: *semantics-mltl-ext* (*drop* l π) α
using *Suc*(*1*)[*OF* α*-welldef* α*-nnf* α*-composition*, *of drop* l π *?D*]
using *x-in x-semantics semantics-mltl-ext-def* **by** *auto*
then have *?thesis* **unfolding** *Future-mltl-ext semantics-mltl-ext-def*
unfolding *to-mltl.simps semantics-mltl.simps*
using *length-*π*-ge-b a-leq-b l-bound bound* **by** *auto*
} moreover {
assume $*$: $\psi \in$ *set ?back*
then obtain i **where** ψ*-is*: $\psi \in$ *set* (*And-mltl-list*
[*Global-mltl-ext* (*?s* ! *0*) (*?s* ! $i - 1$) [*?s*!i − *?s*!*0*] (*Not*$_c$ α)]
(*Future-mltl-list ?D* (*?s* ! i) (*?s* ! $(i + 1) - 1$) [*?s* ! $(i + 1)$ − *?s* ! i]))
and *i-bound*: $1 \leq i \wedge i <$ *length* L
by *force*
obtain x **where** ψ*-is*: ψ = *And-mltl-ext*
(*Global-mltl-ext* (*?s* ! *0*) (*?s* ! $i - 1$) [*?s*!i − *?s*!*0*] (*Not*$_c$ α))
(*Future-mltl-ext* (*?s* ! i) (*?s* ! $(i + 1) - 1$) [*?s* ! $(i + 1)$ − *?s* ! i] x)
and *x-in*: $x \in$ *set ?D*
using ψ*-is* **unfolding** *Future-mltl-list.simps* **by** *auto*
obtain l **where** *x-semantics*: *semantics-mltl* (*drop* l π) (*to-mltl* x) **and**
l-bound: *?s* ! $i \leq l \wedge l \leq$ *?s* ! $(i + 1) - 1$
using ψ*-semantics* **unfolding** ψ*-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *auto*
have *interval-times* a L ! $(i + 1) \leq$ *interval-times* a L ! *length* L
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L i+1 ?s*]
using *i-bound*
by (*metis less-iff-succ-less-eq order-le-less*)
then have *bound*: *interval-times* a L ! $(i + 1) \leq$ *b+1*
unfolding *slast* **by** *blast*
then have $l \leq b$
using *l-bound slast* **by** *auto*

```
    then have wpd-mltl (to-mltl α) ≤ length (drop l π)
      using l-bound α-wpd by simp
    then have α-ih: semantics-mltl-ext (drop l π) α
      using Suc(1)[OF α-welldef α-nnf α-composition, of drop l π ?D]
      using x-in x-semantics semantics-mltl-ext-def by blast
    have lb: a ≤ interval-times a L ! i
      using interval-times-diff-ge-general[OF a-leq-b L-composition, of i 0 ?s]
      using sfirst i-bound by auto
    have ?thesis unfolding Future-mltl-ext semantics-mltl-ext-def
      unfolding to-mltl.simps semantics-mltl.simps
        using length-π-ge-b a-leq-b l-bound α-ih lb bound unfolding seman-
tics-mltl-ext-def
      by (metis ‹l ≤ b› dual-order.trans order-le-less-trans)
  }
  ultimately show ?thesis using cases by blast
 next
  case (Global-mltl-ext a b L α)
  have α-welldef: intervals-welldef (to-mltl α) and
      a-leq-b: a ≤ b
    using Suc(2) unfolding Global-mltl-ext by simp-all
  have α-nnf: ∃φ-init. α = convert-nnf-ext φ-init
    using Suc(3) unfolding Global-mltl-ext
  by (metis convert-nnf-ext.simps(7) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(6))

  have α-composition: is-composition-MLTL α
      using Suc(4) unfolding Global-mltl-ext is-composition-MLTL.simps by
simp-all
  have α-convert: convert-nnf-ext α = α
    using α-nnf convert-nnf-ext-convert-nnf-ext by metis
  have α-wpd: length π ≥ b + wpd-mltl (to-mltl α)
    using Suc(5) unfolding Global-mltl-ext to-mltl.simps wpd-mltl.simps
    by simp-all
  then have length-π-ge-b: length π > b
    using wpd-geq-one[of to-mltl α] by auto
  obtain ψ where ψ-in: ψ ∈ set D
          and ψ-semantics: semantics-mltl-ext π ψ
    using Suc(7) by blast
  let ?D = LP-mltl-aux α k
  {
    assume ∗: length ?D ≤ 1
    have D = [Global-mltl-ext a b L α]
      using Suc(6) unfolding Global-mltl-ext LP-mltl-aux.simps
      using ∗ α-convert by auto
    then have ?thesis using Suc
      by (simp add: Global-mltl-ext)
  } moreover {
    assume ∗: length ?D > 1
    then have D-is: D = (Global-mltl-decomp ?D a (b − a) L)
      using Suc α-nnf α-convert unfolding Global-mltl-ext LP-mltl-aux.simps
```

by *simp*
obtain ψ **where** ψ-*in*: $\psi \in$ *set* (*Global-mltl-decomp ?D a* $(b - a)$ *L*)
and ψ-*semantics*: *semantics-mltl-ext* π ψ
using *Suc*(7) **unfolding** *D-is* **by** *blast*
then obtain *X* **where** ψ-*is*: ψ = *Ands-mltl-ext X*
and *X-fact*: $\forall i{<}$*length X*. $\exists y{\in}$*set* (*LP-mltl-aux* α *k*).
X ! *i* = *Global-mltl-ext* $(a + i)$ $(a + i)$ [*1*] *y*
and *length-X*: *length X* = *Suc* $(b - a)$
using *in-Global-mltl-decomp-exact-forward*[*OF* $*$ ψ-*in*] **by** *blast*
have *semantics-mltl* (*drop i* π) (*to-mltl* α)
if *i-bound*: $a \leq i \wedge i \leq b$ **for** *i*
proof−
have $i{-}a <$ *length X*
using *i-bound length-X a-leq-b* **by** *linarith*
then obtain *y* **where** *y-in*: *y* ∈ *set ?D*
and *Xi-is*: *X*!$(i{-}a)$ = *Global-mltl-ext* $(a{+}i{-}a)$ $(a{+}i{-}a)$ [*1*] *y*
using *X-fact i-bound* **by** *auto*
have *semantics-mltl-ext* (*drop i* π) *y*
proof−
have *i-length-trace*: *i*< *length* π
using *i-bound length-*π*-ge-b* **by** *auto*
have *Ands-semantics*: $(\forall x{\in}$*set X*. *semantics-mltl-ext* π *x*)
using ψ-*semantics* **unfolding** ψ-*is*
using *Ands-mltl-semantics*[*of X* π] *length-X* **by** *auto*
have (*Global-mltl-ext i i* [*1*] *y*) ∈ *set X*
using *Xi-is i-bound* ‹*i* − *a* < *length X*› *nth-mem* **by** *fastforce*
then have *semantics-mltl-ext* π (*Global-mltl-ext i i* [*1*] *y*)
using *Ands-semantics* **by** *blast*
then show *?thesis* **unfolding** *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *i-length-trace* **by** *simp*
qed
then have *semantics*: $\exists a{\in}$*set ?D*. *semantics-mltl-ext* (*drop i* π) *a*
using *y-in* **by** *blast*
have *wpd*: *wpd-mltl* (*to-mltl* α) ≤ *length* (*drop i* π)
using *length-*π*-ge-b* α*-wpd i-bound* **by** *auto*
show *?thesis*
using *Suc*(*1*)[*OF* α*-welldef* α*-nnf* α*-composition*, *of drop i* π *?D*]
using *wpd semantics* **unfolding** *semantics-mltl-ext-def* **by** *blast*
qed
then have *?thesis* **unfolding** *Global-mltl-ext semantics-mltl-ext-def semantics-mltl.simps to-mltl.simps*
using *a-leq-b length-*π*-ge-b* **by** *blast*
}
ultimately show *?thesis* **by** *linarith*
next
case (*Until-mltl-ext* α *a b L* β)
have α-*welldef*: *intervals-welldef* (*to-mltl* α) **and**
β-*welldef*: *intervals-welldef* (*to-mltl* β) **and**

```
    a-leq-b: a ≤ b
   using Suc(2) unfolding Until-mltl-ext by simp-all
  have α-nnf: ∃φ-init. α = convert-nnf-ext φ-init
   using Suc(3) unfolding Until-mltl-ext
  by (metis convert-nnf-ext.simps(8) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(7))
  have β-nnf: ∃φ-init. β = convert-nnf-ext φ-init
   using Suc(3) unfolding Until-mltl-ext
  by (metis convert-nnf-ext.simps(8) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(7))
  have α-composition: is-composition-MLTL α and
    β-composition: is-composition-MLTL β and
    L-composition: is-composition (b−a+1) L
  using Suc(4) unfolding Until-mltl-ext is-composition-MLTL.simps by simp-all
  have α-convert: convert-nnf-ext α = α
   using α-nnf convert-nnf-ext-convert-nnf-ext by metis
  have β-convert: convert-nnf-ext β = β
   using β-nnf convert-nnf-ext-convert-nnf-ext by metis
  have α-wpd: length π ≥ b+wpd-mltl (to-mltl α)−1 and
    β-wpd: length π ≥ b+wpd-mltl (to-mltl β)
   using Suc(5) unfolding Until-mltl-ext to-mltl.simps wpd-mltl.simps
   by simp-all
  then have length-π-ge-b: length π > b
   using wpd-geq-one[of to-mltl β] by auto
  obtain ψ where ψ-in: ψ ∈ set D
          and ψ-semantics: semantics-mltl-ext π ψ
   using Suc(7) by blast
  let ?D = LP-mltl-aux β k
  let ?s = interval-times a L
  have length-L: 1 ≤ length L
   using composition-length-lb[OF L-composition] a-leq-b by linarith
  have sfirst: ?s!0 = a
   using interval-times-first by simp
  have slast: ?s!(length L) = b+1
   using interval-times-last[OF a-leq-b L-composition] by blast
  let ?front = (Until-mltl-list α ?D (?s ! 0) (?s ! 1 − 1) [?s ! 1 − ?s ! 0])
  let ?back = (concat (map (λi. And-mltl-list
                    [Global-mltl-ext
                      (?s ! 0) (?s ! i − 1) [?s!i − ?s!0] (And-mltl-ext α (Not_c
β))]
                    (Until-mltl-list α ?D (?s ! i) (?s ! (i + 1) − 1)
                      [?s ! (i + 1) − ?s ! i])) [1..<length L]))
  have D-union: set D = (set ?front) ∪ (set ?back)
   using Suc(6) unfolding Until-mltl-ext LP-mltl-aux.simps
   using α-convert β-convert list-concat-set-union by metis
  obtain ψ where ψ-in: ψ ∈ set D and ψ-semantics: semantics-mltl-ext π ψ
   using Suc(7) by blast
  {
   assume ∗: ψ ∈ set ?front
   then obtain y where ψ-is: ψ = Until-mltl-ext α (interval-times a L ! 0)
            (interval-times a L ! 1 − 1) [interval-times a L ! 1 − interval-times
```

a L ! 0] *y*
and *y-in*: *y* ∈ *set ?D*
by *auto*
have *length-s*: *1 < length ?s* **using** *ψ-is*
by (*metis One-nat-def add.commute add-gr-0 add-less-cancel-right L-composition composition-length-lb interval-times-length plus-1-eq-Suc zero-less-one*)
then have *length-L*: *1 ≤ length L*
unfolding *interval-times-def*
by (*simp add*: *less-eq-iff-succ-less*)
have *interval-times a L ! 1 ≤ interval-times a L ! (length L)*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L 1 ?s*]
using *length-L* **by** *force*
then have *ub*: *interval-times a L ! 1 − 1 ≤ b*
using *slast* **by** *auto*
obtain *l* **where** *y-semantics*: *semantics-mltl-ext (drop l π) y*
and *α-global*: (∀ *j*. *interval-times a L ! 0 ≤ j ∧ j < l* ⟶
semantics-mltl (drop j π) (to-mltl α))
and *l-bound*: *?s ! 0 ≤ l ∧ l ≤ ?s ! 1 − 1*
using *ψ-semantics* **unfolding** *ψ-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *blast*
have *l-ab*: *a ≤ l ∧ l ≤ b*
using *l-bound sfirst ub* **by** *simp*
have *sem*: ∃ *a*∈*set* (*LP-mltl-aux β k*). *semantics-mltl-ext (drop l π) a*
using *y-in y-semantics* **by** *blast*
have *wpd-mltl (to-mltl β) ≤ length (drop l π)*
using *l-bound length-π-ge-b β-wpd ub* **by** *auto*
then have *ih*: *semantics-mltl-ext (drop l π) β*
using *Suc(1)*[*OF β-welldef β-nnf β-composition*, *of drop l π ?D*]
using *sem* **by** *blast*
have *semantics-mltl (drop j π) (to-mltl α)*
if *j-bound*: *a ≤ j ∧ j < l* **for** *j*
using *α-global* **unfolding** *sfirst* **using** *j-bound l-bound ub* **by** *blast*
then have (∃ *i*. (*a ≤ i ∧ i ≤ b*) ∧
semantics-mltl (drop i π) (to-mltl β) ∧
(∀ *j*. *a ≤ j ∧ j < i* ⟶
semantics-mltl (drop j π) (to-mltl α)))
using *ih l-ab* **unfolding** *semantics-mltl-ext-def* **by** *blast*
then have *?thesis* **unfolding** *Until-mltl-ext semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *a-leq-b length-π-ge-b* **by** *simp*
} moreover {
assume ∗: *ψ* ∈ *set ?back*
then obtain *i y* **where**
ψ-is: *ψ = And-mltl-ext (Global-mltl-ext (?s!0) (?s!i−1) [?s!i − ?s!0] (And-mltl-ext α (Not$_c$ β)))*
(Until-mltl-ext α (?s!i) (?s!(i+1)−1) [(?s!(i+1)) − (?s!i)] y)
and *i-bound*: *1 ≤ i ∧ i < length L*

and *y-in*: $y \in set\ ?D$
by *auto*
have *bound1*: *interval-times a L ! i < interval-times a L ! (i+1)*
using *interval-times-diff-ge*[*OF a-leq-b L-composition*, *of i ?s*]
using *i-bound* **by** *blast*
have *interval-times a L ! (i + 1) ≤ interval-times a L ! (length L)*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L i+1 ?s*]
using *i-bound* **by** (*metis less-iff-succ-less-eq order-le-less*)
then have *bound2*: *interval-times a L ! (i+1) ≤ b+1*
using *slast* **by** *simp*
have *interval-times a L ! i > interval-times a L ! 0*
using *i-bound interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i 0 ?s*]
by *auto*
then have *interval-times a L ! i > 0*
unfolding *interval-times-def* **by** *simp*
then have *interval-times a L ! i ≤ b*
using *bound1 bound2* **by** *simp*
have $\alpha\beta$-*global*: $(\forall\, ia.\ a \leq ia \wedge ia \leq$ *interval-times a L ! i − 1* $\longrightarrow$
semantics-mltl (*drop ia* π) (*to-mltl* α) $\wedge$
$\neg$ *semantics-mltl* (*drop ia* π) (*to-mltl* β))
using ψ-*semantics* **unfolding** ψ-*is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
unfolding *sfirst* **by** *auto*
have *until*: $(\exists\, ia.$ (*interval-times a L ! i ≤ ia* $\wedge$
ia ≤ interval-times a L ! (i + 1) − 1) $\wedge$
semantics-mltl (*drop ia* π) (*to-mltl y*) $\wedge$
$(\forall\, j.$ *interval-times a L ! i ≤ j* $\wedge$ *j < ia* $\longrightarrow$
semantics-mltl (*drop j* π) (*to-mltl* α)))
using ψ-*semantics* **unfolding** ψ-*is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
unfolding *sfirst* **by** *auto*
obtain *l* **where** *y-semantics*: *semantics-mltl-ext* (*drop l* π) *y*
and α-*global*: $(\forall\, j.$ *?s ! i ≤ j* $\wedge$ *j < l* $\longrightarrow$
semantics-mltl (*drop j* π) (*to-mltl* α))
and *l-bound*: *?s ! i ≤ l* $\wedge$ *l ≤ ?s ! (i+1) − 1*
using *until* **unfolding** *semantics-mltl-ext-def* **by** *blast*
have *ub*: *?s ! (i+1) − 1 ≤ b*
using *i-bound bound2* **by** *auto*
have *lb*: *a < ?s!i*
using *i-bound interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i 0 ?s*]
using *sfirst* **by** *auto*
have *l-ab*: *a ≤ l* $\wedge$ *l ≤ b*
using *l-bound* **using** *ub lb* **by** *simp*
have *sem*: $\exists\, a{\in}set$ (*LP-mltl-aux* β *k*). *semantics-mltl-ext* (*drop l* π) *a*
using *y-in y-semantics* **by** *blast*
have *wpd-mltl* (*to-mltl* β) ≤ *length* (*drop l* π)

```
      using β-wpd l-bound length-π-ge-b ub by auto
    then have ih: semantics-mltl-ext (drop l π) β
      using Suc(1)[OF β-welldef β-nnf β-composition - - sem] by blast
    have l-ab: a ≤ l ∧ l ≤ b
      using l-bound lb ub by simp
    have semantics-mltl (drop j π) (to-mltl α)
      if j-bound: a ≤ j ∧ j < l for j
    proof−
      have case1: ∀ ia. a ≤ ia ∧ ia ≤ ?s ! i − 1 ⟶
       semantics-mltl (drop ia π) (to-mltl α)
        using αβ-global by blast
      {
        assume ∗: a ≤ j ∧ j ≤ ?s ! i − 1
        then have ?thesis
          using case1 by blast
      } moreover {
        assume ∗: ?s!i ≤ j ∧ j < l
        then have ?thesis
          using α-global by blast
      }
      ultimately show ?thesis using j-bound by linarith
    qed
    then have (∃ i. (a ≤ i ∧ i ≤ b) ∧
      semantics-mltl (drop i π) (to-mltl β) ∧
      (∀ j. a ≤ j ∧ j < i ⟶
          semantics-mltl (drop j π) (to-mltl α)))
      using ih l-ab semantics-mltl-ext-def by auto
  then have ?thesis unfolding Until-mltl-ext semantics-mltl-ext-def to-mltl.simps
semantics-mltl.simps
      using a-leq-b length-π-ge-b by simp
  }
  ultimately show ?thesis using D-union ψ-in by blast
 next
  case (Release-mltl-ext α a b L β)
  have α-welldef: intervals-welldef (to-mltl α) and
       β-welldef: intervals-welldef (to-mltl β) and
       a-leq-b: a ≤ b
    using Suc(2) unfolding Release-mltl-ext by simp-all
  have α-nnf: ∃ φ-init. α = convert-nnf-ext φ-init
    using Suc(3) unfolding Release-mltl-ext
  by (metis convert-nnf-ext.simps(9) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(8))
  have β-nnf: ∃ φ-init. β = convert-nnf-ext φ-init
    using Suc(3) unfolding Release-mltl-ext
  by (metis convert-nnf-ext.simps(9) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(8))
  have α-composition: is-composition-MLTL α and
       β-composition: is-composition-MLTL β and
       L-composition: is-composition (b−a+1) L
      using Suc(4) unfolding Release-mltl-ext is-composition-MLTL.simps by
simp-all
```

have α-*convert*: *convert-nnf-ext* $\alpha = \alpha$
using α-*nnf convert-nnf-ext-convert-nnf-ext* **by** *metis*
have β-*convert*: *convert-nnf-ext* $\beta = \beta$
using β-*nnf convert-nnf-ext-convert-nnf-ext* **by** *metis*
have α-*wpd*: *length* $\pi \geq b$+*wpd-mltl* (*to-mltl* α) **and**
β-*wpd*: *length* $\pi \geq b$+*wpd-mltl* (*to-mltl* β)
using *Suc*(*5*) **unfolding** *Release-mltl-ext to-mltl.simps wpd-mltl.simps*
by *simp-all*
then have *length-*π*-ge-b*: *length* $\pi > b$
using *wpd-geq-one*[*of to-mltl* β] **by** *auto*
obtain ψ **where** ψ-*in*: $\psi \in$ *set D*
and ψ-*semantics*: *semantics-mltl-ext* π ψ
using *Suc*(*7*) **by** *blast*
let *?D* = *LP-mltl-aux* α *k*
let *?s* = *interval-times a L*
have *length-L*: *1* $\leq$ *length L*
using *composition-length-lb*[*OF L-composition*] *a-leq-b* **by** *linarith*
have *sfirst*: *?s*!*0* = *a*
using *interval-times-first* **by** *simp*
have *slast*: *?s*!(*length L*) = *b*+*1*
using *interval-times-last*[*OF a-leq-b L-composition*] **by** *blast*
let *?front* = *set* [*Global-mltl-ext a b L* (*And-mltl-ext* (*Not*$_c$ α) β)]
let *?middle* = *set* (*Mighty-Release-mltl-list ?D* β (*?s* ! *0*) (*?s* ! *1* − *1*)
[*?s* ! *1* − *?s* ! *0*])
let *?back* = *set* (*concat* (*map* (λ*i*. *And-mltl-list*
[*Global-mltl-ext*
(*?s* ! *0*) (*?s* ! *i* − *1*) [*?s*!*i* − *?s*!*0*] (*And-mltl-ext* (*Not*$_c$ α)
β)]
(*Mighty-Release-mltl-list ?D* β (*?s* ! *i*)
(*?s* ! (*i* + *1*) − *1*) [*?s* ! (*i* + *1*) − *?s* ! *i*]))
[*1*..<*length L*]))
let *?P* = λ*j*. (*semantics-mltl* (*drop j* π) (*to-mltl* α) $\wedge$
($\forall$ *k*. *a* $\leq$ *k* $\wedge$ *k* $\leq$ *j* $\longrightarrow$
semantics-mltl (*drop k* π) (*to-mltl* β)))
have *D-is*: *set D* = *?front* $\cup$ *?middle* $\cup$ *?back*
unfolding *Suc*(*6*) *Release-mltl-ext LP-mltl-aux.simps*
using α-*convert list-concat-set-union*
by (*metis append-assoc*)
have *split*: $\psi \in$ *?front* $\cup$ *?middle* $\cup$ *?back*
using ψ-*in D-is* **by** *blast*
{
assume $*$: $\psi \in$ *?front*
then have ψ-*is*: ψ = *Global-mltl-ext a b L* (*And-mltl-ext* (*Not*$_c$ α) β)
by *auto*
then have *?thesis* **using** ψ-*semantics* **unfolding** ψ-*is*
unfolding *Release-mltl-ext semantics-mltl-ext-def to-mltl.simps seman-*
tics-mltl.simps
by *blast*
} moreover {

assume $*$: $\psi \in$ *?middle*
then obtain x **where** ψ*-is*: $\psi =$ *Mighty-Release-mltl-ext* $x\ \beta\ a$ (*?s* ! *1* − *1*) [*?s* ! *1* − a]
and *x-in*: $x \in$ *set ?D*
using *sfirst* **by** *auto*
have *welldef*: $a <$ *?s*!*1* **using** *sfirst*
using *interval-times-diff-ge*[*OF a-leq-b L-composition*, *of 0 ?s*]
using *length-L* **by** *force*
have *ub*: *?s*!*1* ≤ *b*+*1*
using *length-L slast*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L 1 ?s*]
by *force*
obtain i **where** *i-bound*: $a \leq i \wedge i \leq$ *interval-times a L* ! *1* − *1*
and *x-semantics*: *semantics-mltl* (*drop i* π) (*to-mltl x*)
using ψ*-semantics* **unfolding** ψ*-is Mighty-Release-mltl-ext.simps*
unfolding *Release-mltl-ext semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *auto*
have *wpd-mltl* (*to-mltl* α) ≤ *length* (*drop i* π)
using α*-wpd i-bound ub* **by** *auto*
then have α*-semantics*: *semantics-mltl-ext* (*drop i* π) α
using *Suc*(*1*)[*OF* α*-welldef* α*-nnf* α*-composition*, *of drop i* π *?D*]
using *x-in x-semantics* **unfolding** *semantics-mltl-ext-def* **by** *blast*
let *?globally-*β = ($\forall\, i.\ a \leq i \wedge i \leq$ *interval-times a L* ! *1* − *1* ⟶ *semantics-mltl* (*drop i* π) (*to-mltl* β))
let *?release* = ($\exists\, j \geq a.\ j \leq$ *interval-times a L* ! *1* − *1* − *1* ∧ *semantics-mltl* (*drop j* π) (*to-mltl x*) ∧ ($\forall\, k.\ a \leq k \wedge k \leq j$ ⟶ *semantics-mltl* (*drop k* π) (*to-mltl* β)))
have *eo*: *?globally-*β ∨ *?release*
using ψ*-semantics* **unfolding** ψ*-is Mighty-Release-mltl-ext.simps*
unfolding *Release-mltl-ext semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *auto*
{
assume $**$: *?globally-*β
{
assume *interval-times a L* ! *1* − *1* = *b*
then have *?thesis* **unfolding** *Release-mltl-ext semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using $**$ *a-leq-b* **by** *simp*
} moreover {
assume *s1-bound*: *interval-times a L* ! *1* − *1* < *b*
have *semantics-mltl* (*drop k* π) (*to-mltl* β)
if *k-bound*: $a \leq k \wedge k \leq i$ **for** k
using $**$ *k-bound i-bound s1-bound* **by** *auto*
then have *?thesis* **using** $**$ α*-semantics i-bound ub a-leq-b*
unfolding *semantics-mltl-ext-def Release-mltl-ext to-mltl.simps seman-*

tics-mltl.simps
using *s1-bound* **by** *force*
}
ultimately have *?thesis* **using** *ub* **by** *linarith*
} moreover {
assume $**$: *?release*
have *bound*: *interval-times a L ! 1 − 1 − 1 ≤ b−1*
using *ub* **by** *simp*
then obtain *j* **where** *sem*: $a \leq j \wedge j \leq$ *interval-times a L ! 1 − 1 − 1* $\wedge$
semantics-mltl (*drop j* π) (*to-mltl x*) $\wedge$
($\forall$ *k*. $a \leq k \wedge k \leq j \longrightarrow$
semantics-mltl (*drop k* π) (*to-mltl* β))
using $**$ **by** *blast*
have *wpd-mltl* (*to-mltl* α) ≤ *length* (*drop j* π)
using α*-wpd sem ub* **by** *auto*
then have *semantics-mltl* (*drop j* π) (*to-mltl* α)
using *Suc*(*1*)[*OF* α*-welldef* α*-nnf* α*-composition*, *of drop j* π *?D*]
using *sem x-in* **unfolding** *semantics-mltl-ext-def* **by** *blast*
then have ($\exists$ *j*$\geq$*a*. *j ≤ b − 1* $\wedge$
semantics-mltl (*drop j* π) (*to-mltl* α) $\wedge$
($\forall$ *k*. $a \leq k \wedge k \leq j \longrightarrow$
semantics-mltl (*drop k* π) (*to-mltl* β)))
using *sem ub* **by** *auto*
then have *?thesis*
unfolding *semantics-mltl-ext-def Release-mltl-ext to-mltl.simps semantics-mltl.simps*
using *a-leq-b* **by** *blast*
}
ultimately have *?thesis* **using** *eo* **by** *blast*
} moreover {
assume $*$: $\psi \in$ *?back*
then obtain *i x* **where** ψ*-is*: ψ = *And-mltl-ext*
(*Global-mltl-ext*
(*interval-times a L ! 0*)
(*interval-times a L ! i − 1*) [*?s*!*i − ?s*!*0*] (*And-mltl-ext* (*Not*$_c$ α) β))
(*Mighty-Release-mltl-ext x* β
(*interval-times a L ! i*)
(*interval-times a L ! (i + 1) − 1*)
[*interval-times a L ! (i + 1) −*
interval-times a L ! i])
and *x-in*: *x ∈ set ?D*
and *i-bound*: *1 ≤ i ∧ i < length L*
by *auto*
have *lb*: *a < ?s*!*i*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i 0 ?s*]
using *sfirst i-bound* **by** *simp*
have *welldef*: (*interval-times a L ! i*) < (*interval-times a L ! (i + 1)*)
using *interval-times-diff-ge*[*OF a-leq-b L-composition*, *of i ?s*]

```
    using i-bound by simp
  have ub: ?s!(i+1) ≤ b+1
    using slast i-bound
     using interval-times-diff-ge-general[OF a-leq-b L-composition, of length L
i+1 ?s]
     by (metis Orderings.order-eq-iff less-iff-succ-less-eq order-le-imp-less-or-eq
order-less-imp-le)

  have globally-before: ∀ ia. interval-times a L ! 0 ≤ ia ∧ ia ≤ interval-times a
L ! i − 1 ⟶
      ¬ semantics-mltl (drop ia π) (to-mltl α) ∧
      semantics-mltl (drop ia π) (to-mltl β)
       using ψ-semantics unfolding ψ-is semantics-mltl-ext-def to-mltl.simps
semantics-mltl.simps Mighty-Release-mltl-ext.simps
    using length-π-ge-b a-leq-b sfirst by auto
  have release: (∀ ia. interval-times a L ! i ≤ ia ∧
      ia ≤ interval-times a L ! (i + 1) − 1 ⟶
      semantics-mltl (drop ia π) (to-mltl β)) ∨
 (∃ j≥interval-times a L ! i.
    j ≤ interval-times a L ! (i + 1) − 1 − 1 ∧
    semantics-mltl (drop j π) (to-mltl x) ∧
    (∀ k. interval-times a L ! i ≤ k ∧ k ≤ j ⟶
        semantics-mltl (drop k π) (to-mltl β)))
       using ψ-semantics unfolding ψ-is semantics-mltl-ext-def to-mltl.simps
semantics-mltl.simps Mighty-Release-mltl-ext.simps
    by auto
  obtain ia where ia-bound: interval-times a L ! i ≤ ia ∧
    ia ≤ interval-times a L ! (i + 1) − 1
                and x-semantics: semantics-mltl (drop ia π) (to-mltl x)
       using ψ-semantics unfolding ψ-is semantics-mltl-ext-def to-mltl.simps
semantics-mltl.simps Mighty-Release-mltl-ext.simps
    by blast
  have wpd-mltl (to-mltl α) ≤ length (drop ia π)
    using α-wpd ia-bound ub by auto
  then have α-semantics: semantics-mltl (drop ia π) (to-mltl α)
    using Suc(1)[OF α-welldef α-nnf α-composition, of drop ia π ?D]
    using x-semantics x-in unfolding semantics-mltl-ext-def by blast
  {
    assume global-β: (∀ ia. interval-times a L ! i ≤ ia ∧
      ia ≤ interval-times a L ! (i + 1) − 1 ⟶
      semantics-mltl (drop ia π) (to-mltl β))
    {
      assume eq: interval-times a L ! (i + 1) − 1 = b
      have semantics-mltl (drop j π) (to-mltl β)
        if j-bound: a ≤ j ∧ j ≤ b for j
      proof−
        have 1: j ≤ interval-times a L ! i − 1 ⟹ ?thesis
          using globally-before j-bound unfolding sfirst by blast
        have 2: j ≥ interval-times a L ! i ⟹ ?thesis
```

```
        using global-β j-bound eq by blast
      show ?thesis
        using 1 2 by linarith
    qed
    then have ?thesis
      unfolding Release-mltl-ext semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps
      using a-leq-b by blast
  } moreover {
    assume le: interval-times a L ! (i + 1) − 1 < b
    have 1: semantics-mltl (drop k π) (to-mltl β)
      if k-bound: a ≤ k ∧ k ≤ ia for k
    proof−
      have 1: k ≤ interval-times a L ! i − 1 ⟹ ?thesis
        using globally-before k-bound sfirst ia-bound by auto
      have 2: k ≥ interval-times a L ! i ⟹ ?thesis
        using global-β ia-bound k-bound by auto
      show ?thesis
        using 1 2 by linarith
    qed
    have 2: a ≤ ia ∧ ia ≤ b − 1
      using ia-bound ub lb le by auto
    then have (∃j≥a. j ≤ b − 1 ∧
      semantics-mltl (drop j π) (to-mltl α) ∧
      (∀k. a ≤ k ∧ k ≤ j ⟶
          semantics-mltl (drop k π) (to-mltl β)))
      using α-semantics ia-bound le ub lb welldef 1 2 by blast
    then have ?thesis
      unfolding Release-mltl-ext semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps
      using a-leq-b by auto
  }
  ultimately have ?thesis using ub by linarith
} moreover {
  assume (∃j≥interval-times a L ! i.
  j ≤ interval-times a L ! (i + 1) − 1 − 1 ∧
  semantics-mltl (drop j π) (to-mltl x) ∧
  (∀k. interval-times a L ! i ≤ k ∧ k ≤ j ⟶
      semantics-mltl (drop k π) (to-mltl β)))
then obtain j where j-bound: interval-times a L ! i ≤ j ∧ j ≤ interval-times a L ! (i + 1) − 1 − 1
                    and x-semantics: semantics-mltl (drop j π) (to-mltl x)
                    and global: ∀k. interval-times a L ! i ≤ k ∧ k ≤ j ⟶
      semantics-mltl (drop k π) (to-mltl β)
    by blast
  have wpd-mltl (to-mltl α) ≤ length (drop j π)
    using α-wpd j-bound ub by auto
  then have α-semantics: semantics-mltl (drop j π) (to-mltl α)
    using Suc(1)[OF α-welldef α-nnf α-composition, of drop j π ?D]
```

using *x-in x-semantics* **unfolding** *semantics-mltl-ext-def* **by** *blast*
have *g*: *semantics-mltl* (*drop k π*) (*to-mltl β*)
if *k-bound*: $a \leq k \wedge k \leq j$ **for** *k*
proof−
have *1*: $k \leq$ *interval-times a L* ! $i - 1 \Longrightarrow$ *?thesis*
using *globally-before k-bound sfirst ia-bound* **by** *auto*
have *2*: $k \geq$ *interval-times a L* ! $i \Longrightarrow$ *?thesis*
using *global ia-bound k-bound* **by** *auto*
show *?thesis*
using *1 2* **by** *linarith*
qed
have $a \leq j \wedge j \leq b - 1$
using *j-bound ub lb* **by** *auto*
then have $(\exists\, j{\geq}a.\ j \leq b - 1 \wedge$
semantics-mltl (*drop j π*) (*to-mltl α*) $\wedge$
$(\forall\, k.\ a \leq k \wedge k \leq j \longrightarrow$
semantics-mltl (*drop k π*) (*to-mltl β*)))
using *α-semantics g* **by** *blast*
then have *?thesis*
unfolding *Release-mltl-ext semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *a-leq-b* **by** *blast*
}
ultimately have *?thesis* **using** *release* **by** *blast*
}
ultimately show *?thesis* **using** *split* **by** *blast*
qed
qed

Top Level Union Theorem **lemma** *LP-mltl-aux-language-union*:
fixes *φ*::*'a mltl-ext* **and** *k*::*nat* **and** *π*::*'a set list*
assumes *intervals-welldef*: *intervals-welldef* (*to-mltl φ*)
assumes *is-nnf*: $\exists\, \varphi\text{-init}.\ \varphi =$ *convert-nnf-ext φ-init*
assumes *trace-length*: *length π* $\geq$ *wpd-mltl* (*to-mltl φ*)
assumes *composition*: *is-composition-MLTL φ*
assumes *D-is*: *D* = *LP-mltl-aux φ k*
shows *semantics-mltl-ext π φ* $\longleftrightarrow$
$(\exists\, \psi \in$ *set D*. *semantics-mltl-ext π ψ*)
using *assms*
using *LP-mltl-aux-language-union-converse*
using *LP-mltl-aux-language-union-forward* **by** *fast*

theorem *LP-mltl-language-union-explicit*:
fixes *φ*::*'a mltl-ext* **and** *k*::*nat* **and** *π*::*'a set list*
assumes *intervals-welldef*: *intervals-welldef* (*to-mltl φ*)
assumes *composition*: *is-composition-MLTL φ*
assumes *D-is*: *D* = *set* (*LP-mltl φ k*)
assumes *trace-length*: *length π* $\geq$ *wpd-mltl* (*to-mltl φ*)
shows *semantics-mltl-ext π φ* $\longleftrightarrow$ $(\exists\, \psi{\in}D.$ *semantics-mltl π ψ*)

proof−
have $D = set\ (map\ to\text{-}mltl$
$(map\ convert\text{-}nnf\text{-}ext\ (LP\text{-}mltl\text{-}aux\ (convert\text{-}nnf\text{-}ext\ \varphi)\ k)))$
using *D-is* **unfolding** *LP-mltl.simps* **by** *blast*
let *?D-aux* = $LP\text{-}mltl\text{-}aux\ (convert\text{-}nnf\text{-}ext\ \varphi)\ k$
let *?φ-nnf* = $convert\text{-}nnf\text{-}ext\ \varphi$
have *wpd-decomp*: $wpd\text{-}mltl\ \psi \leq wpd\text{-}mltl\ (to\text{-}mltl\ \varphi)$
if $\psi\text{-}in : \psi \in D$ **for** ψ
proof−
obtain x **where** $\psi\text{-}is$: $\psi = to\text{-}mltl\ (convert\text{-}nnf\text{-}ext\ x)$
and *x-in*: $x \in set\ (LP\text{-}mltl\text{-}aux\ (convert\text{-}nnf\text{-}ext\ \varphi)\ k)$
using $\psi\text{-}in$ **unfolding** *D-is LP-mltl.simps* **by** *auto*
have *xphi*: $wpd\text{-}mltl\ (to\text{-}mltl\ x) \leq wpd\text{-}mltl\ (to\text{-}mltl\ \varphi)$
using $LP\text{-}mltl\text{-}aux\text{-}wpd[of\ (convert\text{-}nnf\text{-}ext\ \varphi)\ x\ k]$
by (*metis composition convert-nnf-ext-to-mltl-commute intervals-welldef is-composition-convert-nnf-ext nnf-intervals-welldef wpd-convert-nnf x-in*)
have $wpd\text{-}mltl\ (to\text{-}mltl\ x) = wpd\text{-}mltl\ \psi$
unfolding $\psi\text{-}is$ **using** *convert-nnf-ext-to-mltl-commute*
by (*metis wpd-convert-nnf*)
then show *?thesis* **using** *xphi* **by** *auto*
qed
have *len-biconditional*: $\bigwedge\pi.\ length\ \pi \geq wpd\text{-}mltl\ (to\text{-}mltl\ \varphi) \Longrightarrow$
$(semantics\text{-}mltl\ \pi\ (to\text{-}mltl\ \varphi) \longleftrightarrow (\exists\,\psi{\in}D.\ semantics\text{-}mltl\ \pi\ \psi))$
proof−
fix π::$'a\ set\ list$
assume ∗: $length\ \pi \geq wpd\text{-}mltl\ (to\text{-}mltl\ \varphi)$
let *?thesis* = $semantics\text{-}mltl\ \pi\ (to\text{-}mltl\ \varphi) \longleftrightarrow$
$(\exists\,\psi{\in}D.\ semantics\text{-}mltl\ \pi\ \psi)$
have $intervals\text{-}welldef\ (convert\text{-}nnf\ (to\text{-}mltl\ \varphi))$
using *intervals-welldef nnf-intervals-welldef* **by** *blast*
then have *cond1*: $intervals\text{-}welldef\ (to\text{-}mltl\ (convert\text{-}nnf\text{-}ext\ \varphi))$
by (*simp add*: *convert-nnf-ext-to-mltl-commute*)
have *?φ-nnf* = $convert\text{-}nnf\text{-}ext\ (?\varphi\text{-}nnf)$
using *convert-nnf-ext-convert-nnf-ext* **by** *blast*
then have *cond2*: $\exists\,\varphi\text{-}init.\ convert\text{-}nnf\text{-}ext\ \varphi = convert\text{-}nnf\text{-}ext\ \varphi\text{-}init$
by *blast*
have *cond3*: $wpd\text{-}mltl\ (to\text{-}mltl\ (convert\text{-}nnf\text{-}ext\ \varphi)) \leq length\ \pi$
proof−
have $wpd\text{-}mltl\ (convert\text{-}nnf\ (to\text{-}mltl\ \varphi)) \leq length\ \pi$
using ∗ **by** (*simp add*: *wpd-convert-nnf*)
then show *?thesis*
using *convert-nnf-ext-to-mltl-commute* **by** *metis*
qed
have *cond4*: $is\text{-}composition\text{-}MLTL\ (convert\text{-}nnf\text{-}ext\ \varphi)$
using *composition intervals-welldef is-composition-convert-nnf-ext*
by *blast*
have *aux-fact*: $semantics\text{-}mltl\text{-}ext\ \pi\ (convert\text{-}nnf\text{-}ext\ \varphi) =$
$(\exists\,\psi{\in}set\ (LP\text{-}mltl\text{-}aux\ (convert\text{-}nnf\text{-}ext\ \varphi)\ k).\ semantics\text{-}mltl\text{-}ext\ \pi\ \psi)$
using $LP\text{-}mltl\text{-}aux\text{-}language\text{-}union[OF\ cond1\ cond2\ cond3\ cond4]$ **by** *blast*

have *forward*: ($\exists\psi{\in}$*set* (*LP-mltl-aux* (*convert-nnf-ext* φ) k).
semantics-mltl π (*to-mltl* ψ)) $\Longrightarrow$
($\exists\psi{\in}$*set* (*map to-mltl*
(*map convert-nnf-ext* (*LP-mltl-aux* (*convert-nnf-ext* φ) k))).
semantics-mltl π ψ)
proof −
assume $\exists\psi{\in}$*set* (*LP-mltl-aux* (*convert-nnf-ext* φ) k).
semantics-mltl π (*to-mltl* ψ)
then obtain ψ **where** $*$: $\psi{\in}$*set* (*LP-mltl-aux* (*convert-nnf-ext* φ) k) **and**
$**$: *semantics-mltl* π (*to-mltl* ψ)
by *blast*
have *in-set*: (*to-mltl* (*convert-nnf-ext* ψ)) $\in$ *set* (*map to-mltl*
(*map convert-nnf-ext* (*LP-mltl-aux* (*convert-nnf-ext* φ) k)))
using $*$ **by** *auto*
have *intervals-welldef* (*to-mltl* ψ)
using *intervals-welldef* $*$
using *LP-mltl-aux-intervals-welldef*
using *composition* **by** *auto*
then have *semantics-mltl* π (*convert-nnf* (*to-mltl* ψ))
using $**$ *convert-nnf-preserves-semantics*[*of to-mltl* ψ π]
by *blast*
then have *semantics*: *semantics-mltl* π (*to-mltl* (*convert-nnf-ext* ψ))
by (*simp add*: *convert-nnf-ext-to-mltl-commute*)
show *?thesis* **using** *in-set semantics* **by** *blast*
qed
have *converse*: ($\exists\psi{\in}$*set* (*map to-mltl*
(*map convert-nnf-ext* (*LP-mltl-aux* (*convert-nnf-ext* φ) k))).
semantics-mltl π ψ) $\Longrightarrow$ ($\exists\psi{\in}$*set* (*LP-mltl-aux* (*convert-nnf-ext* φ) k).
semantics-mltl π (*to-mltl* ψ))
proof −
assume $\exists\psi{\in}$*set* (*map to-mltl*
(*map convert-nnf-ext* (*LP-mltl-aux* (*convert-nnf-ext* φ) k))).
semantics-mltl π ψ
then obtain ψ **where** $*$: $\psi{\in}$*set* (*map to-mltl*
(*map convert-nnf-ext* (*LP-mltl-aux* (*convert-nnf-ext* φ) k)))
and $**$: *semantics-mltl* π ψ
by *blast*
obtain ψ-*aux* **where** *aux-in*: ψ-*aux* $\in$ *set* (*LP-mltl-aux* (*convert-nnf-ext* φ)
k) **and**
is-aux: ψ = *to-mltl* (*convert-nnf-ext* ψ-*aux*)
using $*$ *D-is LP-mltl-element* ‹D = *set* (*map to-mltl* (*map convert-nnf-ext*
(*LP-mltl-aux* (*convert-nnf-ext* φ) k)))› **by** *blast*
have *semantics*: *semantics-mltl* π (*to-mltl* ψ-*aux*)
using $**$ **unfolding** *is-aux*
by (*metis LP-mltl-aux-intervals-welldef aux-in composition convert-nnf-ext-to-mltl-commute*
convert-nnf-preserves-semantics intervals-welldef)
show *?thesis* **using** *aux-in semantics* **by** *blast*
qed
have ($\exists\psi{\in}$*set* (*LP-mltl-aux* (*convert-nnf-ext* φ) k).

semantics-mltl π (*to-mltl* ψ)) =
(∃ ψ∈*set* (*map to-mltl*
(*map convert-nnf-ext* (*LP-mltl-aux* (*convert-nnf-ext* φ) *k*))).
semantics-mltl π ψ)
using *forward converse* **by** *blast*
then show *?thesis*
unfolding *D-is LP-mltl.simps semantics-mltl-ext-def*
using *aux-fact convert-nnf-ext-to-mltl-commute convert-nnf-preserves-semantics*
by (*metis intervals-welldef semantics-mltl-ext-def*)
qed
show *?thesis*
using *len-biconditional*[*of* π] *assms*(*4*)
unfolding *semantics-mltl-ext-def* **by** *blast*
qed

theorem *LP-mltl-language-union*:
fixes φ::'*a mltl-ext* **and** *k*::*nat*
assumes *intervals-welldef*: *intervals-welldef* (*to-mltl* φ)
assumes *composition*: *is-composition-MLTL* φ
assumes *D-is*: *D* = *set* (*LP-mltl* φ *k*)
assumes *r*: *r* = *wpd-mltl* (*to-mltl* φ)
shows *language-mltl-r* (*to-mltl* φ) *r*
= (⋃ ψ∈*D*. *language-mltl-r* ψ *r*)
proof−
have π ∈ *language-mltl-r* (*to-mltl* φ) *r* ⟷
π ∈ (⋃ψ∈*D*. *language-mltl-r* ψ *r*)
if *length*: *length* π ≥ *r* **for** π
proof−
have *equiv*: (∃ ψ∈*D*. *semantics-mltl* π ψ) ⟷ π ∈ (⋃ψ∈*D*. *language-mltl-r* ψ *r*)
unfolding *language-mltl-r-def* **using** *length* **by** *blast*
have *semantics-mltl-ext* π φ = (∃ ψ∈*D*. *semantics-mltl* π ψ)
using *LP-mltl-language-union-explicit*[*of* φ *D k* π]
using *assms length* **by** *blast*
then show *?thesis*
using *equiv length*
unfolding *language-mltl-r-def semantics-mltl-ext-def* **by** *blast*
qed
then show *?thesis* **unfolding** *language-mltl-r-def*
by *blast*
qed

8.3 Disjointedness Theorem

lemma *LP-mltl-language-disjoint-aux-helper*:
fixes φ ψ*1* ψ*2*::'*a mltl-ext* **and** *k*::*nat* **and** π::'*a set list*
assumes *intervals-welldef*: *intervals-welldef* (*to-mltl* φ)
assumes *is-nnf*: ∃ φ-*init*. φ = *convert-nnf-ext* φ-*init*
assumes *composition-allones*: *is-composition-MLTL-allones* φ

assumes *tracelen*: *length* $\pi \geq$ *wpd-mltl* (*to-mltl* φ)
assumes *D-decomp*: $D = set\ (LP\text{-}mltl\text{-}aux\ \varphi\ k)$
assumes *diff-formulas*: $(\psi 1 \in D) \wedge (\psi 2 \in D) \wedge \psi 1 \neq \psi 2$
assumes *sat1*: *semantics-mltl-ext* $\pi\ \psi 1$
assumes *sat2*: *semantics-mltl-ext* $\pi\ \psi 2$
shows *False*
using *assms*
proof(*induction k arbitrary*: $D\ \varphi\ \psi 1\ \psi 2\ \pi$)
case *0*
then show *?case* **unfolding** *LP-mltl.simps LP-mltl-aux.simps*
by *auto*
next
case (*Suc k*)
then show *?case*
proof(*cases* φ)
case *True-mltl-ext*
then show *?thesis* **using** *Suc*
unfolding *True-mltl-ext LP-mltl.simps LP-mltl-aux.simps*
by *auto*
next
case *False-mltl-ext*
then show *?thesis* **using** *Suc*
unfolding *False-mltl-ext LP-mltl.simps LP-mltl-aux.simps*
by *auto*
next
case (*Prop-mltl-ext p*)
then show *?thesis* **using** *Suc*
unfolding *Prop-mltl-ext LP-mltl.simps LP-mltl-aux.simps*
by *auto*
next
case (*Not-mltl-ext q*)
then have $\exists p.\ q =$ *Prop-mltl-ext p*
using *convert-nnf-form-Not-Implies-Prop Suc*
by (*metis convert-nnf-ext-to-mltl-commute to-mltl.simps*(*4*) *to-mltl-prop-bijective*)

then obtain *p* **where** $q =$ *Prop-mltl-ext p* **by** *blast*
then show *?thesis*
using *Suc* **unfolding** *Not-mltl-ext LP-mltl.simps LP-mltl-aux.simps*
by *auto*
next
case (*And-mltl-ext* $\alpha\ \beta$)
let *?Dx* = *LP-mltl-aux* $\alpha\ k$
let *?Dy* = *LP-mltl-aux* $\beta\ k$
obtain *x1 y1* **where** $\psi 1\text{-}is$: $\psi 1 =$ *And-mltl-ext x1 y1*
and *x1-in*: *x1* $\in$ *set ?Dx* **and** *y1-in*: *y1* $\in$ *set ?Dy*
using *And-mltl-list-member Suc.prems*
by (*metis* (*no-types*, *lifting*) *And-mltl-ext LP-mltl-aux.simps*(*6*) *convert-nnf-ext.simps*(*4*)
convert-nnf-ext-convert-nnf-ext in-set-member mltl-ext.inject(*3*))
obtain *x2 y2* **where** $\psi 2\text{-}is$: $\psi 2 =$ *And-mltl-ext x2 y2*

and *x2-in*: $x2 \in$ *set ?Dx* **and** *y2-in*: $y2 \in$ *set ?Dy*
using *And-mltl-list-member Suc.prems*
by (*metis* (*no-types*, *lifting*) *And-mltl-ext LP-mltl-aux.simps*(*6*) *convert-nnf-ext.simps*(*4*) *convert-nnf-ext-convert-nnf-ext in-set-member mltl-ext.inject*(*3*))
have *eo*: $x1 \neq x2 \vee y1 \neq y2$
using *Suc*(*7*) ψ*1-is* ψ*2-is* **by** *blast*
have α*iwd*: *intervals-welldef* (*to-mltl* α) **and**
β*iwd*: *intervals-welldef* (*to-mltl* β)
using *Suc*(*2*) **unfolding** *And-mltl-ext* **by** *simp-all*
have α*nnf*: $\exists \varphi$*-init*. $\alpha =$ *convert-nnf-ext* φ*-init*
using *Suc*(*3*) **unfolding** *And-mltl-ext*
by (*metis convert-nnf-ext.simps*(*4*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*3*))
have β*nnf*: $\exists \varphi$*-init*. $\beta =$ *convert-nnf-ext* φ*-init*
using *Suc*(*3*) **unfolding** *And-mltl-ext*
by (*metis convert-nnf-ext.simps*(*4*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*3*))
have α*is-comp-allones*: *is-composition-MLTL-allones* α **and**
β*is-comp-allones*: *is-composition-MLTL-allones* β
using *Suc*(*4*) **unfolding** *And-mltl-ext is-composition-MLTL-allones.simps*
by *simp-all*
have α*is-comp*: *is-composition-MLTL* α
using α*is-comp-allones allones-implies-is-composition-MLTL*
by *blast*
have β*is-comp*: *is-composition-MLTL* β
using β*is-comp-allones allones-implies-is-composition-MLTL*
by *blast*
have α*wpd*: *wpd-mltl* (*to-mltl* α) $\leq$ *length* π **and**
β*wpd*: *wpd-mltl* (*to-mltl* β) $\leq$ *length* π
using *Suc*(*5*) **unfolding** *And-mltl-ext* **by** *simp-all*
let *?r* = *wpd-mltl* (*to-mltl* α)
{
assume *xs-neq*: $x1 \neq x2$
have *x1-semantics*: *semantics-mltl-ext* π *x1*
using *Suc*(*8*) **unfolding** ψ*1-is semantics-mltl-ext-def* **by** *simp*
have *x2-semantics*: *semantics-mltl-ext* π *x2*
using *Suc*(*9*) **unfolding** ψ*2-is semantics-mltl-ext-def* **by** *simp*
have *?thesis*
using *Suc*(*1*)[*OF* α*iwd* α*nnf* α*is-comp-allones* α*wpd*, *of set ?Dx x1 x2*]
using α*wpd xs-neq x1-in x2-in x1-semantics x2-semantics* **by** *blast*
} moreover {
assume *ys-neq*: $y1 \neq y2$
have *y1-semantics*: *semantics-mltl-ext* π *y1*
using *Suc*(*8*) **unfolding** ψ*1-is semantics-mltl-ext-def* **by** *simp*
have *y2-semantics*: *semantics-mltl-ext* π *y2*
using *Suc*(*9*) **unfolding** ψ*2-is semantics-mltl-ext-def* **by** *simp*
have *?thesis*
using *Suc*(*1*)[*OF* β*iwd* β*nnf* β*is-comp-allones* β*wpd*, *of set ?Dy y1 y2*]
using β*wpd ys-neq y1-in y2-in y1-semantics y2-semantics* **by** *blast*
}

ultimately show *?thesis*
using *eo* **by** *argo*
next
case (*Or-mltl-ext* α β)
let *?Dx* = *LP-mltl-aux* (*convert-nnf-ext* α) *k*
let *?Dy* = *LP-mltl-aux* (*convert-nnf-ext* β) *k*
have *D-is*: *D* = *set* (*And-mltl-list ?Dx ?Dy* @
And-mltl-list [Not_c α] *?Dy* @
And-mltl-list ?Dx [Not_c β])
using *Suc*(*6*) **unfolding** *Or-mltl-ext LP-mltl-aux.simps*
by *metis*
then have ψ*1-eo*: *List.member* (*And-mltl-list ?Dx ?Dy*) ψ*1* $\vee$
List.member (*And-mltl-list* [Not_c α] *?Dy*) ψ*1* $\vee$
List.member (*And-mltl-list ?Dx* [Not_c β]) ψ*1*
using *Suc*(*7*) **by** (*simp add*: *member-def*)
have ψ*2-eo*: *List.member* (*And-mltl-list ?Dx ?Dy*) ψ*2* $\vee$
List.member (*And-mltl-list* [Not_c α] *?Dy*) ψ*2* $\vee$
List.member (*And-mltl-list ?Dx* [Not_c β]) ψ*2*
using *D-is Suc*(*7*) **by** (*simp add*: *member-def*)

have α*-iwd*: *intervals-welldef* (*to-mltl* α)
using *Suc*(*2*) **unfolding** *Or-mltl-ext* **by** *simp*
have α*-nnf*: $\exists \varphi$*-init*. α = *convert-nnf-ext* φ*-init*
using *Suc*(*3*) **unfolding** *Or-mltl-ext*
by (*metis convert-nnf-ext.simps*(*5*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*4*))
have α*-is-comp*: *is-composition-MLTL-allones* α
using *Suc*(*4*) **unfolding** *Or-mltl-ext* **by** *simp*
have α*-wpd*: *wpd-mltl* (*to-mltl* α) $\leq$ *length* π
using *Suc*(*5*) **unfolding** *Or-mltl-ext* **by** *simp*
have α*-conv-same*: *set* (*LP-mltl-aux* (*convert-nnf-ext* α) *k*) = *set* (*LP-mltl-aux* α *k*)
by (*metis* α*-nnf convert-nnf-ext-convert-nnf-ext*)

have β*-iwd*: *intervals-welldef* (*to-mltl* β)
using *Suc*(*2*) **unfolding** *Or-mltl-ext*
by *simp*
have β*-nnf*: $\exists \varphi$*-init*. β = *convert-nnf-ext* φ*-init*
using *Suc*(*3*) **unfolding** *Or-mltl-ext*
by (*metis convert-nnf-ext.simps*(*5*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*4*))
have β*-is-comp*: *is-composition-MLTL-allones* β
using *Suc*(*4*) **unfolding** *Or-mltl-ext*
by *simp*
have β*-wpd*: *wpd-mltl* (*to-mltl* β) $\leq$ *length* π
using *Suc*(*5*) **unfolding** *Or-mltl-ext* **by** *simp*
have β*-conv-same*: *set* (*LP-mltl-aux* (*convert-nnf-ext* β) *k*) = *set* (*LP-mltl-aux* β *k*)
by (*metis* β*-nnf convert-nnf-ext-convert-nnf-ext*)

{

assume *List.member* (*And-mltl-list ?Dx ?Dy*) $\psi 1$
then obtain *x1 y1* **where** $\psi 1$*-is*: $\psi 1 =$ *And-mltl-ext x1 y1*
and *x1y1*: (*x1* $\in$ *set ?Dx* $\wedge$ *y1* $\in$ *set ?Dy*)
using *And-mltl-list-member*
by (*metis in-set-member*)
have *x1-semantics*: *semantics-mltl-ext* π *x1* **and**
y1-semantics: *semantics-mltl-ext* π *y1*
using *Suc*(*8*) **unfolding** *semantics-mltl-ext-def* ψ*1-is* **by** *simp-all*
have α*-semantics*: *semantics-mltl-ext* π α **using** *LP-mltl-aux-language-union-converse*
by (*metis* α*-wpd* α*-is-comp* α*-iwd* α*-nnf allones-implies-is-composition-MLTL convert-nnf-ext-convert-nnf-ext x1-semantics x1y1*)
have β*-semantics*: *semantics-mltl-ext* π β **using** *LP-mltl-aux-language-union-converse*
by (*metis* β*-wpd* β*-is-comp* β*-iwd* β*-nnf allones-implies-is-composition-MLTL convert-nnf-ext-convert-nnf-ext x1y1 y1-semantics*)

{
assume *List.member* (*And-mltl-list ?Dx ?Dy*) $\psi 2$
then obtain *x2 y2* **where** ψ*2-is*: $\psi 2 =$ *And-mltl-ext x2 y2*
and *x2y2*: (*x2* $\in$ *set ?Dx* $\wedge$ *y2* $\in$ *set ?Dy*)
using *And-mltl-list-member*
by (*metis in-set-member*)
have *x2-semantics*: *semantics-mltl-ext* π *x2* **and**
y2-semantics: *semantics-mltl-ext* π *y2*
using *Suc*(*9*) **unfolding** *semantics-mltl-ext-def* ψ*2-is* **by** *simp-all*
have *xs-ys-eo*: *x1* $\neq$ *x2* $\vee$ *y1* $\neq$ *y2*
using *x1y1 x2y2 Suc*(*7*) ψ*1-is* ψ*2-is* **by** *blast*
have *xs-neq*: *x1* $\neq$ *x2* $\Longrightarrow$ *False*
using *Suc*(*1*)[*OF* α*-iwd* α*-nnf* α*-is-comp* α*-wpd* α*-conv-same*, *of x1 x2*]
using *x1y1 x2y2 x1-semantics x2-semantics* **by** *blast*
have *ys-neq*: *y1* $\neq$ *y2* $\Longrightarrow$ *False*
using *Suc*(*1*)[*OF* β*-iwd* β*-nnf* β*-is-comp* β*-wpd* β*-conv-same*, *of y1 y2*]
using *x1y1 x2y2 y1-semantics y2-semantics* **by** *blast*
have *?thesis*
using *xs-neq ys-neq xs-ys-eo* **by** *blast*
} moreover {
assume *List.member* (*And-mltl-list* [Not_c α] *?Dy*) $\psi 2$
then obtain *x2 y2* **where** ψ*2-is*: $\psi 2 =$ *And-mltl-ext x2 y2*
and *x2y2*: (*x2* $= Not_c\ \alpha \wedge$ *y2* $\in$ *set ?Dy*)
using *And-mltl-list-member*
by (*metis member-def member-rec*(*1*) *member-rec*(*2*))
have *x2-is*: *x2* $= Not_c\ \alpha$
using *x2y2* **by** *auto*
have *x2-semantics*: *semantics-mltl-ext* π *x2* **and**
y2-semantics: *semantics-mltl-ext* π *y2*
using *Suc*(*9*) **unfolding** *semantics-mltl-ext-def* ψ*2-is* **by** *simp-all*
have *xs-ys-eo*: *x1* $\neq$ *x2* $\vee$ *y1* $\neq$ *y2*
using *x1y1 x2y2 Suc*(*7*) ψ*1-is* ψ*2-is* **by** *blast*
have *?thesis*
using α*-semantics x2-semantics* **unfolding** *x2-is semantics-mltl-ext-def*

by *simp*
} moreover {
assume *List.member* (*And-mltl-list ?Dx* [Not_c β]) $\psi 2$
then obtain *x2 y2* **where** $\psi 2$*-is*: $\psi 2$ = *And-mltl-ext x2 y2*
and *x2y2*: (*x2* ∈ *set ?Dx* ∧ *y2* = Not_c β)
using *And-mltl-list-member*
by (*metis member-def member-rec*(*1*) *member-rec*(*2*))
have *y2-is*: *y2* = Not_c β
using *x2y2* **by** *auto*
have *x2-semantics*: *semantics-mltl-ext* π *x2* **and**
y2-semantics: *semantics-mltl-ext* π *y2*
using *Suc*(*9*) **unfolding** *semantics-mltl-ext-def* $\psi 2$*-is* **by** *simp-all*
have *xs-ys-eo*: *x1* ≠ *x2* ∨ *y1* ≠ *y2*
using *x1y1 x2y2 Suc*(*7*) $\psi 1$*-is* $\psi 2$*-is* **by** *blast*
have *?thesis*
using β*-semantics y2-semantics* **unfolding** *y2-is semantics-mltl-ext-def*
by *simp*
}
ultimately have *?thesis*
using $\psi 2$*-eo* **by** *argo*
} moreover {
assume *List.member* (*And-mltl-list* [Not_c α] *?Dy*) $\psi 1$
then obtain *x1 y1* **where** $\psi 1$*-is*: $\psi 1$ = *And-mltl-ext x1 y1*
and *x1y1*: (*x1* = Not_c α ∧ *y1* ∈ *set ?Dy*)
using *And-mltl-list-member*
by (*metis member-def member-rec*(*1*) *member-rec*(*2*))
have *x1-semantics*: *semantics-mltl-ext* π *x1* **and**
y1-semantics: *semantics-mltl-ext* π *y1*
using *Suc*(*8*) **unfolding** *semantics-mltl-ext-def* $\psi 1$*-is* **by** *simp-all*
have *x1-is*: *x1* = Not_c α
using *x1y1* **by** *auto*
have *not-α-semantics*: ¬*semantics-mltl-ext* π α
using *x1y1 x1-semantics* **unfolding** *semantics-mltl-ext-def* **by** *auto*
have β*-semantics*: *semantics-mltl-ext* π β **using** *LP-mltl-aux-language-union-converse*
by (*metis* β*-wpd* β*-is-comp* β*-iwd* β*-nnf allones-implies-is-composition-MLTL*
convert-nnf-ext-convert-nnf-ext x1y1 y1-semantics)

{
assume *List.member* (*And-mltl-list ?Dx ?Dy*) $\psi 2$
then obtain *x2 y2* **where** $\psi 2$*-is*: $\psi 2$ = *And-mltl-ext x2 y2*
and *x2y2*: (*x2* ∈ *set ?Dx* ∧ *y2* ∈ *set ?Dy*)
using *And-mltl-list-member*
by (*metis in-set-member*)
have *x1-semantics*: *semantics-mltl-ext* π *x2*
using *Suc*(*9*) **unfolding** $\psi 2$*-is semantics-mltl-ext-def to-mltl.simps* **by**
simp
have *semantics-mltl-ext* π α
using *LP-mltl-aux-language-union-converse*
by (*metis* α*-wpd* α*-is-comp* α*-iwd* α*-nnf allones-implies-is-composition-MLTL*

convert-nnf-ext-convert-nnf-ext x1-semantics x2y2)
then have *?thesis* **using** *not-α-semantics* **by** *blast*
} moreover {
assume *List.member* (*And-mltl-list* [Not_c α] *?Dy*) $\psi 2$
then obtain *x2 y2* **where** $\psi 2$*-is*: $\psi 2$ = *And-mltl-ext x2 y2*
and *x2y2*: (*x2* = Not_c α ∧ *y2* ∈ *set ?Dy*)
using *And-mltl-list-member*
by (*metis member-def member-rec*(*1*) *member-rec*(*2*))

have *y2-semantics*: *semantics-mltl-ext* π *y2*
using *Suc*(*9*) **unfolding** $\psi 2$*-is semantics-mltl-ext-def to-mltl.simps* **by** *simp*
have *ys-neq*: *y1* ≠ *y2*
using *x1y1 x2y2 Suc*(*7*) $\psi 1$*-is* $\psi 2$*-is* **by** *blast*
then have *?thesis*
using *Suc*(*1*)
using β*-wpd* β*-conv-same* β*-is-comp* β*-iwd* β*-nnf x1y1 x2y2 y1-semantics y2-semantics* **by** *blast*
} moreover {
assume *List.member* (*And-mltl-list ?Dx* [Not_c β]) $\psi 2$
then obtain *x2 y2* **where** $\psi 2$*-is*: $\psi 2$ = *And-mltl-ext x2 y2*
and *x2y2*: (*x2* ∈ *set ?Dx* ∧ *y2* = Not_c β)
using *And-mltl-list-member*
by (*metis member-def member-rec*(*1*) *member-rec*(*2*))
have *x2-semantics*: *semantics-mltl-ext* π *x2*
using *Suc*(*9*) **unfolding** $\psi 2$*-is semantics-mltl-ext-def to-mltl.simps* **by** *simp*
have *?thesis*
by (*metis LP-mltl-aux-language-union-converse* α*-wpd* α*-is-comp* α*-iwd* α*-nnf allones-implies-is-composition-MLTL convert-nnf-ext-convert-nnf-ext not-α-semantics x2-semantics x2y2*)
}
ultimately have *?thesis*
using $\psi 2$*-eo* **by** *argo*
} moreover {
assume *List.member* (*And-mltl-list ?Dx* [Not_c β]) $\psi 1$
then obtain *x1 y1* **where** $\psi 1$*-is*: $\psi 1$ = *And-mltl-ext x1 y1*
and *x1y1*: (*x1* ∈ *set ?Dx* ∧ *y1* = Not_c β)
using *And-mltl-list-member*
by (*metis member-def member-rec*(*1*) *member-rec*(*2*))
have *x1-semantics*: *semantics-mltl-ext* π *x1* **and**
y1-semantics: *semantics-mltl-ext* π *y1*
using *Suc*(*8*) **unfolding** *semantics-mltl-ext-def* $\psi 1$*-is* **by** *simp-all*
have *x1-is*: *y1* = Not_c β
using *x1y1* **by** *auto*
have β*-semantics*: ¬*semantics-mltl-ext* π β
using *x1y1 y1-semantics* **unfolding** *semantics-mltl-ext-def* **by** *auto*
have α*-semantics*: *semantics-mltl-ext* π α **using** *LP-mltl-aux-language-union-converse*
by (*metis* α*-wpd* α*-is-comp* α*-iwd* α*-nnf allones-implies-is-composition-MLTL*

convert-nnf-ext-convert-nnf-ext x1-semantics x1y1)

{
assume *List.member* (*And-mltl-list ?Dx ?Dy*) $\psi 2$
then obtain *x2 y2* **where** $\psi 2$*-is*: $\psi 2 = $ *And-mltl-ext x2 y2*
and *x2y2*: (*x2* $\in$ *set ?Dx* $\wedge$ *y2* $\in$ *set ?Dy*)
using *And-mltl-list-member*
by (*metis in-set-member*)
have *semantics-mltl-ext* π *y2*
using *Suc*(*9*) **unfolding** $\psi 2$*-is semantics-mltl-ext-def to-mltl.simps* **by** *auto*
then have β*-semantics*: *semantics-mltl-ext* π β
using *LP-mltl-aux-language-union-converse*
by (*metis* β*-wpd* β*-is-comp* β*-iwd* β*-nnf allones-implies-is-composition-MLTL convert-nnf-ext-convert-nnf-ext x2y2*)
then have *?thesis*
by (*simp add*: *not-*β*-semantics*)
} **moreover** {
assume *List.member* (*And-mltl-list* [Not_c α] *?Dy*) $\psi 2$
then obtain *x2 y2* **where** $\psi 2$*-is*: $\psi 2 = $ *And-mltl-ext x2 y2*
and *x2y2*: (*x2* $= Not_c\ \alpha \wedge$ *y2* $\in$ *set ?Dy*)
using *And-mltl-list-member*
by (*metis member-def member-rec*(*1*) *member-rec*(*2*))
have *semantics-mltl-ext* π *y2*
using *Suc*(*9*) **unfolding** $\psi 2$*-is semantics-mltl-ext-def to-mltl.simps* **by** *auto*
then have β*-semantics*: *semantics-mltl-ext* π β
using *LP-mltl-aux-language-union-converse*
by (*metis* β*-wpd* β*-is-comp* β*-iwd* β*-nnf allones-implies-is-composition-MLTL convert-nnf-ext-convert-nnf-ext x2y2*)
then have *?thesis*
by (*simp add*: *not-*β*-semantics*)
} **moreover** {
assume *List.member* (*And-mltl-list ?Dx* [Not_c β]) $\psi 2$
then obtain *x2 y2* **where** $\psi 2$*-is*: $\psi 2 = $ *And-mltl-ext x2 y2*
and *x2y2*: (*x2* $\in$ *set ?Dx* $\wedge$ *y2* $= Not_c\ \beta$)
using *And-mltl-list-member*
by (*metis member-def member-rec*(*1*) *member-rec*(*2*))
have *semantics-mltl-ext* π *x2*
using *Suc*(*9*) **unfolding** $\psi 2$*-is semantics-mltl-ext-def to-mltl.simps* **by** *auto*
then have *?thesis*
using *Suc.IH Suc.prems*(*6*) α*-wpd* α*-conv-same* α*-is-comp* α*-iwd* α*-nnf* $\psi 1$*-is* $\psi 2$*-is x1-semantics x1y1 x2y2* **by** *blast*
}
ultimately have *?thesis*
using $\psi 2$*-eo* **by** *argo*
}
ultimately show *?thesis*

```
      using ψ1-eo by argo
    next
      case (Future-mltl-ext a b L α)
      have a-leq-b: a ≤ b and
          α-welldef: intervals-welldef (to-mltl α)
       using Suc(2) unfolding intervals-welldef.simps Future-mltl-ext to-mltl.simps
        by simp-all
      have α-nnf: ∃φ-init. α = convert-nnf-ext φ-init
        using Suc(3) unfolding Future-mltl-ext
      by (metis convert-nnf-ext.simps(6) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(5))

      have α-convert: convert-nnf-ext α = α
        using α-nnf convert-nnf-ext-convert-nnf-ext by metis
      have α-composition-allones: is-composition-MLTL-allones α and
          L-composition-allones: is-composition-allones (b−a+1) L
        using Future-mltl-ext Suc.prems(3) by simp-all
      have α-composition: is-composition-MLTL α
         using Future-mltl-ext Suc.prems(3) allones-implies-is-composition-MLTL
by auto
      have L-composition: is-composition (b−a+1) L
         using Future-mltl-ext Suc.prems(3) allones-implies-is-composition-MLTL
is-composition-MLTL.simps(5) by blast
      have α-wpd: b + wpd-mltl (to-mltl α) ≤ length π
        using Suc(5) unfolding Future-mltl-ext to-mltl.simps wpd-mltl.simps
        by auto
      let ?D = LP-mltl-aux α k
      let ?s = interval-times a L
      have length-L: 1 ≤ length L
        using composition-length-lb[OF L-composition] a-leq-b by linarith
      have length-L-allones: length L = b−a+1
        using L-composition-allones
        by (simp add: length-is-composition-allones)
      have sfirst: ?s!0 = a
        using interval-times-first by simp
      have slast: ?s!(length L) = b+1
        using interval-times-last[OF a-leq-b L-composition] by blast
      have length-s: length ?s = length L + 1
        using interval-times-length by simp
      let ?front = set (Future-mltl-list ?D (?s ! 0) (?s ! 1 − 1) [?s ! 1 − ?s ! 0])
      let ?back = set (concat (map (λi. And-mltl-list
                        [Global-mltl-ext (?s ! 0) (?s ! i − 1) [?s!i − ?s!0] (Not_c α)]
                          (Future-mltl-list ?D (?s ! i) (?s ! (i + 1) − 1) [?s ! (i +
1) − ?s ! i]))
                  [1..<length L]))
      have D-is: D = ?front ∪ ?back
        using Suc(6) unfolding Future-mltl-ext LP-mltl-aux.simps to-mltl.simps
        using α-convert list-concat-set-union by metis
      have s1: ?s!1 = a+1
          using interval-times-allones[OF a-leq-b L-composition-allones] length-s
```

length-L
by *force*
have *dropa-wpd*: *wpd-mltl* (*to-mltl* α) $\leq$ *length* (*drop a* π)
using α*-wpd a-leq-b* **by** *simp*
{
assume $*$: ψ*1* $\in$ *?front*
obtain *x1* **where** ψ*1-is*: ψ*1* = *Future-mltl-ext a a* [*1*] *x1*
and *x1-in*: *x1* $\in$ *set ?D*
using $*$ **unfolding** *sfirst s1 Future-mltl-list.simps* **by** *auto*
have *x1-semantics*: *semantics-mltl-ext* (*drop a* π) *x1*
using *Suc*(*8*) **unfolding** ψ*1-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *auto*
have α*-semantics*: *semantics-mltl-ext* (*drop a* π) α
using *LP-mltl-aux-language-union-converse*[*OF* α*-welldef* α*-nnf* α*-composition dropa-wpd*, *of ?D k*]
using *x1-semantics x1-in* **by** *blast*
{
assume $**$: ψ*2* $\in$ *?front*
obtain *x2* **where** ψ*2-is*: ψ*2* = *Future-mltl-ext a a* [*1*] *x2*
and *x2-in*: *x2* $\in$ *set ?D*
using $**$ **unfolding** *sfirst s1 Future-mltl-list.simps* **by** *auto*
have *x2-semantics*: *semantics-mltl-ext* (*drop a* π) *x2*
using *Suc*(*9*) **unfolding** ψ*2-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *auto*
have *xs-neq*: *x1* $\neq$ *x2*
using *Suc*(*7*) **unfolding** ψ*1-is* ψ*2-is* **by** *blast*
have *?thesis* **using** *dropa-wpd*
using *Suc*(*1*)[*OF* α*-welldef* α*-nnf* α*-composition-allones*, *of drop a* π *set ?D x1 x2*]
using *xs-neq x1-in x2-in x1-semantics x2-semantics* **by** *blast*
} moreover {
assume $**$: ψ*2* $\in$ *?back*
then obtain *i* **where** ψ*2-is*: ψ*2* $\in$ *set* ((*And-mltl-list*
[*Global-mltl-ext* (*?s* ! *0*) (*?s* ! *i* $-$ *1*) [*?s*!*i* $-$ *?s*!*0*] (*Not*$_c$ α)]
(*Future-mltl-list ?D* (*?s* ! *i*) (*?s* ! (*i* + *1*) $-$ *1*) [*?s* ! (*i* + *1*) $-$ *?s* ! *i*])))
and *i-bound*: *1* $\leq$ *i* $\wedge$ *i* $<$ *length L*
by *force*
have *si*: *?s*!*i* = *a*+*i*
using *interval-times-allones*
using *L-composition-allones a-leq-b i-bound length-s* **by** *auto*
have *si1*: *?s*!(*i*+*1*) = *a*+*i*+*1*
using *interval-times-allones*
using *L-composition-allones a-leq-b i-bound length-s* **by** *auto*
obtain *x2* **where** ψ*2-is*: ψ*2* = *And-mltl-ext* (*Global-mltl-ext a* (*a*+*i*$-$*1*) [*i*] (*Not*$_c$ α))
(*Future-mltl-ext* (*a*+*i*) (*a*+*i*) [*1*] *x2*)

and *x2-in*: $x2 \in set\ ?D$
using $\psi 2\text{-}is\ si\ si1\ sfirst$ **by** *auto*
then have *?thesis* **using** *Suc*(9) **unfolding** $\psi 2\text{-}is$ *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *i-bound* α*-wpd*
by (*metis* α*-semantics wpd-geq-one drop-eq-Nil2 dropa-wpd eq-imp-le le-neq-implies-less length-0-conv less-nat-zero-code not-one-le-zero semantics-mltl-ext-def*)

}
ultimately have *?thesis*
using *Suc*(7) *D-is* **by** *blast*
} **moreover** {
assume $*$: $\psi 1 \in ?back$
then obtain *i1* **where** $\psi 1\text{-}is$: $\psi 1 \in set\ ((And\text{-}mltl\text{-}list\ [Global\text{-}mltl\text{-}ext\ (?s\ !\ 0)\ (?s\ !\ i1\ -\ 1)\ [?s!i1\ -\ ?s!0]\ (Not_c\ \alpha)]\ (Future\text{-}mltl\text{-}list\ ?D\ (?s\ !\ i1)\ (?s\ !\ (i1\ +\ 1)\ -\ 1)\ [?s\ !\ (i1\ +\ 1)\ -\ ?s\ !\ i1])))$
and *i1-bound*: $1 \leq i1 \wedge i1 < length\ L$
by *force*
have *si1*: $?s!i1 = a+i1$
using *interval-times-allones*
using *L-composition-allones a-leq-b i1-bound length-s* **by** *auto*
have *si'1*: $?s!(i1+1) = a+i1+1$
using *interval-times-allones*
using *L-composition-allones a-leq-b i1-bound length-s* **by** *auto*
obtain *x1* **where** $\psi 1\text{-}is$: $\psi 1 = And\text{-}mltl\text{-}ext\ (Global\text{-}mltl\text{-}ext\ a\ (a+i1-1)\ [?s!i1\ -\ ?s!0]\ (Not_c\ \alpha))\ (Future\text{-}mltl\text{-}ext\ (a+i1)\ (a+i1)\ [1]\ x1)$
and *x1-in*: $x1 \in set\ ?D$
using $\psi 1\text{-}is\ si1\ si'1\ sfirst$ **by** *auto*
have *not-*α*-semantics*: $\neg semantics\text{-}mltl\text{-}ext\ (drop\ a\ \pi)\ \alpha$
using *Suc*(8) **unfolding** $\psi 1\text{-}is$ *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *auto*
{
assume $**$: $\psi 2 \in ?front$
obtain *x2* **where** $\psi 2\text{-}is$: $\psi 2 = Future\text{-}mltl\text{-}ext\ a\ a\ [1]\ x2$
and *x2-in*: $x2 \in set\ ?D$
using $**$ **unfolding** *sfirst s1 Future-mltl-list.simps* **by** *auto*
have *x2-semantics*: $semantics\text{-}mltl\text{-}ext\ (drop\ a\ \pi)\ x2$
using *Suc*(9) **unfolding** $\psi 2\text{-}is$ *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *auto*
have α*-semantics*: $semantics\text{-}mltl\text{-}ext\ (drop\ a\ \pi)\ \alpha$
using *LP-mltl-aux-language-union-converse*[*OF* α*-welldef* α*-nnf* α*-composition dropa-wpd*, *of ?D k*]
using *x2-semantics x2-in* **by** *blast*
then have *?thesis* **using** *not-*α*-semantics* **by** *blast*

```
    } moreover {
      assume **: ψ2 ∈ ?back
      then obtain i2 where ψ2-is: ψ2 ∈ set ((And-mltl-list
                  [Global-mltl-ext (?s ! 0) (?s ! i2 − 1) [?s!i2 − ?s!0] (Not_c
α)]
                   (Future-mltl-list ?D (?s ! i2) (?s ! (i2 + 1) − 1) [?s ! (i2
+ 1) − ?s ! i2])))
        and i2-bound: 1 ≤ i2 ∧ i2 < length L
        by force
      have si2: ?s!i2 = a+i2
        using interval-times-allones
        using L-composition-allones a-leq-b i2-bound length-s by auto
      have si'2: ?s!(i2+1) = a+i2+1
        using interval-times-allones
        using L-composition-allones a-leq-b i2-bound length-s by auto
      obtain x2 where ψ2-is: ψ2 = And-mltl-ext (Global-mltl-ext a (a+i2−1)
[?s!i2 − ?s!0] (Not_c α))
                                   (Future-mltl-ext (a+i2) (a+i2) [1] x2)
                  and x2-in: x2 ∈ set ?D
        using ψ2-is si2 si'2 sfirst by auto
      have x1-semantics: semantics-mltl-ext (drop (a+i1) π) x1
          using Suc(8) unfolding ψ1-is semantics-mltl-ext-def to-mltl.simps
semantics-mltl.simps
        using i1-bound α-wpd by auto
      have wpd-mltl (to-mltl α) ≤ length (drop (a + i1) π)
        using i1-bound unfolding length-L-allones
        using a-leq-b α-wpd by auto
      then have α-semantics: semantics-mltl-ext (drop (a+i1) π) α
      using LP-mltl-aux-language-union-converse[OF α-welldef α-nnf α-composition,
of drop (a+i1) π ?D k]
        using x1-semantics x1-in by blast
      have x2-semantics: semantics-mltl-ext (drop (a+i2) π) x2
          using Suc(9) unfolding ψ2-is semantics-mltl-ext-def to-mltl.simps
semantics-mltl.simps
        using i2-bound α-wpd by auto
      have wpd-mltl (to-mltl α) ≤ length (drop (a + i2) π)
        using i2-bound unfolding length-L-allones
        using a-leq-b α-wpd by auto
      then have α-semantics2: semantics-mltl-ext (drop (a+i2) π) α
      using LP-mltl-aux-language-union-converse[OF α-welldef α-nnf α-composition,
of drop (a+i2) π ?D k]
        using x2-semantics x2-in by blast
      {
        assume i1-eq-i2: i1 = i2
        have wpd: wpd-mltl (to-mltl α) ≤ length (drop (a + i1) π)
          using i1-bound α-wpd a-leq-b unfolding length-L-allones by auto
        have x1 ≠ x2
          using i1-eq-i2 ψ1-is ψ2-is Suc(7) by blast
        then have ?thesis
```

```
          using Suc(1)[OF α-welldef α-nnf α-composition-allones, of drop (a+i1)
π set ?D x1 x2]
            using x1-in x1-semantics x2-in x2-semantics wpd i1-eq-i2 by blast
          } moreover {
            assume i1-le-i2: i1 < i2
            then have a ≤ a+i1 ∧ a+i1 ≤ a + i2 − 1
              by simp
            then have x1-semantics: ¬semantics-mltl-ext (drop (a+i1) π) α
                using Suc(9) unfolding ψ2-is semantics-mltl-ext-def to-mltl.simps
semantics-mltl.simps
              using i2-bound α-wpd a-leq-b by auto
            then have ?thesis using α-semantics by blast
          } moreover {
            assume i1-ge-i2: i1 > i2
            then have a ≤ a+i2 ∧ a+i2 ≤ a + i1 − 1
              by simp
            then have x2-semantics: ¬semantics-mltl-ext (drop (a+i2) π) α
                using Suc(8) unfolding ψ1-is semantics-mltl-ext-def to-mltl.simps
semantics-mltl.simps
              using i1-bound α-wpd a-leq-b by auto
            then have ?thesis using α-semantics2 by blast
          }
          ultimately have ?thesis by linarith
        }
        ultimately have ?thesis
          using Suc(7) D-is by blast
      }
      ultimately show ?thesis
        using Suc(7) D-is by blast
    next
      case (Global-mltl-ext a b L α)
      have a-leq-b: a ≤ b and
           α-welldef: intervals-welldef (to-mltl α)
       using Suc(2) unfolding intervals-welldef.simps Global-mltl-ext to-mltl.simps
         by simp-all
      have α-nnf: ∃φ-init. α = convert-nnf-ext φ-init
        using Suc(3) unfolding Global-mltl-ext
      by (metis convert-nnf-ext.simps(7) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(6))

      have α-convert: convert-nnf-ext α = α
        using α-nnf convert-nnf-ext-convert-nnf-ext by metis
      have α-composition-allones: is-composition-MLTL-allones α
        using Global-mltl-ext Suc.prems(3) by simp-all
      have α-composition: is-composition-MLTL α
       using Global-mltl-ext Suc.prems(3) allones-implies-is-composition-MLTL by
auto
      have α-wpd: b + wpd-mltl (to-mltl α) ≤ length π
        using Suc(5) unfolding Global-mltl-ext to-mltl.simps wpd-mltl.simps
        by auto
```

let *?D = LP-mltl-aux α k*
{
assume ∗: *length ?D ≤ 1*
then have *D-is*: *D* = {*Global-mltl-ext a b L α*}
using *Suc*(*6*) **unfolding** *Global-mltl-ext LP-mltl-aux.simps*
using *α-convert* **by** *auto*
then have *?thesis*
using *Suc*(*7*) **by** *blast*
} moreover {
assume ∗: *length ?D > 1*
then have *D-is*: *D* = *set* (*Global-mltl-decomp ?D a* (*b* − *a*) *L*)
using *Suc*(*6*) **unfolding** *Global-mltl-ext LP-mltl-aux.simps*
using *α-convert* **by** *auto*
obtain *X1* **where** *ψ1-is*: *ψ1* = *Ands-mltl-ext X1*
and *X1-fact*: ∀ *i*<*length X1*. ∃ *y*∈*set* (*LP-mltl-aux α k*).
X1 ! *i* = *Global-mltl-ext* (*a* + *i*) (*a* + *i*) [*1*] *y*
and *length-X1*: *length X1* = *Suc* (*b* − *a*)
using *in-Global-mltl-decomp-exact-forward*[*OF* ∗]
using *Suc*(*7*) *D-is* **by** *blast*
obtain *X2* **where** *ψ2-is*: *ψ2* = *Ands-mltl-ext X2*
and *X2-fact*: ∀ *i*<*length X2*. ∃ *y*∈*set* (*LP-mltl-aux α k*).
X2 ! *i* = *Global-mltl-ext* (*a* + *i*) (*a* + *i*) [*1*] *y*
and *length-X2*: *length X2* = *Suc* (*b* − *a*)
using *in-Global-mltl-decomp-exact-forward*[*OF* ∗]
using *Suc*(*7*) *D-is* **by** *blast*
have *X1-neq-X2*: *X1* ≠ *X2*
using *Suc*(*7*) *ψ1-is ψ2-is* **by** *blast*
then have ∃ *i* < *b*−*a*+*1*. *X1*!*i* ≠ *X2*!*i*
using *length-X1 length-X2*
by (*metis add.commute nth-equalityI plus-1-eq-Suc*)
then obtain *i* **where** *i-bound*: *i* < *b*−*a*+*1*
and *X1i-neq-X2i*: *X1*!*i* ≠ *X2*!*i* **by** *blast*
obtain *y1* **where** *X1i-is*: *X1*!*i* = *Global-mltl-ext* (*a* + *i*) (*a* + *i*) [*1*] *y1*
and *y1-in*: *y1* ∈ *set ?D*
using *X1-fact i-bound length-X1* **by** *auto*
obtain *y2* **where** *X2i-is*: *X2*!*i* = *Global-mltl-ext* (*a* + *i*) (*a* + *i*) [*1*] *y2*
and *y2-in*: *y2* ∈ *set ?D*
using *X2-fact i-bound length-X2* **by** *auto*
have *y1-neq-y2*: *y1* ≠ *y2*
using *X1i-is X2i-is X1i-neq-X2i* **by** *simp*
have *semantics-mltl-ext π* (*X1*!*i*)
using *Ands-mltl-semantics*[*of X1 π*] *Suc*(*8*) **unfolding** *ψ1-is*
by (*metis Suc-eq-plus1 i-bound le-add2 length-X1 nth-mem*)
then have *y1-semantics*: *semantics-mltl-ext* (*drop* (*a*+*i*) *π*) *y1*
unfolding *X1i-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *i-bound α-wpd a-leq-b*
by (*metis Nat.add-diff-assoc Nat.le-diff-conv2 add-leD1 wpd-geq-one diff-add-inverse diff-add-inverse2 less-eq-iff-succ-less not-add-less1 order-refl*)

```
      have semantics-mltl-ext π (X2!i)
        using Ands-mltl-semantics[of X2 π] Suc(9) unfolding ψ2-is
        by (metis Suc-eq-plus1 i-bound le-add2 length-X2 nth-mem)
      then have y2-semantics: semantics-mltl-ext (drop (a+i) π) y2
       unfolding X2i-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps
        using i-bound α-wpd a-leq-b
            by (metis Nat.add-diff-assoc Nat.le-diff-conv2 add-leD1 wpd-geq-one
diff-add-inverse diff-add-inverse2 less-eq-iff-succ-less not-add-less1 order-refl)

      have wpd: wpd-mltl (to-mltl α) ≤ length (drop (a+i) π)
        using α-wpd i-bound a-leq-b by auto
      have ?thesis
        using Suc(1)[OF α-welldef α-nnf α-composition-allones wpd, of set ?D
y1 y2]
        using y1-in y2-in y1-semantics y2-semantics y1-neq-y2 by simp
    }
    ultimately show ?thesis by linarith
  next
    case (Until-mltl-ext α a b L β)
    have a-leq-b: a ≤ b and
        α-welldef: intervals-welldef (to-mltl α) and
        β-welldef: intervals-welldef (to-mltl β)
     using Suc(2) unfolding intervals-welldef.simps Until-mltl-ext to-mltl.simps
      by simp-all
    have α-nnf: ∃φ-init. α = convert-nnf-ext φ-init
      using Suc(3) unfolding Until-mltl-ext
    by (metis convert-nnf-ext.simps(8) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(7))

    have α-convert: convert-nnf-ext α = α
      using α-nnf convert-nnf-ext-convert-nnf-ext by metis
    have β-nnf: ∃φ-init. β = convert-nnf-ext φ-init
      using Suc(3) unfolding Until-mltl-ext
    by (metis convert-nnf-ext.simps(8) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(7))

    have β-convert: convert-nnf-ext β = β
      using β-nnf convert-nnf-ext-convert-nnf-ext by metis
    have α-composition-allones: is-composition-MLTL-allones α and
        β-composition-allones: is-composition-MLTL-allones β and
        L-composition-allones: is-composition-allones (b−a+1) L
      using Until-mltl-ext Suc.prems(3) by simp-all
    have α-composition: is-composition-MLTL α
      using Until-mltl-ext Suc.prems(3) allones-implies-is-composition-MLTL by
auto
    have β-composition: is-composition-MLTL β
        using Until-mltl-ext Suc.prems(3) allones-implies-is-composition-MLTL
is-composition-MLTL.simps(5)
      by force
    have L-composition: is-composition (b−a+1) L
      using L-composition-allones allones-implies-is-composition by auto
```

have α-*wpd*: b + *wpd-mltl* (*to-mltl* α)$-1 \leq$ *length* π **and**
β-*wpd*: b + *wpd-mltl* (*to-mltl* β) $\leq$ *length* π
using *Suc*(5) **unfolding** *Until-mltl-ext to-mltl.simps wpd-mltl.simps*
by *auto*
let *?D* = *LP-mltl-aux* β *k*
let *?s* = *interval-times a L*
have *length-L*: $1 \leq$ *length L*
using *composition-length-lb*[*OF L-composition*] *a-leq-b* **by** *linarith*
have *length-L-allones*: *length L* = $b-a+1$
using *L-composition-allones*
by (*simp add*: *length-is-composition-allones*)
have *sfirst*: *?s*!*0* = *a*
using *interval-times-first* **by** *simp*
have *slast*: *?s*!(*length L*) = $b+1$
using *interval-times-last*[*OF a-leq-b L-composition*]
by *blast*
have *length-s*: *length ?s* = *length L* + *1*
using *interval-times-length* **by** *simp*
have *s1*: *?s* ! *1* = $a+1$
using *interval-times-allones*
by (*metis L-composition-allones a-leq-b length-L length-s less-eq-iff-succ-less*)
let *?front* = *set* (*Until-mltl-list* α *?D* (*?s* ! *0*) (*?s* ! *1* − *1*) [*?s* ! *1* − *?s* ! *0*])
let *?back* = *set* (*concat* (*map* (λ*i*. *And-mltl-list*
[*Global-mltl-ext*
(*?s* ! *0*) (*?s* ! *i* − *1*) [*?s*!*i* − *?s*!*0*] (*And-mltl-ext* α (Not_c
β))]
(*Until-mltl-list* α *?D* (*?s* ! *i*) (*?s* ! (*i* + *1*) − *1*)
[*?s* ! (*i* + *1*) − *?s* ! *i*])) [*1*..<*length L*]))
have *split*: *D* = *?front* ∪ *?back*
using *Suc*(6) **unfolding** *Until-mltl-ext LP-mltl-aux.simps*
using α-*convert* β-*convert list-concat-set-union*
by *metis*
{
assume $*$: ψ*1* ∈ *?front*
then obtain *x1* **where** ψ*1-is*: ψ*1* = *Until-mltl-ext* α *a a* [*1*] *x1*
and *x1-in*: *x1* ∈ *set ?D*
unfolding *sfirst s1* **by** *auto*
have *x1-semantics*: *semantics-mltl* (*drop a* π) (*to-mltl x1*)
using *Suc*(8) **unfolding** ψ*1-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *auto*
have *wpd-dropa*: *wpd-mltl* (*to-mltl* β) $\leq$ *length* (*drop a* π)
using β-*wpd a-leq-b* **by** *simp*
then have β-*semantics*: *semantics-mltl-ext* (*drop a* π) β
unfolding *semantics-mltl-ext-def*
using *LP-mltl-aux-language-union-converse*[*OF* β-*welldef* β-*nnf* β-*composition*, *of drop a* π *?D k*]
using *x1-semantics x1-in* **unfolding** *semantics-mltl-ext-def* **by** *blast*
{

assume $**$: $\psi 2 \in$ *?front*
then obtain *x2* **where** *ψ2-is*: $\psi 2 =$ *Until-mltl-ext* α *a a* [*1*] *x2*
and *x2-in*: *x2* $\in$ *set ?D*
unfolding *sfirst s1* **by** *auto*
have *x2-semantics*: *semantics-mltl* (*drop a* π) (*to-mltl x2*)
using *Suc*(*9*) **unfolding** *ψ2-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *auto*
have *x1-neq-x2*: *x1* $\neq$ *x2*
using *Suc*(*7*) *ψ1-is ψ2-is* **by** *simp*
have *?thesis*
using *Suc*(*1*)[*OF β-welldef β-nnf β-composition-allones*, *of drop a π set ?D x1 x2*]
using *x1-semantics x1-in x2-semantics x2-in x1-neq-x2*
using *semantics-mltl-ext-def wpd-dropa* **by** *blast*
} moreover {
assume $**$: *ψ2* $\in$ *?back*
then obtain *i y2* **where**
ψ2-is: *ψ2* = *And-mltl-ext* (*Global-mltl-ext* (*?s*!*0*) (*?s*!*i*−*1*) [*?s*!*i* − *?s*!*0*] (*And-mltl-ext* α (*Not*$_c$ β)))
(*Until-mltl-ext* α (*?s*!*i*) (*?s*!(*i*+*1*)−*1*) [(*?s*!(*i*+*1*)) − (*?s*!*i*)] *y2*)
and *i-bound*: *1* $\leq$ *i* $\wedge$ *i* < *length L*
and *y2-in*: *y2* $\in$ *set ?D*
by *auto*
have *p*: ¬*semantics-mltl-ext* (*drop a* π) β
using *Suc*(*9*) **unfolding** *ψ2-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *i-bound length-L-allones*
by (*metis wpd-dropa wpd-geq-one drop-all eq-imp-le le-neq-implies-less length-0-conv less-nat-zero-code not-one-le-zero sfirst*)
have *?thesis* **using** *β-semantics p*
by *metis*
}
ultimately have *?thesis* **using** *Suc*(*7*) *split* **by** *blast*
} moreover {
assume $*$: *ψ1* $\in$ *?back*
then obtain *i1 y1* **where**
ψ1-is: *ψ1* = *And-mltl-ext* (*Global-mltl-ext* (*?s*!*0*) (*?s*!*i1*−*1*) [*?s*!*i1* − *?s*!*0*] (*And-mltl-ext* α (*Not*$_c$ β)))
(*Until-mltl-ext* α (*?s*!*i1*) (*?s*!(*i1*+*1*)−*1*) [(*?s*!(*i1*+*1*)) − (*?s*!*i1*)] *y1*)
and *i1-bound*: *1* $\leq$ *i1* $\wedge$ *i1* < *length L*
and *y1-in*: *y1* $\in$ *set ?D*
by *auto*
have *si1*: *?s*!*i1* = *a* + *i1*
using *interval-times-allones*
using *L-composition-allones a-leq-b i1-bound length-s* **by** *auto*
have *si1′*: *?s*!(*i1*+*1*) = *a*+*i1*+*1*
using *interval-times-allones*
using *L-composition-allones a-leq-b i1-bound length-s* **by** *auto*

have *ψ1-is*: *ψ1 = And-mltl-ext (Global-mltl-ext a (a+i1−1) [i1] (And-mltl-ext α (Not$_c$ β)))*
(Until-mltl-ext α (a+i1) (a+i1) [1] y1)
using *si1 si1′ sfirst ψ1-is* **by** *auto*
have *y1-semantics*: *semantics-mltl-ext (drop (a+i1) π) y1*
using *Suc(8)* **unfolding** *ψ1-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *auto*
have *wpd-mltl (to-mltl β) ≤ length (drop (a + i1) π)*
using *β-wpd i1-bound length-L-allones* **by** *auto*
then have *β-semantics1*: *semantics-mltl-ext (drop (a+i1) π) β*
using *LP-mltl-aux-language-union-converse[OF β-welldef β-nnf β-composition, of drop (a+i1) π ?D k]*
using *y1-semantics y1-in* **by** *blast*
{
assume ∗∗: *ψ2 ∈ ?front*
then obtain *x2* **where** *ψ2-is*: *ψ2 = Until-mltl-ext α a a [1] x2*
and *x2-in*: *x2 ∈ set ?D*
unfolding *sfirst s1* **by** *auto*
have *x2-semantics*: *semantics-mltl (drop a π) (to-mltl x2)*
using *Suc(9)* **unfolding** *ψ2-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *auto*
have *wpd-mltl (to-mltl β) ≤ length (drop a π)*
using *β-wpd a-leq-b* **by** *auto*
then have *β-semantics2*: *semantics-mltl (drop a π) (to-mltl β)*
using *LP-mltl-aux-language-union-converse[OF β-welldef β-nnf β-composition, of drop a π ?D k]*
using *x2-semantics x2-in* **unfolding** *semantics-mltl-ext-def*
by *blast*
then have *?thesis*
using *Suc(8)* **unfolding** *ψ1-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *auto*
} moreover {
assume ∗∗: *ψ2 ∈ ?back*
then obtain *i2 y2* **where**
ψ2-is: *ψ2 = And-mltl-ext (Global-mltl-ext (?s!0) (?s!i2−1) [?s!i2 − ?s!0] (And-mltl-ext α (Not$_c$ β)))*
(Until-mltl-ext α (?s!i2) (?s!(i2+1)−1) [(?s!(i2+1)) − (?s!i2)] y2)
and *i2-bound*: *1 ≤ i2 ∧ i2 < length L*
and *y2-in*: *y2 ∈ set ?D*
by *auto*
have *si2*: *?s!i2 = a + i2*
using *interval-times-allones*
using *L-composition-allones a-leq-b i2-bound length-s* **by** *auto*
have *si2′*: *?s!(i2+1) = a+i2+1*
using *interval-times-allones*
using *L-composition-allones a-leq-b i2-bound length-s* **by** *auto*

have *ψ2-is*: *ψ2* = *And-mltl-ext* (*Global-mltl-ext a* (*a*+*i2*−*1*) [*i2*] (*And-mltl-ext* *α* (*Not*$_c$ *β*)))
(*Until-mltl-ext α* (*a*+*i2*) (*a*+*i2*) [*1*] *y2*)
using *si2 si2′ sfirst ψ2-is* **by** *auto*
have *y2-semantics*: *semantics-mltl-ext* (*drop* (*a*+*i2*) *π*) *y2*
using *Suc*(*9*) **unfolding** *ψ2-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *auto*
have *wpd-dropi2*: *wpd-mltl* (*to-mltl β*) ≤ *length* (*drop* (*a* + *i2*) *π*)
using *β-wpd i2-bound length-L-allones* **by** *auto*
then have *β-semantics2*: *semantics-mltl-ext* (*drop* (*a*+*i2*) *π*) *β*
using *LP-mltl-aux-language-union-converse*[*OF β-welldef β-nnf β-composition*, *of drop* (*a*+*i2*) *π ?D k*]
using *y2-semantics y2-in* **by** *blast*
{
assume *i1-eq-i2*: *i1* = *i2*
then have *y1-neq-y2*: *y1* ≠ *y2*
using *ψ1-is ψ2-is Suc*(*7*) **by** *blast*
then have *?thesis*
using *Suc*(*1*)[*OF β-welldef β-nnf β-composition-allones*, *of drop* (*a*+*i1*) *π set ?D y1 y2*]
using *wpd-dropi2 i1-eq-i2 y1-semantics y1-in y2-semantics y2-in*
by *blast*
} moreover {
assume *i1-le-i2*: *i1* < *i2*
then have ¬*semantics-mltl-ext* (*drop* (*a* + *i1*) *π*) *β*
using *Suc*(*9*) **unfolding** *ψ2-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *add.assoc add-le-imp-le-diff* **by** *force*
then have *?thesis*
using *β-semantics1* **by** *blast*
} moreover {
assume *i1-ge-i2*: *i1* > *i2*
then have ¬*semantics-mltl-ext* (*drop* (*a* + *i2*) *π*) *β*
using *Suc*(*8*) **unfolding** *ψ1-is semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *add.assoc add-le-imp-le-diff* **by** *force*
then have *?thesis*
using *β-semantics2* **by** *blast*
}
ultimately have *?thesis* **by** *linarith*
}
ultimately have *?thesis*
using *split Suc*(*7*) **by** *blast*
}
ultimately show *?thesis*
using *split Suc*(*7*) **by** *blast*
next
case (*Release-mltl-ext α a b L β*)

have *a-leq-b*: $a \leq b$ **and**
α-*welldef*: *intervals-welldef* (*to-mltl* α) **and**
β-*welldef*: *intervals-welldef* (*to-mltl* β)
using *Suc*(*2*) **unfolding** *intervals-welldef.simps Release-mltl-ext to-mltl.simps*
by *simp-all*
have α-*nnf*: $\exists \varphi$-*init*. α = *convert-nnf-ext* φ-*init*
using *Suc*(*3*) **unfolding** *Release-mltl-ext*
by (*metis convert-nnf-ext.simps*(*9*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*8*))

have α-*convert*: *convert-nnf-ext* $\alpha = \alpha$
using α-*nnf convert-nnf-ext-convert-nnf-ext* **by** *metis*
have β-*nnf*: $\exists \varphi$-*init*. β = *convert-nnf-ext* φ-*init*
using *Suc*(*3*) **unfolding** *Release-mltl-ext*
by (*metis convert-nnf-ext.simps*(*9*) *convert-nnf-ext-convert-nnf-ext mltl-ext.inject*(*8*))

have β-*convert*: *convert-nnf-ext* $\beta = \beta$
using β-*nnf convert-nnf-ext-convert-nnf-ext* **by** *metis*
have α-*composition-allones*: *is-composition-MLTL-allones* α **and**
β-*composition-allones*: *is-composition-MLTL-allones* β **and**
L-composition-allones: *is-composition-allones* ($b-a+1$) *L*
using *Release-mltl-ext Suc.prems*(*3*) **by** *simp-all*
have α-*composition*: *is-composition-MLTL* α
using *Release-mltl-ext Suc.prems*(*3*) *allones-implies-is-composition-MLTL*
by *auto*
have β-*composition*: *is-composition-MLTL* β
using *Release-mltl-ext Suc.prems*(*3*) *allones-implies-is-composition-MLTL*
is-composition-MLTL.simps(*5*)
by *force*
have *L-composition*: *is-composition* ($b-a+1$) *L*
using *L-composition-allones allones-implies-is-composition* **by** *auto*
have α-*wpd*: b + *wpd-mltl* (*to-mltl* α) $\leq$ *length* π **and**
β-*wpd*: b + *wpd-mltl* (*to-mltl* β) $\leq$ *length* π
using *Suc*(*5*) **unfolding** *Release-mltl-ext to-mltl.simps wpd-mltl.simps*
by *auto*
let *?D* = *LP-mltl-aux* α *k*
let *?s* = *interval-times a L*
have *length-L*: $1 \leq$ *length L*
using *composition-length-lb*[*OF L-composition*] *a-leq-b* **by** *linarith*
have *length-L-allones*: *length L* = $b-a+1$
using *L-composition-allones*
by (*simp add*: *length-is-composition-allones*)
have *sfirst*: *?s*!*0* = *a*
using *interval-times-first* **by** *simp*
have *slast*: *?s*!(*length L*) = $b+1$
using *interval-times-last*[*OF a-leq-b L-composition*]
by *blast*
have *length-s*: *length ?s* = *length L* + *1*
using *interval-times-length* **by** *simp*
have *length-L*: *length L* = $b-a+1$

using *length-is-composition-allones*[*OF L-composition-allones*]
by *blast*
have *s1*: *?s ! 1 = a+1*
using *interval-times-allones*
using *L-composition L-composition-allones a-leq-b add-gr-0 composition-length-lb length-s* **by** *auto*
have *length-π-ge-b*: *length π > b*
using *α-wpd wpd-geq-one*
by (*metis One-nat-def Suc-n-not-le-n add-diff-cancel-left' add-leD1 diff-is-0-eq' le-neq-implies-less*)
let *?front = set* [*Global-mltl-ext a b L* (*And-mltl-ext* (*Not*$_c$ *α*) *β*)]
let *?middle = set* (*Mighty-Release-mltl-list ?D β* (*?s ! 0*) (*?s ! 1 − 1*)
[*?s ! 1 − ?s ! 0*])
let *?back = set* (*concat* (*map* (*λi. And-mltl-list*
[*Global-mltl-ext*
(*?s ! 0*) (*?s ! i − 1*) [*?s!i − ?s!0*] (*And-mltl-ext* (*Not*$_c$ *α*) *β*)]
(*Mighty-Release-mltl-list ?D β* (*?s ! i*)
(*?s ! (i + 1) − 1*) [*?s ! (i + 1) − ?s ! i*]))
[*1..<length L*]))
have *D-is*: *D = ?front ∪ ?middle ∪ ?back*
using *Suc*(*6*) **unfolding** *Release-mltl-ext LP-mltl-aux.simps*
using *α-convert list-concat-set-union*
by (*metis append-assoc*)
{
assume ∗: *ψ1 ∈ ?front*
then have *ψ1*: *ψ1 = Global-mltl-ext a b L* (*And-mltl-ext* (*Not*$_c$ *α*) *β*)
by *auto*
{
assume ∗∗: *ψ2 ∈ ?front*
have *?thesis* **using** ∗ ∗∗ *Suc*(*7*) **by** *auto*
} moreover {
assume ∗∗: *ψ2 ∈ ?middle*
then obtain *x* **where** *ψ2*: *ψ2 = Mighty-Release-mltl-ext x β*
a (*?s ! 1 − 1*) [*?s ! 1 − a*]
and *x-in*: *x ∈ set ?D*
using *sfirst* **by** *auto*
have *ψ2*: *ψ2 = Mighty-Release-mltl-ext x β a a* [*1*]
using *s1 ψ2* **by** *simp*
have *x-semantics*: *semantics-mltl* (*drop a π*) (*to-mltl x*)
using *Suc*(*9*) **unfolding** *ψ1 ψ2 semantics-mltl-ext-def to-mltl.simps Mighty-Release-mltl-ext.simps semantics-mltl.simps*
by *force*
have *wpd-mltl* (*to-mltl α*) ≤ *length* (*drop a π*)
using *α-wpd a-leq-b* **by** *auto*
then have *semantics-mltl* (*drop a π*) (*to-mltl α*)
using *LP-mltl-aux-language-union-converse*[*OF α-welldef α-nnf α-composition*, *of drop a π ?D k*]
using *x-semantics x-in* **unfolding** *semantics-mltl-ext-def* **by** *blast*

then have *?thesis*
using *Suc*(*8*) **unfolding** *ψ1 semantics-mltl-ext-def to-mltl.simps Mighty-Release-mltl-ext.simps semantics-mltl.simps*
using *length-π-ge-b* **by** *auto*
} moreover {
assume ∗∗: *ψ2* ∈ *?back*
then obtain *i2* **where** *ψ2-in*: *ψ2* ∈ *set* (*And-mltl-list*
[*Global-mltl-ext*
(*interval-times a L* ! *0*)
(*interval-times a L* ! *i2* − *1*) [*?s*!*i2* − *?s*!*0*] (*And-mltl-ext* (*Not*$_c$ *α*) *β*)]
(*Mighty-Release-mltl-list* (*LP-mltl-aux α k*) *β*
(*interval-times a L* ! *i2*)
(*interval-times a L* ! (*i2* + *1*) − *1*)
[*interval-times a L* ! (*i2* + *1*) −
interval-times a L ! *i2*]))
and *i2-bound*: *1* ≤ *i2* ∧ *i2* < *length L*
by *force*
have *si2*: *?s*!*i2* = *a*+*i2*
using *interval-times-allones*[*OF a-leq-b L-composition-allones*, *of i2*]
using *i2-bound length-L length-s* **by** *auto*
have *si2′*: *?s*!(*i2*+*1*) = *a*+*i2*+*1*
using *interval-times-allones*[*OF a-leq-b L-composition-allones*, *of i2*+*1*]
using *i2-bound length-L length-s* **by** *auto*
obtain *x2* **where** *ψ2*: *ψ2* = *And-mltl-ext*
(*Global-mltl-ext a* (*a* + *i2* − *1*) [*i2*] (*And-mltl-ext* (*Not*$_c$ *α*) *β*))
(*Mighty-Release-mltl-ext x2 β* (*a*+ *i2*) (*a*+ *i2*) [*1*])
and *x2-in*: *x2* ∈ *set ?D*
using *ψ2-in sfirst si2 si2′* **by** *auto*
have *x2-semantics*: *semantics-mltl* (*drop* (*a* + *i2*) *π*) (*to-mltl x2*)
using *Suc*(*9*) **unfolding** *ψ2 semantics-mltl-ext-def to-mltl.simps Mighty-Release-mltl-ext.simps semantics-mltl.simps*
by *force*
have *wpd-mltl* (*to-mltl α*) ≤ *length* (*drop* (*a* + *i2*) *π*)
using *α-wpd a-leq-b i2-bound length-L* **by** *auto*
then have *semantics-mltl* (*drop* (*a* + *i2*) *π*) (*to-mltl α*)
using *LP-mltl-aux-language-union-converse*[*OF α-welldef α-nnf α-composition*, *of drop* (*a* + *i2*) *π* *?D k*]
using *x2-semantics x2-in* **unfolding** *semantics-mltl-ext-def* **by** *blast*
then have *?thesis*
using *Suc*(*8*) **unfolding** *ψ1 semantics-mltl-ext-def to-mltl.simps Mighty-Release-mltl-ext.simps semantics-mltl.simps*
using *length-π-ge-b i2-bound length-L* **by** *auto*
}
ultimately have *?thesis* **using** *Suc*(*7*) *D-is* **by** *blast*
} moreover {
assume ∗: *ψ1* ∈ *?middle*
then obtain *x1* **where** *ψ1*: *ψ1* = *Mighty-Release-mltl-ext x1 β*
a (*?s* ! *1* − *1*) [*?s* ! *1* − *a*]

and *x1-in*: $x1 \in set\ ?D$
using *sfirst* **by** *auto*
have $\psi 1$: $\psi 1 = Mighty\text{-}Release\text{-}mltl\text{-}ext\ x1\ \beta\ a\ a\ [1]$
using *s1* $\psi 1$ **by** *simp*
have *x1-semantics*: $semantics\text{-}mltl\ (drop\ a\ \pi)\ (to\text{-}mltl\ x1)$
using *Suc*(8) **unfolding** $\psi 1$ *semantics-mltl-ext-def to-mltl.simps Mighty-Release-mltl-ext.simps semantics-mltl.simps*
by *force*
have $wpd\text{-}mltl\ (to\text{-}mltl\ \alpha) \leq length\ (drop\ a\ \pi)$
using α-*wpd a-leq-b* **by** *auto*
then have α-*semantics*: $semantics\text{-}mltl\ (drop\ a\ \pi)\ (to\text{-}mltl\ \alpha)$
using *LP-mltl-aux-language-union-converse*[*OF* α-*welldef* α-*nnf* α-*composition*, *of drop a* π *?D k*]
using *x1-semantics x1-in* **unfolding** *semantics-mltl-ext-def* **by** *blast*
{
assume $**$: $\psi 2 \in ?front$
then have $\psi 2$: $\psi 2 = Global\text{-}mltl\text{-}ext\ a\ b\ L\ (And\text{-}mltl\text{-}ext\ (Not_c\ \alpha)\ \beta)$
by *auto*
have *?thesis*
using α-*semantics* **using** *Suc*(9) **unfolding** $\psi 2$ *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *a-leq-b length-*π*-ge-b* **by** *simp*
} moreover {
assume $**$: $\psi 2 \in ?middle$
then obtain *x2* **where** $\psi 2$: $\psi 2 = Mighty\text{-}Release\text{-}mltl\text{-}ext\ x2\ \beta\ a\ (?s\ !\ 1\ -\ 1)\ [?s\ !\ 1\ -\ a]$
and *x2-in*: $x2 \in set\ ?D$
using *sfirst* **by** *auto*
have $\psi 2$: $\psi 2 = Mighty\text{-}Release\text{-}mltl\text{-}ext\ x2\ \beta\ a\ a\ [1]$
using *s1* $\psi 2$ **by** *simp*
have *x2-semantics*: $semantics\text{-}mltl\ (drop\ a\ \pi)\ (to\text{-}mltl\ x2)$
using *Suc*(9) **unfolding** $\psi 2$ *semantics-mltl-ext-def to-mltl.simps Mighty-Release-mltl-ext.simps semantics-mltl.simps*
by *force*
have *x1-neq-x2*: $x1 \neq x2$
using *Suc*(7) $\psi 1$ $\psi 2$ **by** *blast*
have $wpd\text{-}mltl\ (to\text{-}mltl\ \alpha) \leq length\ (drop\ a\ \pi)$
using α-*wpd a-leq-b* **by** *simp*
then have *?thesis*
using *Suc*(1)[*OF* α-*welldef* α-*nnf* α-*composition-allones*, *of drop a* π *set ?D x1 x2*]
using *x1-neq-x2 x1-semantics x2-semantics x1-in x2-in*
unfolding *semantics-mltl-ext-def* **by** *blast*
} moreover {
assume $**$: $\psi 2 \in ?back$
then obtain *i2* **where** $\psi 2\text{-}in$: $\psi 2 \in set\ (And\text{-}mltl\text{-}list$
$[Global\text{-}mltl\text{-}ext$
$(interval\text{-}times\ a\ L\ !\ 0)$
$(interval\text{-}times\ a\ L\ !\ i2\ -\ 1)\ [?s!i2\ -\ ?s!0]\ (And\text{-}mltl\text{-}ext$

$(Not_c\ \alpha)\ \beta)]$
(*Mighty-Release-mltl-list* (*LP-mltl-aux* α *k*) β
(*interval-times a L* ! *i2*)
(*interval-times a L* ! (*i2* + *1*) − *1*)
[*interval-times a L* ! (*i2* + *1*) −
interval-times a L ! *i2*]))
and *i2-bound*: *1* ≤ *i2* ∧ *i2* < *length L*
by *force*
have *si2*: *?s*!*i2* = *a*+*i2*
using *interval-times-allones*[*OF a-leq-b L-composition-allones*, *of i2*]
using *i2-bound length-L length-s* **by** *auto*
have *si2'*: *?s*!(*i2*+*1*) = *a*+*i2*+*1*
using *interval-times-allones*[*OF a-leq-b L-composition-allones*, *of i2*+*1*]
using *i2-bound length-L length-s* **by** *auto*
obtain *x2* **where** $\psi 2$: $\psi 2$ = *And-mltl-ext*
(*Global-mltl-ext a* (*a* + *i2* − *1*) [*i2*] (*And-mltl-ext* (*Not*$_c$ α) β))
(*Mighty-Release-mltl-ext x2* β (*a*+ *i2*) (*a*+ *i2*) [*1*])
and *x2-in*: *x2* ∈ *set ?D*
using $\psi 2$*-in sfirst si2 si2'* **by** *auto*
have *x2-semantics*: *semantics-mltl* (*drop* (*a* + *i2*) π) (*to-mltl x2*)
using *Suc*(*9*) **unfolding** $\psi 2$ *semantics-mltl-ext-def to-mltl.simps Mighty-Release-mltl-ext.simps semantics-mltl.simps*
by *force*
have *wpd-mltl* (*to-mltl* α) ≤ *length* (*drop* (*a* + *i2*) π)
using α*-wpd a-leq-b i2-bound length-L* **by** *auto*
then have *semantics-mltl* (*drop* (*a* + *i2*) π) (*to-mltl* α)
using *LP-mltl-aux-language-union-converse*[*OF* α*-welldef* α*-nnf* α*-composition*, *of drop* (*a* + *i2*) π *?D k*]
using *x2-semantics x2-in* **unfolding** *semantics-mltl-ext-def* **by** *blast*
have *?thesis* **using** α*-semantics*
using *Suc*(*9*) **unfolding** $\psi 2$ *Mighty-Release-mltl-ext.simps semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *auto*
}
ultimately have *?thesis* **using** *Suc*(*7*) *D-is* **by** *blast*
} moreover {
assume ∗: $\psi 1$ ∈ *?back*
then obtain *i1* **where** $\psi 1$*-in*: $\psi 1$ ∈ *set* (*And-mltl-list*
[*Global-mltl-ext*
(*interval-times a L* ! *0*)
(*interval-times a L* ! *i1* − *1*) [*?s*!*i1* − *?s*!*0*] (*And-mltl-ext* $(Not_c\ \alpha)\ \beta)]$
(*Mighty-Release-mltl-list* (*LP-mltl-aux* α *k*) β
(*interval-times a L* ! *i1*)
(*interval-times a L* ! (*i1* + *1*) − *1*)
[*interval-times a L* ! (*i1* + *1*) −
interval-times a L ! *i1*]))
and *i1-bound*: *1* ≤ *i1* ∧ *i1* < *length L*
by *force*

have *si1*: *?s!i1 = a+i1*
using *interval-times-allones*[*OF a-leq-b L-composition-allones*, *of i1*]
using *i1-bound length-L length-s* **by** *auto*
have *si1'*: *?s!(i1+1) = a+i1+1*
using *interval-times-allones*[*OF a-leq-b L-composition-allones*, *of i1+1*]
using *i1-bound length-L length-s* **by** *auto*
obtain *x1* **where** *ψ1*: *ψ1 = And-mltl-ext*
(*Global-mltl-ext a* (*a + i1 − 1*) [*i1*] (*And-mltl-ext* (*Not$_c$ α*) *β*))
(*Mighty-Release-mltl-ext x1 β* (*a+ i1*) (*a+ i1*) [*1*])
and *x1-in*: *x1 ∈ set ?D*
using *ψ1-in sfirst si1 si1'* **by** *auto*
have *x1-semantics*: *semantics-mltl* (*drop* (*a + i1*) *π*) (*to-mltl x1*)
using *Suc*(*8*) **unfolding** *ψ1 semantics-mltl-ext-def to-mltl.simps Mighty-Release-mltl-ext.simps semantics-mltl.simps*
by *force*
have *complen1*: *wpd-mltl* (*to-mltl α*) ≤ *length* (*drop* (*a + i1*) *π*)
using *α-wpd a-leq-b i1-bound length-L* **by** *auto*
then have *α-semantics1*: *semantics-mltl* (*drop* (*a + i1*) *π*) (*to-mltl α*)
using *LP-mltl-aux-language-union-converse*[*OF α-welldef α-nnf α-composition*, *of drop* (*a + i1*) *π ?D k*]
using *x1-semantics x1-in* **unfolding** *semantics-mltl-ext-def* **by** *blast*
{
assume ∗: *ψ2 ∈ ?front*
then have *ψ2*: *ψ2 = Global-mltl-ext a b L* (*And-mltl-ext* (*Not$_c$ α*) *β*)
by *auto*
have *?thesis*
using *Suc*(*9*) **unfolding** *ψ2 semantics-mltl-ext-def to-mltl.simps Mighty-Release-mltl-ext.simps semantics-mltl.simps*
using *length-π-ge-b i1-bound length-L*
by (*smt* (*verit*, *best*) ‹*semantics-mltl* (*drop* (*a + i1*) *π*) (*to-mltl α*)› *diff-add-inverse diff-le-mono le-antisym le-trans less-eq-iff-succ-less less-irrefl-nat less-or-eq-imp-le nat-le-iff-add nat-le-linear*)
} moreover {
assume ∗: *ψ2 ∈ ?middle*
then obtain *x2* **where** *ψ2*: *ψ2 = Mighty-Release-mltl-ext x2 β*
a (*?s ! 1 − 1*) [*?s ! 1 − a*]
and *x2-in*: *x2 ∈ set ?D*
using *sfirst* **by** *auto*
have *ψ2*: *ψ2 = Mighty-Release-mltl-ext x2 β a a* [*1*]
using *s1 ψ2* **by** *simp*
have *x2-semantics*: *semantics-mltl* (*drop a π*) (*to-mltl x2*)
using *Suc*(*9*) **unfolding** *ψ2 semantics-mltl-ext-def to-mltl.simps Mighty-Release-mltl-ext.simps semantics-mltl.simps*
by *force*
have *wpd-mltl* (*to-mltl α*) ≤ *length* (*drop a π*)
using *α-wpd a-leq-b* **by** *auto*
then have *α-semantics*: *semantics-mltl* (*drop a π*) (*to-mltl α*)
using *LP-mltl-aux-language-union-converse*[*OF α-welldef α-nnf α-composition*, *of drop a π ?D k*]

using *x2-semantics x2-in* **unfolding** *semantics-mltl-ext-def* **by** *blast*
have *?thesis*
using *Suc(8)* **unfolding** *ψ1 Mighty-Release-mltl-ext.simps semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *α-semantics* **by** *auto*
} moreover {
assume ∗: *ψ2 ∈ ?back*
then obtain *i2* **where** *ψ2-in*: *ψ2 ∈ set* (*And-mltl-list*
[*Global-mltl-ext*
(*interval-times a L ! 0*)
(*interval-times a L ! i2 − 1*) [*?s!i2 − ?s!0*] (*And-mltl-ext* (*Not*$_c$ *α*) *β*)]
(*Mighty-Release-mltl-list* (*LP-mltl-aux α k*) *β*
(*interval-times a L ! i2*)
(*interval-times a L ! (i2 + 1) − 1*)
[*interval-times a L ! (i2 + 1) −*
interval-times a L ! i2]))
and *i2-bound*: *1 ≤ i2 ∧ i2 < length L*
by *force*
have *si2*: *?s!i2 = a+i2*
using *interval-times-allones*[*OF a-leq-b L-composition-allones*, *of i2*]
using *i2-bound length-L length-s* **by** *auto*
have *si2′*: *?s!(i2+1) = a+i2+1*
using *interval-times-allones*[*OF a-leq-b L-composition-allones*, *of i2+1*]
using *i2-bound length-L length-s* **by** *auto*
obtain *x2* **where** *ψ2*: *ψ2 = And-mltl-ext*
(*Global-mltl-ext a* (*a + i2 − 1*) [*i2*] (*And-mltl-ext* (*Not*$_c$ *α*) *β*))
(*Mighty-Release-mltl-ext x2 β* (*a+ i2*) (*a+ i2*) [*1*])
and *x2-in*: *x2 ∈ set ?D*
using *ψ2-in sfirst si2 si2′* **by** *auto*
have *x2-semantics*: *semantics-mltl* (*drop* (*a + i2*) *π*) (*to-mltl x2*)
using *Suc(9)* **unfolding** *ψ2 semantics-mltl-ext-def to-mltl.simps Mighty-Release-mltl-ext.simps semantics-mltl.simps*
by *force*
have *complen2*: *wpd-mltl* (*to-mltl α*) ≤ *length* (*drop* (*a + i2*) *π*)
using *α-wpd a-leq-b i2-bound length-L* **by** *auto*
then have *α-semantics2*: *semantics-mltl* (*drop* (*a + i2*) *π*) (*to-mltl α*)
using *LP-mltl-aux-language-union-converse*[*OF α-welldef α-nnf α-composition*, *of drop* (*a + i2*) *π ?D k*]
using *x2-semantics x2-in* **unfolding** *semantics-mltl-ext-def* **by** *blast*
{
assume *eq*: *i1 = i2*
then have *x1-neq-x2*: *x1 ≠ x2*
using *Suc(7) ψ1 ψ2* **by** *blast*
have *?thesis* **using** *eq*
using *Suc(1)*[*OF α-welldef α-nnf α-composition-allones complen1*, *of set ?D x1 x2*]
using *x1-in x2-in x1-semantics x2-semantics x1-neq-x2* **unfolding** *semantics-mltl-ext-def*

```
          by blast
        } moreover {
          assume le: i1 < i2
          then have ¬semantics-mltl (drop (a + i1) π) (to-mltl α)
          using Suc(9) unfolding ψ2 semantics-mltl-ext-def semantics-mltl.simps
to-mltl.simps
            using length-π-ge-b a-leq-b by simp
          then have ?thesis
            using α-semantics1 by blast
        } moreover {
          assume ge: i1 > i2
          then have ¬semantics-mltl (drop (a + i2) π) (to-mltl α)
          using Suc(8) unfolding ψ1 semantics-mltl-ext-def semantics-mltl.simps
to-mltl.simps
            using length-π-ge-b a-leq-b by simp
          then have ?thesis
            using α-semantics2 by blast
        }
        ultimately have ?thesis by linarith
      }
      ultimately have ?thesis using Suc(7) D-is by blast
    }
    ultimately show ?thesis using Suc(7) D-is by blast
  qed
 qed

lemma LP-mltl-language-disjoint-aux:
 fixes φ::'a mltl-ext and ψ1 ψ2::'a mltl-ext and k::nat
 assumes intervals-welldef: intervals-welldef (to-mltl φ)
 assumes is-nnf: ∃φ-init. φ = convert-nnf-ext φ-init
 assumes composition: is-composition-MLTL-allones φ
 assumes D-decomp: D = set (LP-mltl-aux φ k)
 assumes diff-formulas: (ψ1 ∈ D) ∧ (ψ2 ∈ D) ∧ ψ1 ≠ ψ2
 assumes r-wpd: r ≥ wpd-mltl (to-mltl φ)
 shows (language-mltl-r (to-mltl ψ1) r)
     ∩ (language-mltl-r (to-mltl ψ2) r) = {}
proof−
 {
   assume contra: (language-mltl-r (to-mltl ψ1) r)
     ∩ (language-mltl-r (to-mltl ψ2) r) ≠ {}
   then have ∃π. π ∈ (language-mltl-r (to-mltl ψ1) r) ∧
                π ∈ (language-mltl-r (to-mltl ψ2) r)
     by auto
   then obtain π where in1: π ∈ (language-mltl-r (to-mltl ψ1) r)
           and in2: π ∈ (language-mltl-r (to-mltl ψ2) r)
     by blast
   have sem1: semantics-mltl-ext π ψ1 and
        sem2: semantics-mltl-ext π ψ2 and
        len: length π ≥ wpd-mltl (to-mltl φ)
```

using *in1 in2 assms*(*6*)
unfolding *language-mltl-r-def semantics-mltl-ext-def*
by *simp-all*
have *False*
using *LP-mltl-language-disjoint-aux-helper*[*OF assms*(*1*−*3*) *len assms*(*4*, *5*) *sem1 sem2*]
by *simp*
}
then show *?thesis* **by** *blast*
qed

theorem *LP-mltl-language-disjoint*:
fixes φ::*'a mltl-ext* **and** $\psi 1$ $\psi 2$::*'a mltl* **and** *k*::*nat*
assumes *intervals-welldef*: *intervals-welldef* (*to-mltl* φ)
assumes *composition*: *is-composition-MLTL-allones* φ
assumes *D-decomp*: *D* = *set* (*LP-mltl* φ *k*)
assumes *diff-formulas*: ($\psi 1 \in D$) ∧ ($\psi 2 \in D$) ∧ $\psi 1 \neq \psi 2$
assumes *r-wpd*: *r* ≥ *wpd-mltl* (*to-mltl* φ)
shows (*language-mltl-r* $\psi 1$ *r*) ∩ (*language-mltl-r* $\psi 2$ *r*) = {}
proof−
let *?D* = *LP-mltl-aux* (*convert-nnf-ext* φ) *k*
let *?*φ = *convert-nnf-ext* φ
have *cond1*: *intervals-welldef* (*to-mltl* (*convert-nnf-ext* φ))
using *intervals-welldef*
by (*metis convert-nnf-ext-to-mltl-commute nnf-intervals-welldef*)
have *cond2*: ∃φ*-init*. *convert-nnf-ext* φ = *convert-nnf-ext* φ*-init*
by *blast*
have *cond3*: *is-composition-MLTL-allones* (*convert-nnf-ext* φ)
using *composition*
by (*simp add*: *intervals-welldef is-composition-allones-convert-nnf-ext*)
have *cond4*: *set* (*LP-mltl-aux* (*convert-nnf-ext* φ) *k*) =
set (*LP-mltl-aux* (*convert-nnf-ext* φ) *k*)
by *blast*
obtain $\psi 1'$ $\psi 2'$ **where** $\psi 1$: $\psi 1$ = *to-mltl* (*convert-nnf-ext* $\psi 1'$)
and $\psi 1'$*-in*: $\psi 1' \in$ *set ?D*
and $\psi 2$: $\psi 2$ = *to-mltl* (*convert-nnf-ext* $\psi 2'$)
and $\psi 2'$*-in*: $\psi 2' \in$ *set ?D*
using *D-decomp* **unfolding** *LP-mltl.simps*
using *diff-formulas* **by** *auto*
have ψ'*s-neq*: $\psi 1' \neq \psi 2'$
using *diff-formulas* $\psi 1$ $\psi 2$ **by** *blast*
have $\psi 1$*-welldef*: *intervals-welldef* $\psi 1$
using *assms*(*4*) *D-decomp* **unfolding** *LP-mltl.simps*
using *LP-mltl-aux-intervals-welldef*
by (*metis* $\psi 1$ $\psi 1'$*-in allones-implies-is-composition-MLTL composition convert-nnf-ext-to-mltl-commute intervals-welldef nnf-intervals-welldef*)
then have $\psi 1'$*-welldef*: *intervals-welldef* (*to-mltl* $\psi 1'$)
using $\psi 1$

using *LP-mltl-aux-intervals-welldef* $\psi 1'$*-in allones-implies-is-composition-MLTL composition intervals-welldef* **by** *auto*
have $\psi 2$*-welldef*: *intervals-welldef* $\psi 2$
using *assms*(*4*) *D-decomp* **unfolding** *LP-mltl.simps*
using *LP-mltl-aux-intervals-welldef*
by (*metis* $\psi 2$ $\psi 2'$*-in allones-implies-is-composition-MLTL composition convert-nnf-ext-to-mltl-commute intervals-welldef nnf-intervals-welldef*)
then have $\psi 2'$*-welldef*: *intervals-welldef* (*to-mltl* $\psi 2'$)
using $\psi 2$
using *LP-mltl-aux-intervals-welldef* $\psi 2'$*-in allones-implies-is-composition-MLTL composition intervals-welldef* **by** *auto*
have *intersect*: *language-mltl-r* (*to-mltl* $\psi 1'$) $r \cap$
language-mltl-r (*to-mltl* $\psi 2'$) $r = \{\}$
using *LP-mltl-language-disjoint-aux*[*OF cond1 cond2 cond3 cond4*, *of* $\psi 1'$ $\psi 2'$ r]
using $\psi 1'$*-in* $\psi 2'$*-in* ψ*'s-neq r-wpd*
by (*metis convert-nnf-ext-preserves-wpd*)
have *semantics-mltl* π (*to-mltl* (*convert-nnf-ext* φ)) =
semantics-mltl π (*to-mltl* φ)
if *intervals-welldef* (*to-mltl* φ)
for φ::*'a mltl-ext* **and** π
using *that* **unfolding** *semantic-equiv-ext-def*
by (*metis convert-nnf-ext-to-mltl-commute convert-nnf-preserves-semantics*)
then show *?thesis* **using** *intersect*
unfolding *language-mltl-r-def* $\psi 1$ $\psi 2$
using $\psi 1'$*-welldef* $\psi 2'$*-welldef*
by *auto*
qed

8.4 Disjointedness Theorem (special case of k=1)

lemma *LP-mltl-language-disjoint-aux-helper-k1*:
fixes φ $\psi 1$ $\psi 2$::*'a mltl-ext* **and** π::*'a set list*
assumes *intervals-welldef*: *intervals-welldef* (*to-mltl* φ)
assumes *is-nnf*: $\exists \varphi$*-init*. φ = *convert-nnf-ext* φ*-init*
assumes *composition*: *is-composition-MLTL* φ
assumes *tracelen*: *length* $\pi \geq$ *wpd-mltl* (*to-mltl* φ)
assumes *D-decomp*: D = *set* (*LP-mltl-aux* φ (*Suc 0*))
assumes *diff-formulas*: ($\psi 1 \in D$) $\wedge$ ($\psi 2 \in D$) $\wedge$ $\psi 1 \neq \psi 2$
assumes *sat1*: *semantics-mltl-ext* π $\psi 1$
assumes *sat2*: *semantics-mltl-ext* π $\psi 2$
shows *False*
proof(*cases* φ)
case *True-mltl-ext*
then show *?thesis* **using** *assms*
unfolding *True-mltl-ext LP-mltl.simps LP-mltl-aux.simps*
by *auto*
next
case *False-mltl-ext*

```
  then show ?thesis using assms
    unfolding False-mltl-ext LP-mltl.simps LP-mltl-aux.simps
    by auto
next
  case (Prop-mltl-ext p)
  then show ?thesis using assms
    unfolding Prop-mltl-ext LP-mltl.simps LP-mltl-aux.simps
    by auto
next
  case (Not-mltl-ext q)
  then have ∃ p. q = Prop-mltl-ext p
    using convert-nnf-form-Not-Implies-Prop assms
  by (metis convert-nnf-ext-to-mltl-commute to-mltl.simps(4) to-mltl-prop-bijective)

  then obtain p where q = Prop-mltl-ext p by blast
  then show ?thesis
    using assms unfolding Not-mltl-ext LP-mltl.simps LP-mltl-aux.simps
    by auto
next
  case (And-mltl-ext α β)
  show ?thesis
    using assms(5) unfolding And-mltl-ext LP-mltl-aux.simps
    using assms(6) by auto
next
  case (Or-mltl-ext α β)
  let ?Dx = [convert-nnf-ext α]
  let ?Dy = [convert-nnf-ext β]
  have D-is: D = set ( And-mltl-list ?Dx ?Dy @
          And-mltl-list [Not_c α] ?Dy @
          And-mltl-list ?Dx [Not_c β])
    using assms(5) unfolding Or-mltl-ext LP-mltl-aux.simps
    by metis
  then have ψ1-eo: List.member (And-mltl-list ?Dx ?Dy) ψ1 ∨
      List.member (And-mltl-list [Not_c α] ?Dy) ψ1 ∨
       List.member (And-mltl-list ?Dx [Not_c β]) ψ1
    using assms(6) by (simp add: member-def)
  have ψ2-eo: List.member (And-mltl-list ?Dx ?Dy) ψ2 ∨
      List.member (And-mltl-list [Not_c α] ?Dy) ψ2 ∨
       List.member (And-mltl-list ?Dx [Not_c β]) ψ2
    using D-is assms(6) by (simp add: member-def)

  have α-iwd: intervals-welldef (to-mltl α)
    using assms(1) unfolding Or-mltl-ext by simp
  have α-nnf: ∃ φ-init. α = convert-nnf-ext φ-init
    using assms(2) unfolding Or-mltl-ext
  by (metis convert-nnf-ext.simps(5) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(4))
  have α-is-comp: is-composition-MLTL α
    using assms unfolding Or-mltl-ext by simp
  have α-wpd: wpd-mltl (to-mltl α) ≤ length π
```

```
    using assms unfolding Or-mltl-ext by simp
  have α-conv-same: set (LP-mltl-aux (convert-nnf-ext α) 1) = set (LP-mltl-aux
α 1)
    by (metis α-nnf convert-nnf-ext-convert-nnf-ext)

  have β-iwd: intervals-welldef (to-mltl β)
    using assms unfolding Or-mltl-ext
    by simp
  have β-nnf: ∃φ-init. β = convert-nnf-ext φ-init
    using assms unfolding Or-mltl-ext
  by (metis convert-nnf-ext.simps(5) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(4))
  have β-is-comp: is-composition-MLTL β
    using assms unfolding Or-mltl-ext
    by simp
  have β-wpd: wpd-mltl (to-mltl β) ≤ length π
    using assms unfolding Or-mltl-ext by simp
  have β-conv-same: set (LP-mltl-aux (convert-nnf-ext β) k) = set (LP-mltl-aux
β k)
    by (metis β-nnf convert-nnf-ext-convert-nnf-ext)

  {
    assume List.member (And-mltl-list ?Dx ?Dy) ψ1
    then have ψ1-is: ψ1 = And-mltl-ext α β
      unfolding List.member-def
      using α-nnf β-nnf convert-nnf-ext-convert-nnf-ext
     by (metis And-mltl-list-member ‹List.member (And-mltl-list [convert-nnf-ext
α] [convert-nnf-ext β]) ψ1› member-rec(1) member-rec(2))
    have x1-semantics: semantics-mltl-ext π α and
        y1-semantics: semantics-mltl-ext π β
      using assms(7) unfolding ψ1-is semantics-mltl-ext-def by simp-all
    {
      assume List.member (And-mltl-list ?Dx ?Dy) ψ2
      then have ψ2-is: ψ2 = And-mltl-ext α β
        unfolding List.member-def
        using α-nnf β-nnf convert-nnf-ext-convert-nnf-ext
            by (metis And-mltl-list-member-forward ‹List.member (And-mltl-list
[convert-nnf-ext α] [convert-nnf-ext β]) ψ2› member-rec(1) member-rec(2))
      then have ?thesis
        using ψ1-is assms by blast
    } moreover {
      assume  List.member (And-mltl-list [Not_c α] ?Dy) ψ2
      then have ψ2-is: ψ2 = And-mltl-ext (Not_c α) β
        unfolding List.member-def
        using α-nnf β-nnf convert-nnf-ext-convert-nnf-ext
          by (metis And-mltl-list-member ‹List.member (And-mltl-list [Not_c α]
[convert-nnf-ext β]) ψ2› member-rec(1) member-rec(2))
      have x2-semantics: semantics-mltl-ext π (Not_c α) and
          y2-semantics: semantics-mltl-ext π β
        using assms unfolding semantics-mltl-ext-def ψ2-is by simp-all
```

then have *?thesis*
using *x1-semantics* **unfolding** *semantics-mltl-ext-def* **by** *simp*
} moreover {
assume *List.member* (*And-mltl-list ?Dx* [Not_c β]) $\psi 2$
then have $\psi 2$*-is*: $\psi 2$ = *And-mltl-ext* α (Not_c β)
unfolding *List.member-def*
using α*-nnf* β*-nnf convert-nnf-ext-convert-nnf-ext*
by (*metis And-mltl-list-member* ‹*List.member* (*And-mltl-list* [*convert-nnf-ext* α] [Not_c β]) $\psi 2$› *member-rec*(*1*) *member-rec*(*2*))
have *x2-semantics*: *semantics-mltl-ext* π α **and**
y2-semantics: *semantics-mltl-ext* π (Not_c β)
using *assms* **unfolding** *semantics-mltl-ext-def* $\psi 2$*-is* **by** *simp-all*
then have *?thesis*
using *y1-semantics* **unfolding** *semantics-mltl-ext-def* **by** *simp*
}
ultimately have *?thesis*
using $\psi 2$*-eo* **by** *argo*
} moreover {
assume *List.member* (*And-mltl-list* [Not_c α] *?Dy*) $\psi 1$
then have $\psi 1$*-is*: $\psi 1$ = *And-mltl-ext* (Not_c α) (β)
unfolding *List.member-def*
using α*-nnf* β*-nnf convert-nnf-ext-convert-nnf-ext*
by (*metis And-mltl-list-member* ‹*List.member* (*And-mltl-list* [Not_c α] [*convert-nnf-ext* β]) $\psi 1$› *member-rec*(*1*) *member-rec*(*2*))
have *x1-semantics*: *semantics-mltl-ext* π (Not_c α) **and**
y1-semantics: *semantics-mltl-ext* π (β)
using *assms* **unfolding** *semantics-mltl-ext-def* $\psi 1$*-is* **by** *simp-all*

{
assume *List.member* (*And-mltl-list ?Dx ?Dy*) $\psi 2$
then have $\psi 2$*-is*: $\psi 2$ = *And-mltl-ext* α β
unfolding *List.member-def*
using α*-nnf* β*-nnf convert-nnf-ext-convert-nnf-ext*
by (*metis And-mltl-list-member* ‹*List.member* (*And-mltl-list* [*convert-nnf-ext* α] [*convert-nnf-ext* β]) $\psi 2$› *member-rec*(*1*) *member-rec*(*2*))
have *?thesis*
using *assms*(*7*,*8*) **unfolding** $\psi 1$*-is* $\psi 2$*-is semantics-mltl-ext-def* **by** *auto*
} moreover {
assume *List.member* (*And-mltl-list* [Not_c α] *?Dy*) $\psi 2$
then have $\psi 2$*-is*: $\psi 2$ = *And-mltl-ext* (Not_c α) β
unfolding *List.member-def*
using α*-nnf* β*-nnf convert-nnf-ext-convert-nnf-ext*
by (*metis And-mltl-list-member* ‹*List.member* (*And-mltl-list* [Not_c α] [*convert-nnf-ext* β]) $\psi 2$› *member-rec*(*1*) *member-rec*(*2*))
have *x2-semantics*: *semantics-mltl-ext* π (Not_c α) **and**
y2-semantics: *semantics-mltl-ext* π β
using *assms* **unfolding** *semantics-mltl-ext-def* $\psi 2$*-is* **by** *simp-all*
then have *?thesis*
using $\psi 1$*-is* $\psi 2$*-is assms* **by** *blast*

```
    } moreover {
      assume List.member (And-mltl-list ?Dx [Not_c β]) ψ2
      then have ψ2-is: ψ2 = And-mltl-ext α (Not_c β)
        unfolding List.member-def
        using α-nnf β-nnf convert-nnf-ext-convert-nnf-ext
       by (metis And-mltl-list-member ‹List.member (And-mltl-list [convert-nnf-ext
α] [Not_c β]) ψ2› member-rec(1) member-rec(2))
      have x2-semantics: semantics-mltl-ext π α and
           y2-semantics: semantics-mltl-ext π (Not_c β)
        using assms unfolding semantics-mltl-ext-def ψ2-is by simp-all
      then have ?thesis
        using y1-semantics unfolding semantics-mltl-ext-def by simp
    }
    ultimately have ?thesis
      using ψ2-eo by argo
  } moreover {
    assume List.member (And-mltl-list ?Dx [Not_c β]) ψ1
    then have ψ1-is: ψ1 = And-mltl-ext α (Not_c β)
      unfolding List.member-def
      using α-nnf β-nnf convert-nnf-ext-convert-nnf-ext
     by (metis And-mltl-list-member ‹List.member (And-mltl-list [convert-nnf-ext
α] [Not_c β]) ψ1› member-rec(1) member-rec(2))
    have x1-semantics: semantics-mltl-ext π α and
         y1-semantics: semantics-mltl-ext π (Not_c β)
      using assms unfolding semantics-mltl-ext-def ψ1-is by simp-all

    {
      assume List.member (And-mltl-list ?Dx ?Dy) ψ2
      then have ψ2-is: ψ2 = And-mltl-ext α β
        unfolding List.member-def
        using α-nnf β-nnf convert-nnf-ext-convert-nnf-ext
          by (metis And-mltl-list-member-forward ‹List.member (And-mltl-list
[convert-nnf-ext α] [convert-nnf-ext β]) ψ2› member-rec(1) member-rec(2))
      have ?thesis
        using assms(7,8) unfolding ψ1-is ψ2-is semantics-mltl-ext-def by auto
    } moreover {
      assume  List.member (And-mltl-list [Not_c α] ?Dy) ψ2
      then have ψ2-is: ψ2 = And-mltl-ext (Not_c α) β
        unfolding List.member-def
        using α-nnf β-nnf convert-nnf-ext-convert-nnf-ext
          by (metis And-mltl-list-member ‹List.member (And-mltl-list [Not_c α]
[convert-nnf-ext β]) ψ2› member-rec(1) member-rec(2))
      have x2-semantics: semantics-mltl-ext π (Not_c α) and
           y2-semantics: semantics-mltl-ext π β
        using assms unfolding semantics-mltl-ext-def ψ2-is by simp-all
      then have ?thesis
         using x1-semantics x2-semantics unfolding semantics-mltl-ext-def by
auto
    } moreover {
```

```
      assume List.member (And-mltl-list ?Dx [Not_c β]) ψ2
      then have ψ2-is: ψ2 = And-mltl-ext α (Not_c β)
        unfolding List.member-def
        using α-nnf β-nnf convert-nnf-ext-convert-nnf-ext
      by (metis And-mltl-list-member ‹List.member (And-mltl-list [convert-nnf-ext
α] [Not_c β]) ψ2› member-rec(1) member-rec(2))
      have x2-semantics: semantics-mltl-ext π α and
           y2-semantics: semantics-mltl-ext π (Not_c β)
        using assms unfolding semantics-mltl-ext-def ψ2-is by simp-all
      then have ?thesis
        using ψ1-is ψ2-is assms by blast
    }
    ultimately have ?thesis
      using ψ2-eo by argo
  }
  ultimately show ?thesis
    using ψ1-eo by argo
next
  case (Future-mltl-ext a b L α)
  have a-leq-b: a ≤ b and
       α-welldef: intervals-welldef (to-mltl α)
    using assms unfolding intervals-welldef.simps Future-mltl-ext to-mltl.simps
    by simp-all
  have α-nnf: ∃φ-init. α = convert-nnf-ext φ-init
    using assms unfolding Future-mltl-ext
  by (metis convert-nnf-ext.simps(6) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(5))

  have α-convert: convert-nnf-ext α = α
    using α-nnf convert-nnf-ext-convert-nnf-ext by metis
  have α-composition: is-composition-MLTL α and
       L-composition: is-composition (b−a+1) L
    using Future-mltl-ext assms by simp-all
  have α-wpd: b + wpd-mltl (to-mltl α) ≤ length π
    using assms unfolding Future-mltl-ext to-mltl.simps wpd-mltl.simps
    by auto
  let ?D = [α]
  let ?s = interval-times a L
  have length-L: 1 ≤ length L
    using composition-length-lb[OF L-composition] a-leq-b by linarith
  have sfirst: ?s!0 = a
    using interval-times-first by simp
  have slast: ?s!(length L) = b+1
    using interval-times-last[OF a-leq-b L-composition] by blast
  have length-s: length ?s = length L + 1
    using interval-times-length by simp
  let ?front = set [Future-mltl-ext (?s ! 0) (?s ! 1 − 1) [?s ! 1 − ?s ! 0] α]
  let ?back = set (concat (map (λi. And-mltl-list
                    [Global-mltl-ext (?s ! 0) (?s ! i − 1) [?s!i − ?s!0] (Not_c α)]
                     [Future-mltl-ext (?s ! i) (?s ! (i + 1) − 1) [?s ! (i + 1) −
```

```
?s ! i] α])
                [1..<length L]))
    have front-eq: set (Future-mltl-list ?D (?s ! 0) (?s ! 1 − 1) [?s ! 1 − ?s ! 0])
= ?front
      by simp
    have back-eq: ?back = set (concat
          (map (λi. And-mltl-list
                    [Global-mltl-ext (?s ! 0) (?s ! i − 1) [?s!i − ?s!0] (Not_c α)]
                    (Future-mltl-list ?D (?s ! i) (?s ! (i + 1) − 1)
                      [?s ! (i + 1) − ?s ! i]))
            [1..<length L]))
      by auto
    have D-is: D = ?front ∪ ?back
      using assms(5) unfolding Future-mltl-ext LP-mltl-aux.simps to-mltl.simps
      using list-concat-set-union unfolding α-convert
      using front-eq back-eq
      by (metis (no-types, lifting))
    have dropa-wpd: wpd-mltl (to-mltl α) ≤ length (drop a π)
      using α-wpd a-leq-b by simp
    {
      assume ∗: ψ1 ∈ ?front
      then have ψ1: ψ1 = Future-mltl-ext (?s!0) (?s!1−1) [?s!1 − ?s!0] α
        by auto
      obtain j1 where α-semantics1: semantics-mltl-ext (drop j1 π) α
                  and j1-bound: a ≤ j1 ∧ j1 ≤ ?s!1−1
     using assms(7) unfolding sfirst ψ1 semantics-mltl-ext-def semantics-mltl.simps
to-mltl.simps
        by blast
      {
        assume ∗∗: ψ2 ∈ ?front
        then have ψ2: ψ2 = Future-mltl-ext (?s!0) (?s!1−1) [?s!1 − ?s!0] α
          by auto
        obtain j2 where α-semantics-2: semantics-mltl-ext (drop j2 π) α
                    and j2-bound: a ≤ j2 ∧ j2 ≤ ?s!1−1
              using assms(8) unfolding sfirst ψ2 semantics-mltl-ext-def seman-
tics-mltl.simps to-mltl.simps
          by blast
        have ?thesis
          using assms(6) ψ1 ψ2 by blast
      } moreover {
        assume ∗∗: ψ2 ∈ ?back
        then obtain i2 where ψ2: ψ2 = (And-mltl-ext
                    (Global-mltl-ext (?s ! 0) (?s ! i2 − 1) [?s!i2 − ?s!0] (Not_c α))
                      (Future-mltl-ext (?s ! i2) (?s ! (i2 + 1) − 1) [?s ! (i2 + 1)
− ?s ! i2] α))
          and i2-bound: 1 ≤ i2 ∧ i2 < length L
          by force
        obtain j2 where α-semantics2: semantics-mltl-ext (drop j2 π) α
                  and j2-bound: ?s!i2 ≤ j2 ∧ j2 ≤ ?s!(i2+1)−1
```

```
                and global-before2: ∀ i. a ≤ i ∧ i ≤ ?s ! i2 − 1 ⟶
                    ¬ semantics-mltl (drop i π) (to-mltl α)
        using assms(8) unfolding ψ2 semantics-mltl-ext-def to-mltl.simps seman-
tics-mltl.simps
          unfolding sfirst using α-wpd a-leq-b by auto
        have bound1: interval-times a L ! 1 ≤ interval-times a L ! i2
          using interval-times-diff-ge-general[OF a-leq-b L-composition, of i2 1 ?s]
          using i2-bound by force
        have ?thesis using bound1
       using α-semantics1 global-before2 j1-bound unfolding semantics-mltl-ext-def
          by auto
      }
      ultimately have ?thesis
        using assms(6) D-is by blast
    } moreover {
      assume ∗: ψ1 ∈ ?back
      then obtain i1 where ψ1: ψ1 = (And-mltl-ext
                    (Global-mltl-ext (?s ! 0) (?s ! i1 − 1) [?s!i1 − ?s!0] (Not_c α))
                      (Future-mltl-ext (?s ! i1) (?s ! (i1 + 1) − 1) [?s ! (i1 + 1)
− ?s ! i1] α))
          and i1-bound: 1 ≤ i1 ∧ i1 < length L
        by force
      have lb1: a ≤ ?s!i1
        using interval-times-diff-ge-general[OF a-leq-b L-composition, of i1 0 ?s]
        unfolding sfirst using i1-bound by simp
      have welldef1: ?s!i1 < ?s!(i1+1)
        using interval-times-diff-ge[OF a-leq-b L-composition, of i1 ?s]
        using i1-bound by blast
      have ub1: ?s!(i1+1)−1 ≤ b
        using interval-times-diff-ge-general[OF a-leq-b L-composition, of length L
i1+1 ?s]
        using slast i1-bound
        by (metis le-diff-conv le-eq-less-or-eq less-iff-succ-less-eq)
      obtain j1 where α-semantics1: semantics-mltl-ext (drop j1 π) α
                and j1-bound: ?s!i1 ≤ j1 ∧ j1 ≤ ?s!(i1+1)−1
                and global-before1: ∀ i. a ≤ i ∧ i ≤ ?s ! i1 − 1 ⟶
                    ¬ semantics-mltl (drop i π) (to-mltl α)
        using assms(7) unfolding ψ1 semantics-mltl-ext-def to-mltl.simps seman-
tics-mltl.simps
        unfolding sfirst using α-wpd a-leq-b by auto
      have bound1: interval-times a L ! 1 ≤ interval-times a L ! i1
        using interval-times-diff-ge-general[OF a-leq-b L-composition, of i1 1 ?s]
        using i1-bound by force
      {
        assume ∗∗: ψ2 ∈ ?front
        then have ψ2: ψ2 = Future-mltl-ext (?s!0) (?s!1−1) [?s!1 − ?s!0] α
          by auto
        obtain j2 where α-semantics2: semantics-mltl-ext (drop j2 π) α
                  and j2-bound: a ≤ j2 ∧ j2 ≤ ?s!1−1
```

using *assms*(8) **unfolding** *sfirst* $\psi 2$ *semantics-mltl-ext-def semantics-mltl.simps to-mltl.simps*
by *blast*
then have *?thesis*
using *global-before1* α*-semantics2 bound1*
unfolding *semantics-mltl-ext-def* **by** *auto*
} moreover {
assume $**$: $\psi 2 \in$ *?back*
then obtain *i2* **where** $\psi 2$: $\psi 2 = (And\text{-}mltl\text{-}ext$
$(Global\text{-}mltl\text{-}ext\ (?s\ !\ 0)\ (?s\ !\ i2\ -\ 1)\ [?s!i2\ -\ ?s!0]\ (Not_c\ \alpha))$
$(Future\text{-}mltl\text{-}ext\ (?s\ !\ i2)\ (?s\ !\ (i2\ +\ 1)\ -\ 1)\ [?s\ !\ (i2\ +\ 1)\ -\ ?s\ !\ i2]\ \alpha))$
and *i2-bound*: $1 \leq i2 \wedge i2 < length\ L$
by *force*
obtain *j2* **where** α*-semantics2*: *semantics-mltl-ext* $(drop\ j2\ \pi)\ \alpha$
and *j2-bound*: *?s*!*i2* $\leq$ *j2* $\wedge$ *j2* $\leq$ *?s*!$(i2+1)-1$
and *global-before2*: $\forall i.\ a \leq i \wedge i \leq ?s\ !\ i2\ -\ 1 \longrightarrow$
$\neg$ *semantics-mltl* $(drop\ i\ \pi)\ (to\text{-}mltl\ \alpha)$
using *assms*(8) **unfolding** $\psi 2$ *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
unfolding *sfirst* **using** α*-wpd a-leq-b* **by** *auto*
have *lb2*: $a \leq$ *?s*!*i2*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i2 0 ?s*]
unfolding *sfirst* **using** *i2-bound* **by** *simp*
have *welldef2*: *?s*!*i2* $<$ *?s*!$(i2+1)$
using *interval-times-diff-ge*[*OF a-leq-b L-composition*, *of i2 ?s*]
using *i2-bound* **by** *blast*
have *ub2*: *?s*!$(i2+1)-1 \leq b$
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L i2+1 ?s*]
using *slast i2-bound*
by (*metis le-diff-conv le-eq-less-or-eq less-iff-succ-less-eq*)
{
assume *i1-eq-i2*: *i1* $=$ *i2*
then have *?thesis*
using *assms*(6) $\psi 1$ $\psi 2$ **by** *blast*
} moreover {
assume *i1-le-i2*: *i1* $<$ *i2*
then have *?s* ! (*i1* + *1*) $\leq$ *?s* ! *i2*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i2 i1+1 ?s*]
using *i1-bound i2-bound*
by (*metis le-eq-less-or-eq less-iff-succ-less-eq*)
then have *j1* $\leq$ *interval-times a L* ! *i2* $-$ *1*
using *j1-bound* **by** *auto*
then have *?thesis*
using α*-semantics1 global-before2 j1-bound lb1*
unfolding *semantics-mltl-ext-def* **by** *simp*
} moreover {

```
        assume i1-ge-i2: i1 > i2
        then have ?s ! (i2 + 1) ≤ ?s ! i1
         using interval-times-diff-ge-general[OF a-leq-b L-composition, of i1 i2+1
?s]
          using i2-bound i1-bound
          by (metis le-eq-less-or-eq less-iff-succ-less-eq)
        then have j2 ≤ interval-times a L ! i1 − 1
          using j2-bound by auto
        then have ?thesis
          using α-semantics2 global-before1 j2-bound lb2
          unfolding semantics-mltl-ext-def by simp
       }
       ultimately have ?thesis by linarith
      }
      ultimately have ?thesis
       using assms(6) D-is by blast
    }
    ultimately show ?thesis
     using assms(6) D-is by blast
  next
    case (Global-mltl-ext a b L α)
    have a-leq-b: a ≤ b and
         α-welldef: intervals-welldef (to-mltl α)
      using assms unfolding intervals-welldef.simps Global-mltl-ext to-mltl.simps
       by simp-all
    have α-nnf: ∃ φ-init. α = convert-nnf-ext φ-init
      using assms unfolding Global-mltl-ext
     by (metis convert-nnf-ext.simps(7) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(6))

    have α-convert: convert-nnf-ext α = α
      using α-nnf convert-nnf-ext-convert-nnf-ext by metis
    have α-composition: is-composition-MLTL α
      using Global-mltl-ext assms by simp-all
    have α-wpd: b + wpd-mltl (to-mltl α) ≤ length π
      using assms unfolding Global-mltl-ext to-mltl.simps wpd-mltl.simps
      by auto
    have D-is: D = {Global-mltl-ext a b L α}
      using assms(5) unfolding Global-mltl-ext LP-mltl-aux.simps α-convert
      by auto
    then show ?thesis
      using assms by blast
  next
    case (Until-mltl-ext α a b L β)
    have a-leq-b: a ≤ b and
         α-welldef: intervals-welldef (to-mltl α) and
         β-welldef: intervals-welldef (to-mltl β)
      using assms unfolding intervals-welldef.simps Until-mltl-ext to-mltl.simps
      by simp-all
    have α-nnf: ∃ φ-init. α = convert-nnf-ext φ-init
```

```
  using assms unfolding Until-mltl-ext
 by (metis convert-nnf-ext.simps(8) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(7))

 have α-convert: convert-nnf-ext α = α
  using α-nnf convert-nnf-ext-convert-nnf-ext by metis
 have β-nnf: ∃φ-init. β = convert-nnf-ext φ-init
  using assms unfolding Until-mltl-ext
 by (metis convert-nnf-ext.simps(8) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(7))

 have β-convert: convert-nnf-ext β = β
  using β-nnf convert-nnf-ext-convert-nnf-ext by metis
 have α-composition: is-composition-MLTL α and
     β-composition: is-composition-MLTL β and
     L-composition: is-composition (b−a+1) L
  using Until-mltl-ext assms by simp-all
 have α-wpd: b + wpd-mltl (to-mltl α) ≤ length π and
     β-wpd: b + wpd-mltl (to-mltl β) ≤ length π
  using assms unfolding Until-mltl-ext to-mltl.simps wpd-mltl.simps
  by auto
 let ?s = interval-times a L
 have length-L: 1 ≤ length L
  using composition-length-lb[OF L-composition] a-leq-b by linarith
 have sfirst: ?s!0 = a
  using interval-times-first by simp
 have slast: ?s!(length L) = b+1
  using interval-times-last[OF a-leq-b L-composition]
  by blast
 have length-s: length ?s = length L + 1
  using interval-times-length by simp
 let ?D = [β]
 let ?front = {Until-mltl-ext α (?s ! 0) (?s ! 1 − 1) [?s ! 1 − ?s ! 0] β}
 let ?back = set (map (λi. And-mltl-ext
                    (Global-mltl-ext
                      (?s ! 0) (?s ! i − 1) [?s!i − ?s!0] (And-mltl-ext α (Not_c
β)))
                    (Until-mltl-ext α (?s ! i) (?s ! (i + 1) − 1)
                      [?s ! (i + 1) − ?s ! i] β)) [1..<length L])
 have front-eq: ?front = set (Until-mltl-list α ?D (?s ! 0) (?s ! 1 − 1) [?s ! 1
− ?s ! 0])
  by simp
 have back-eq: ?back = set (concat
       (map (λi. And-mltl-list
                 [Global-mltl-ext
                  (?s ! 0) (?s ! i − 1) [?s!i − ?s!0] (And-mltl-ext α (Not_c β))]
                 (Until-mltl-list α ?D (?s ! i) (?s ! (i + 1) − 1)
                   [?s ! (i + 1) − ?s ! i]))
         [1..<length L]))
  by simp
 have D-is: D = ?front ∪ ?back
```

using *assms*(5) **unfolding** *Until-mltl-ext LP-mltl-aux.simps*
using α*-convert* β*-convert list-concat-set-union* **using** *front-eq back-eq*
by (*smt* (*verit*) *map-eq-conv*)
{
assume $*$: $\psi 1 \in$ *?front*
then have $\psi 1$: $\psi 1$ = *Until-mltl-ext* α (*?s* ! 0) (*?s* ! 1 − 1) [*?s* ! 1 − *?s* ! 0] β
by *blast*
obtain *j1* **where** *j1-bound*: *?s*!0 ≤ *j1* ∧ *j1* ≤ *?s*!1−1
and β*-semantics1*: *semantics-mltl-ext* (*drop j1* π) β
and α*-semantics1*: ∀ *j*. (*?s*!0 ≤ *j* ∧ *j* < *j1*) ⟶ (*semantics-mltl-ext* (*drop j* π) α)
using *assms*(7) **unfolding** $\psi 1$ *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *blast*
{
assume $**$: $\psi 2 \in$ *?front*
then have $\psi 2$: $\psi 2$ = *Until-mltl-ext* α (*?s* ! 0) (*?s* ! 1 − 1) [*?s* ! 1 − *?s* ! 0] β
by *blast*
obtain *j2* **where** *j2-bound*: *?s*!0 ≤ *j2* ∧ *j2* ≤ *?s*!1−1
and β*-semantics2*: *semantics-mltl-ext* (*drop j2* π) β
and α*-semantics2*: ∀ *j*. (*?s*!0 ≤ *j* ∧ *j* < *j2*) ⟶ (*semantics-mltl-ext* (*drop j2* π) α)
using *assms*(8) **unfolding** $\psi 2$ *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using $\psi 1$ $\psi 2$ *diff-formulas* **by** *blast*
have *?thesis*
using $\psi 1$ $\psi 2$ *diff-formulas* **by** *blast*
} moreover {
assume $**$: $\psi 2 \in$ *?back*
then obtain *i2* **where** $\psi 2$: $\psi 2$ = *And-mltl-ext*
(*Global-mltl-ext* (*?s* ! 0) (*?s* ! *i2* − 1) [*?s*!*i2* − *?s*!0] (*And-mltl-ext* α (Not_c β)))
(*Until-mltl-ext* α (*?s* ! *i2*) (*?s* ! (*i2* + 1) − 1) [*?s* ! (*i2* + 1) − *?s* ! *i2*] β)
and *i2-bound*: 1 ≤ *i2* ∧ *i2* < *length L*
by *auto*
obtain *j2* **where** *j2-bound*: (*?s* ! *i2*) ≤ *j2* ∧ *j2* ≤ (*?s* ! (*i2* + 1) − 1)
and β*-semantics2*: *semantics-mltl* (*drop j2* π) (*to-mltl* β)
and α*-semantics2*: (∀ *j*. *interval-times a L* ! *i2* ≤ *j* ∧ *j* < *j2* ⟶ *semantics-mltl* (*drop j* π) (*to-mltl* α))
and *global-before2*: ∀ *i*. *?s* ! 0 ≤ *i* ∧ *i* ≤ *?s* ! *i2* − 1 ⟶ *semantics-mltl* (*drop i* π) (*to-mltl* α) ∧ ¬ *semantics-mltl* (*drop i* π) (*to-mltl* β)
using *assms*(8) **unfolding** $\psi 2$ *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using α*-wpd* **by** *auto*
have *bound1*: *?s* ! 1 ≤ *?s* ! *i2*

using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i2 1 ?s*]
using *i2-bound* **by** *force*
then have *?thesis*
using β*-semantics1 global-before2 j1-bound* **unfolding** *sfirst*
unfolding *semantics-mltl-ext-def* **by** *auto*
}
ultimately have *?thesis* **using** *D-is assms* **by** *blast*
} **moreover** {
assume $*$: $\psi 1 \in$ *?back*
then obtain *i1* **where** $\psi 1$: $\psi 1$ = *And-mltl-ext*
(*Global-mltl-ext* (*?s* ! *0*) (*?s* ! *i1* − *1*) [*?s*!*i1* − *?s*!*0*] (*And-mltl-ext* α (*Not*$_c$ β)))
(*Until-mltl-ext* α (*?s* ! *i1*) (*?s* ! (*i1* + *1*) − *1*) [*?s* ! (*i1* + *1*) − *?s* ! *i1*] β)
and *i1-bound*: *1* ≤ *i1* ∧ *i1* < *length L*
by *auto*
have *lb1*: *a* ≤ *?s*!*i1*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i1 0 ?s*]
unfolding *sfirst* **using** *i1-bound* **by** *simp*
have *welldef1*: *?s*!*i1* < *?s*!(*i1*+*1*)
using *interval-times-diff-ge*[*OF a-leq-b L-composition*, *of i1 ?s*]
using *i1-bound* **by** *blast*
have *ub1*: *?s*!(*i1*+*1*)−*1* ≤ *b*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L i1+1 ?s*]
using *slast i1-bound*
by (*metis le-diff-conv le-eq-less-or-eq less-iff-succ-less-eq*)
obtain *j1* **where** *j1-bound*: (*?s* ! *i1*) ≤ *j1* ∧ *j1* ≤ (*?s* ! (*i1* + *1*) − *1*)
and β*-semantics1*: *semantics-mltl* (*drop j1* π) (*to-mltl* β)
and α*-semantics1*: (∀ *j*. *interval-times a L* ! *i1* ≤ *j* ∧ *j* < *j1* ⟶
semantics-mltl (*drop j* π) (*to-mltl* α))
and *global-before1*: ∀ *i*. *?s* ! *0* ≤ *i* ∧ *i* ≤ *?s* ! *i1* − *1* ⟶
semantics-mltl (*drop i* π) (*to-mltl* α) ∧
¬ *semantics-mltl* (*drop i* π) (*to-mltl* β)
using *assms*(*7*) **unfolding** $\psi 1$ *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using α*-wpd* **by** *auto*
have *bound1*: *?s* ! *1* ≤ *?s* ! *i1*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i1 1 ?s*]
using *i1-bound* **by** *force*
{
assume $**$: $\psi 2 \in$ *?front*
then have $\psi 2$: $\psi 2$ = *Until-mltl-ext* α (*?s* ! *0*) (*?s* ! *1* − *1*) [*?s* ! *1* − *?s* ! *0*] β
by *blast*
have *?thesis*
using *assms*(*8*) **unfolding** $\psi 2$ *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
unfolding *sfirst*

by (*smt* (*verit*, *ccfv-SIG*) *bound1 diff-is-0-eq′ global-before1 interval-times-first le0 le-trans nat-le-linear ordered-cancel-comm-monoid-diff-class.le-diff-conv2*)
} moreover {
assume ∗∗: $\psi 2 \in ?back$
then obtain *i2* **where** $\psi 2$: $\psi 2 = And\text{-}mltl\text{-}ext$
($Global\text{-}mltl\text{-}ext$ ($?s$! 0) ($?s$! $i2 - 1$) [$?s!i2 - ?s!0$] ($And\text{-}mltl\text{-}ext$ α (Not_c β)))
($Until\text{-}mltl\text{-}ext$ α ($?s$! $i2$) ($?s$! ($i2 + 1$) $- 1$) [$?s$! ($i2 + 1$) $-$ $?s$! $i2$] β)
and *i2-bound*: $1 \leq i2 \wedge i2 < length\ L$
by *auto*
have *lb2*: $a \leq ?s!i2$
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i2 0 ?s*]
unfolding *sfirst* **using** *i2-bound* **by** *simp*
have *welldef2*: $?s!i2 < ?s!(i2+1)$
using *interval-times-diff-ge*[*OF a-leq-b L-composition*, *of i2 ?s*]
using *i2-bound* **by** *blast*
have *ub2*: $?s!(i2+1)-1 \leq b$
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L i2+1 ?s*]
using *slast i2-bound*
by (*metis le-diff-conv le-eq-less-or-eq less-iff-succ-less-eq*)
obtain *j2* **where** *j2-bound*: ($?s$! $i2$) $\leq j2 \wedge j2 \leq$ ($?s$! ($i2$ + 1) − 1)
and β*-semantics2*: *semantics-mltl* (*drop j2* π) (*to-mltl* β)
and α*-semantics2*: ($\forall j$. *interval-times a L* ! *i2* $\leq j \wedge j < j2 \longrightarrow$
semantics-mltl (*drop j* π) (*to-mltl* α))
and *global-before2*: $\forall i$. $?s$! $0 \leq i \wedge i \leq ?s$! $i2 - 1 \longrightarrow$
semantics-mltl (*drop i* π) (*to-mltl* α) $\wedge$
$\neg$ *semantics-mltl* (*drop i* π) (*to-mltl* β)
using *assms*(*8*) **unfolding** $\psi 2$ *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using α*-wpd* **by** *auto*
{
assume *i1-eq-i2*: *i1* = *i2*
then have *?thesis*
using *assms*(*6*) $\psi 1$ $\psi 2$ **by** *blast*
} moreover {
assume *i1-le-i2*: *i1* < *i2*
then have $?s$! ($i1$ + 1) $\leq$ $?s$! $i2$
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i2 i1+1 ?s*]
using *i1-bound i2-bound*
by (*metis le-eq-less-or-eq less-iff-succ-less-eq*)
then have *?thesis*
using β*-semantics1 global-before2 j1-bound* **unfolding** *sfirst*
using *lb1* **by** *auto*
} moreover {
assume *i1-ge-i2*: *i1* > *i2*
then have $?s$! ($i2$ + 1) $\leq$ $?s$! *i1*

```
        using interval-times-diff-ge-general[OF a-leq-b L-composition, of i1 i2+1
?s]
         using i1-bound i2-bound
         by (metis le-eq-less-or-eq less-iff-succ-less-eq)
       then have ?thesis
         using β-semantics2 global-before1 j2-bound unfolding sfirst
         using lb2 by auto
      }
      ultimately have ?thesis by linarith
    }
    ultimately have ?thesis
      using D-is assms by blast
  }
  ultimately show ?thesis
    using D-is assms by blast
next
  case (Release-mltl-ext α a b L β)
  have a-leq-b: a ≤ b and
       α-welldef: intervals-welldef (to-mltl α) and
       β-welldef: intervals-welldef (to-mltl β)
    using assms unfolding intervals-welldef.simps Release-mltl-ext to-mltl.simps
    by simp-all
  have α-nnf: ∃φ-init. α = convert-nnf-ext φ-init
    using assms unfolding Release-mltl-ext
  by (metis convert-nnf-ext.simps(9) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(8))

  have α-convert: convert-nnf-ext α = α
    using α-nnf convert-nnf-ext-convert-nnf-ext by metis
  have β-nnf: ∃φ-init. β = convert-nnf-ext φ-init
    using assms unfolding Release-mltl-ext
  by (metis convert-nnf-ext.simps(9) convert-nnf-ext-convert-nnf-ext mltl-ext.inject(8))

  have β-convert: convert-nnf-ext β = β
    using β-nnf convert-nnf-ext-convert-nnf-ext by metis
  have α-composition: is-composition-MLTL α and
       β-composition: is-composition-MLTL β and
       L-composition: is-composition (b−a+1) L
    using Release-mltl-ext assms by simp-all
  have α-wpd: b + wpd-mltl (to-mltl α) ≤ length π and
       β-wpd: b + wpd-mltl (to-mltl β) ≤ length π
    using assms unfolding Release-mltl-ext to-mltl.simps wpd-mltl.simps
    by auto
  let ?s = interval-times a L
  have length-L: 1 ≤ length L
    using composition-length-lb[OF L-composition] a-leq-b by linarith
  have sfirst: ?s!0 = a
    using interval-times-first by simp
  have slast: ?s!(length L) = b+1
    using interval-times-last[OF a-leq-b L-composition]
```

by *blast*
have *length-s*: *length ?s = length L + 1*
using *interval-times-length* **by** *simp*
let *?D* = [α]
let *?front* = {*Global-mltl-ext a b L* (*And-mltl-ext* (*Not*$_c$ α) β)}
let *?middle* = {*Mighty-Release-mltl-ext* α β (*?s ! 0*) (*?s ! 1 − 1*)
[*?s ! 1 − ?s ! 0*]}
let *?back* = *set* (*map* (λ*i. And-mltl-ext*
(*Global-mltl-ext*
(*?s ! 0*) (*?s ! i − 1*) [*?s!i − ?s!0*] (*And-mltl-ext* (*Not*$_c$ α) β))
(*Mighty-Release-mltl-ext* α β (*?s ! i*)
(*?s ! (i + 1) − 1*) [*?s ! (i + 1) − ?s ! i*]))
[*1..<length L*])
have *middle-eq*: *?middle = set* (*Mighty-Release-mltl-list ?D* β (*?s ! 0*) (*?s ! 1 − 1*) [*?s ! 1 − ?s ! 0*])
by *simp*
have *back-eq*: *?back = set* (*concat*
(*map* (λ*i. And-mltl-list*
[*Global-mltl-ext*
(*?s ! 0*) (*?s ! i − 1*) [*?s!i − ?s!0*] (*And-mltl-ext* (*Not*$_c$ α) β)]
(*Mighty-Release-mltl-list ?D* β (*?s ! i*)
(*?s ! (i + 1) − 1*) [*?s ! (i + 1) − ?s ! i*]))
[*1..<length L*]))
by *simp*
have *D-is*: *D = ?front* $\cup$ *?middle* $\cup$ *?back*
using *assms*(*5*) **unfolding** *Release-mltl-ext LP-mltl-aux.simps*
using α*-convert list-concat-set-union*
using *middle-eq back-eq*
by (*smt* (*verit*, *ccfv-SIG*) *append.assoc empty-set list.simps*(*15*) *map-eq-conv*)

{
assume $*$: ψ*1* $\in$ *?front*
then have ψ*1*: ψ*1 = Global-mltl-ext a b L* (*And-mltl-ext* (*Not*$_c$ α) β)
by *auto*
have *global1*: ($\forall$ *i. a* $\leq$ *i* $\wedge$ *i* $\leq$ *b* $\longrightarrow$
$\neg$ *semantics-mltl* (*drop i* π) (*to-mltl* α) $\wedge$
semantics-mltl (*drop i* π) (*to-mltl* β))
using *assms*(*7*) **unfolding** ψ*1 semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using α*-wpd a-leq-b*
by (*metis add-diff-cancel-left' cancel-comm-monoid-add-class.diff-cancel dual-order.trans le-add1 not-one-le-zero order-antisym-conv wpd-geq-one*)
{
assume $**$: ψ*2* $\in$ *?front*
then have ψ*2*: ψ*2 = Global-mltl-ext a b L* (*And-mltl-ext* (*Not*$_c$ α) β)
by *auto*
have *global2*: ($\forall$ *i. a* $\leq$ *i* $\wedge$ *i* $\leq$ *b* $\longrightarrow$
$\neg$ *semantics-mltl* (*drop i* π) (*to-mltl* α) $\wedge$

semantics-mltl (*drop i* π) (*to-mltl* β))
using *assms*(*8*) **unfolding** $\psi 2$ *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using α*-wpd a-leq-b*
by (*metis add-diff-cancel-left' cancel-comm-monoid-add-class.diff-cancel dual-order.trans le-add1 not-one-le-zero order-antisym-conv wpd-geq-one*)
have *?thesis* **using** $*$ $**$ *assms* **by** *auto*
} moreover {
assume $**$: $\psi 2 \in$ *?middle*
then have $\psi 2$: $\psi 2$ = *Mighty-Release-mltl-ext* α β (*?s* ! *0*) (*?s* ! *1* − *1*) [*?s* ! *1* − *?s* ! *0*]
by *blast*
obtain *j2* **where** *j2-bound*: (*?s* ! *0* ≤ *j2* ∧ *j2* ≤ *?s* ! *1* − *1*)
and α*-semantics2*: *semantics-mltl* (*drop j2* π) (*to-mltl* α)
using *assms*(*8*) **unfolding** $\psi 2$ *Mighty-Release-mltl-ext.simps semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *blast*
have *bound1*: *interval-times a L* ! *1* − *1* ≤ *b*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L 1 ?s*]
using *slast length-L* **by** *force*
then have *?thesis* **using** α*-semantics2 global1 j2-bound* **unfolding** *sfirst*
by *simp*
} moreover {
assume $**$: $\psi 2 \in$ *?back*
then obtain *i2* **where** $\psi 2$: $\psi 2$ = *And-mltl-ext*
(*Global-mltl-ext*
(*interval-times a L* ! *0*) (*interval-times a L* ! *i2* − *1*) [*?s*!*i2* − *?s*!*0*] (*And-mltl-ext* (Not_c α) β))
(*Mighty-Release-mltl-ext* α β (*interval-times a L* ! *i2*)
(*interval-times a L* ! (*i2* + *1*) − *1*)
[*interval-times a L* ! (*i2* + *1*) − *interval-times a L* ! *i2*])
and *i2-bound*: *1* ≤ *i2* ∧ *i2* < *length L*
by *auto*
obtain *j2* **where** *j2-bound*: ((*?s* ! *i2*) ≤ *j2* ∧ *j2* ≤ *?s* ! (*i2* + *1*) − *1*)
and α*-semantics2*: *semantics-mltl* (*drop j2* π) (*to-mltl* α)
using *assms*(*8*) **unfolding** $\psi 2$ *Mighty-Release-mltl-ext.simps semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *blast*
have *lb2*: *a* ≤ *?s*!*i2*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i2 0 ?s*]
unfolding *sfirst* **using** *i2-bound* **by** *simp*
have *welldef2*: *?s*!*i2* < *?s*!(*i2*+*1*)
using *interval-times-diff-ge*[*OF a-leq-b L-composition*, *of i2 ?s*]
using *i2-bound* **by** *blast*
have *ub2*: *interval-times a L* ! (*i2* + *1*) − *1* ≤ *b*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L i2+1 ?s*]
using *slast i2-bound*

by (*metis add.commute diff-diff-left diff-is-0-eq le-neq-implies-less less-iff-succ-less-eq less-or-eq-imp-le*)
have *?thesis* **using** *α-semantics2 global1 j2-bound*
unfolding *sfirst* **using** *lb2 ub2* **by** *simp*
}
ultimately have *?thesis* **using** *assms D-is* **by** *blast*
} **moreover** {
assume ∗: *ψ1* ∈ *?middle*
then have *ψ1*: *ψ1* = *Mighty-Release-mltl-ext α β* (*?s* ! *0*) (*?s* ! *1* − *1*) [*?s* ! *1* − *?s* ! *0*]
by *blast*
obtain *j1* **where** *j1-bound*: (*?s* ! *0* ≤ *j1* ∧ *j1* ≤ *?s* ! *1* − *1*)
and *α-semantics1*: *semantics-mltl* (*drop j1 π*) (*to-mltl α*)
using *assms*(*7*) **unfolding** *ψ1 Mighty-Release-mltl-ext.simps semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *blast*
have *bound1*: *interval-times a L* ! *1* − *1* ≤ *b*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L 1 ?s*]
using *slast length-L* **by** *force*
{
assume ∗∗: *ψ2* ∈ *?front*
then have *ψ2*: *ψ2* = *Global-mltl-ext a b L* (*And-mltl-ext* (*Not*$_c$ *α*) *β*)
by *auto*
have *global2*: (∀ *i*. *a* ≤ *i* ∧ *i* ≤ *b* ⟶
¬ *semantics-mltl* (*drop i π*) (*to-mltl α*) ∧
semantics-mltl (*drop i π*) (*to-mltl β*))
using *assms*(*8*) **unfolding** *ψ2 semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using *α-wpd a-leq-b*
by (*metis add-diff-cancel-left' cancel-comm-monoid-add-class.diff-cancel dual-order.trans le-add1 not-one-le-zero order-antisym-conv wpd-geq-one*)
have *?thesis*
using *global2 α-semantics1 j1-bound* **unfolding** *sfirst* **using** *bound1* **by** *simp*
} **moreover** {
assume ∗∗: *ψ2* ∈ *?middle*
then have *ψ2*: *ψ2* = *Mighty-Release-mltl-ext α β* (*?s* ! *0*) (*?s* ! *1* − *1*) [*?s* ! *1* − *?s* ! *0*]
by *blast*
then have *?thesis* **using** *ψ1 assms* **by** *blast*
} **moreover** {
assume ∗∗: *ψ2* ∈ *?back*
then obtain *i2* **where** *ψ2*: *ψ2* = *And-mltl-ext*
(*Global-mltl-ext*
(*interval-times a L* ! *0*) (*interval-times a L* ! *i2* − *1*) [*?s*!*i2* − *?s*!*0*] (*And-mltl-ext* (*Not*$_c$ *α*) *β*))
(*Mighty-Release-mltl-ext α β* (*interval-times a L* ! *i2*)
(*interval-times a L* ! (*i2* + *1*) − *1*)

[*interval-times a L* ! (*i2* + *1*) − *interval-times a L* ! *i2*])
and *i2-bound*: *1* ≤ *i2* ∧ *i2* < *length L*
by *auto*
obtain *j2* **where** *j2-bound*: ((*?s* ! *i2*) ≤ *j2* ∧ *j2* ≤ *?s* ! (*i2* + *1*) − *1*)
and *α-semantics2*: *semantics-mltl* (*drop j2* π) (*to-mltl* α)
and *global-before2*: ∀ *i*. *interval-times a L* ! *0* ≤ *i* ∧ *i* ≤ *interval-times a L* ! *i2* − *1* ⟶
¬ *semantics-mltl* (*drop i* π) (*to-mltl* α) ∧
semantics-mltl (*drop i* π) (*to-mltl* β)
using *assms*(*8*) **unfolding** *ψ2 Mighty-Release-mltl-ext.simps semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
unfolding *sfirst* **using** *α-wpd* **by** *auto*
have *lb2*: *a* ≤ *?s*!*i2*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i2 0 ?s*]
unfolding *sfirst* **using** *i2-bound* **by** *simp*
have *welldef2*: *?s*!*i2* < *?s*!(*i2*+*1*)
using *interval-times-diff-ge*[*OF a-leq-b L-composition*, *of i2 ?s*]
using *i2-bound* **by** *blast*
have *ub2*: *interval-times a L* ! (*i2* + *1*) − *1* ≤ *b*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L i2+1 ?s*]
using *slast i2-bound*
by (*metis add.commute diff-diff-left diff-is-0-eq le-neq-implies-less less-iff-succ-less-eq less-or-eq-imp-le*)
have *bound1*: *interval-times a L* ! *1* ≤ *interval-times a L* ! *i2*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i2 1 ?s*]
using *i2-bound* **by** *force*
have *?thesis* **using** *global-before2 α-semantics1 bound1*
using *j1-bound* **unfolding** *sfirst* **by** *auto*
}
ultimately have *?thesis* **using** *assms D-is* **by** *blast*
} moreover {
assume ∗: *ψ1* ∈ *?back*
then obtain *i1* **where** *ψ1*: *ψ1* = *And-mltl-ext*
(*Global-mltl-ext*
(*interval-times a L* ! *0*) (*interval-times a L* ! *i1* − *1*) [*?s*!*i1* − *?s*!*0*] (*And-mltl-ext* (*Not*$_c$ α) β))
(*Mighty-Release-mltl-ext* α β (*interval-times a L* ! *i1*)
(*interval-times a L* ! (*i1* + *1*) − *1*)
[*interval-times a L* ! (*i1* + *1*) − *interval-times a L* ! *i1*])
and *i1-bound*: *1* ≤ *i1* ∧ *i1* < *length L*
by *auto*
obtain *j1* **where** *j1-bound*: ((*?s* ! *i1*) ≤ *j1* ∧ *j1* ≤ *?s* ! (*i1* + *1*) − *1*)
and *α-semantics1*: *semantics-mltl* (*drop j1* π) (*to-mltl* α)
and *global-before1*: ∀ *i*. *interval-times a L* ! *0* ≤ *i* ∧ *i* ≤ *interval-times a L* ! *i1* − *1* ⟶
¬ *semantics-mltl* (*drop i* π) (*to-mltl* α) ∧
semantics-mltl (*drop i* π) (*to-mltl* β)
using *assms*(*7*) **unfolding** *ψ1 Mighty-Release-mltl-ext.simps semantics-mltl-ext-def*

to-mltl.simps semantics-mltl.simps
unfolding *sfirst* **using** α*-wpd* **by** *auto*
have *lb1*: $a \leq$ *?s*!*i1*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i1 0 ?s*]
unfolding *sfirst* **using** *i1-bound* **by** *simp*
have *welldef1*: *?s*!*i1* < *?s*!(*i1*+*1*)
using *interval-times-diff-ge*[*OF a-leq-b L-composition*, *of i1 ?s*]
using *i1-bound* **by** *blast*
have *ub1*: *interval-times a L* ! (*i1* + *1*) − *1* ≤ *b*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L i1*+*1 ?s*]
using *slast i1-bound*
by (*metis add.commute diff-diff-left diff-is-0-eq le-neq-implies-less less-iff-succ-less-eq less-or-eq-imp-le*)
have *bound1*: *interval-times a L* ! *1* ≤ *interval-times a L* ! *i1*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i1 1 ?s*]
using *i1-bound* **by** *force*
{
assume ∗: $\psi 2 \in$ *?front*
then have $\psi 2$: $\psi 2$ = *Global-mltl-ext a b L* (*And-mltl-ext* (*Not*$_c$ α) β)
by *auto*
have *global2*: (∀ *i*. *a* ≤ *i* ∧ *i* ≤ *b* ⟶
¬ *semantics-mltl* (*drop i* π) (*to-mltl* α) ∧
semantics-mltl (*drop i* π) (*to-mltl* β))
using *assms*(*8*) **unfolding** $\psi 2$ *semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
using α*-wpd a-leq-b*
by (*metis add-diff-cancel-left*′ *cancel-comm-monoid-add-class.diff-cancel dual-order.trans le-add1 not-one-le-zero order-antisym-conv wpd-geq-one*)
have *?thesis* **using** α*-semantics1 global2 j1-bound*
unfolding *sfirst* **using** *lb1 ub1* **by** *simp*
} moreover {
assume ∗: $\psi 2 \in$ *?middle*
then have $\psi 2$: $\psi 2$ = *Mighty-Release-mltl-ext* α β (*?s* ! *0*)
(*?s* ! *1* − *1*) [*?s* ! *1* − *?s* ! *0*]
by *blast*
obtain *j2* **where** *j2-bound*: (*?s* ! *0* ≤ *j2* ∧ *j2* ≤ *?s* ! *1* − *1*)
and α*-semantics2*: *semantics-mltl* (*drop j2* π) (*to-mltl* α)
using *assms*(*8*) **unfolding** $\psi 2$ *Mighty-Release-mltl-ext.simps semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
by *blast*
have *bound1*: *interval-times a L* ! *1* ≤ *interval-times a L* ! *i1*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i1 1 ?s*]
using *i1-bound* **by** *force*
then have *?thesis*
using α*-semantics2 global-before1*
using *j2-bound* **unfolding** *sfirst* **by** *auto*
} moreover {
assume ∗: $\psi 2 \in$ *?back*

then obtain *i2* **where** $\psi 2$: $\psi 2$ = *And-mltl-ext*
(*Global-mltl-ext*
(*interval-times a L ! 0*) (*interval-times a L ! i2 − 1*) [*?s*!*i2 −
?s*!*0*] (*And-mltl-ext* (*Not*$_c$ α) β))
(*Mighty-Release-mltl-ext* α β (*interval-times a L ! i2*)
(*interval-times a L ! (i2 + 1) − 1*)
[*interval-times a L ! (i2 + 1) − interval-times a L ! i2*])
and *i2-bound*: $1 \leq i2 \wedge i2 <$ *length L*
by *auto*
obtain *j2* **where** *j2-bound*: ((*?s ! i2*) $\leq$ *j2* $\wedge$ *j2* $\leq$ *?s ! (i2 + 1) − 1*)
and α-*semantics2*: *semantics-mltl* (*drop j2* π) (*to-mltl* α)
and *global-before2*: $\forall$ *i. interval-times a L ! 0* $\leq i \wedge i \leq$ *interval-times
a L ! i2 − 1* $\longrightarrow$
$\neg$ *semantics-mltl* (*drop i* π) (*to-mltl* α) $\wedge$
semantics-mltl (*drop i* π) (*to-mltl* β)
using *assms*(*8*) **unfolding** $\psi 2$ *Mighty-Release-mltl-ext.simps semantics-mltl-ext-def to-mltl.simps semantics-mltl.simps*
unfolding *sfirst* **using** α-*wpd* **by** *auto*
have *lb2*: *a* $\leq$ *?s*!*i2*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i2 0 ?s*]
unfolding *sfirst* **using** *i2-bound* **by** *simp*
have *welldef2*: *?s*!*i2 < ?s*!(*i2+1*)
using *interval-times-diff-ge*[*OF a-leq-b L-composition*, *of i2 ?s*]
using *i2-bound* **by** *blast*
have *ub2*: *interval-times a L ! (i2 + 1) − 1* $\leq$ *b*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of length L
i2+1 ?s*]
using *slast i2-bound*
by (*metis add.commute diff-diff-left diff-is-0-eq le-neq-implies-less less-iff-succ-less-eq
less-or-eq-imp-le*)
{
assume *eq*: *i1 = i2*
then have *?thesis*
using *assms*(*6*) $\psi 1$ $\psi 2$ **by** *blast*
} moreover {
assume *le*: *i1 < i2*
then have *interval-times a L ! (i1 + 1)* $\leq$ *interval-times a L ! (i2)*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i2 i1+1
?s*]
using *i1-bound i2-bound*
by (*metis le-eq-less-or-eq less-iff-succ-less-eq*)
then have *?thesis*
using α-*semantics1 global-before2 j1-bound*
using *lb1* **unfolding** *sfirst* **by** *auto*
} moreover {
assume *ge*: *i1 > i2*
then have *interval-times a L ! (i2 + 1)* $\leq$ *interval-times a L ! (i1)*
using *interval-times-diff-ge-general*[*OF a-leq-b L-composition*, *of i1 i2+1
?s*]

```
      using i1-bound i2-bound
      by (metis le-eq-less-or-eq less-iff-succ-less-eq)
    then have ?thesis
      using α-semantics2 global-before1 j2-bound
      using lb2 unfolding sfirst by auto
   }
   ultimately have ?thesis by linarith
  }
  ultimately have ?thesis using assms D-is by blast
 }
 ultimately show ?thesis using assms D-is by blast
qed

lemma LP-mltl-language-disjoint-aux-k1:
 fixes φ::'a mltl-ext and ψ1 ψ2::'a mltl-ext and k::nat
 assumes intervals-welldef: intervals-welldef (to-mltl φ)
 assumes is-nnf: ∃φ-init. φ = convert-nnf-ext φ-init
 assumes composition: is-composition-MLTL φ
 assumes D-decomp: D = set (LP-mltl-aux φ 1)
 assumes diff-formulas: (ψ1 ∈ D) ∧ (ψ2 ∈ D) ∧ ψ1 ≠ ψ2
 assumes r-wpd: r ≥ wpd-mltl (to-mltl φ)
 shows (language-mltl-r (to-mltl ψ1) r)
     ∩ (language-mltl-r (to-mltl ψ2) r) = {}
proof−
 {
  assume contra: (language-mltl-r (to-mltl ψ1) r)
    ∩ (language-mltl-r (to-mltl ψ2) r) ≠ {}
  then have ∃π. π ∈ (language-mltl-r (to-mltl ψ1) r) ∧
                π ∈ (language-mltl-r (to-mltl ψ2) r)
    by auto
  then obtain π where in1: π ∈ (language-mltl-r (to-mltl ψ1) r)
          and in2: π ∈ (language-mltl-r (to-mltl ψ2) r)
    by blast
  have sem1: semantics-mltl-ext π ψ1 and
       sem2: semantics-mltl-ext π ψ2 and
       len: length π ≥ wpd-mltl (to-mltl φ)
    using in1 in2 assms(6)
    unfolding language-mltl-r-def semantics-mltl-ext-def
     by simp-all
  have False
   by (metis D-decomp LP-mltl-language-disjoint-aux-helper-k1 One-nat-def com-
position diff-formulas intervals-welldef is-nnf len sem1 sem2)
 }
 then show ?thesis by blast
qed



theorem LP-mltl-language-disjoint-k1:
```

fixes φ::'*a mltl-ext* **and** $\psi 1$ $\psi 2$::'*a mltl* **and** *k*::*nat*
assumes *intervals-welldef*: *intervals-welldef* (*to-mltl* φ)
assumes *composition*: *is-composition-MLTL* φ
assumes *D-decomp*: D = *set* (*LP-mltl* φ *1*)
assumes *diff-formulas*: ($\psi 1 \in D$) $\wedge$ ($\psi 2 \in D$) $\wedge$ $\psi 1 \neq \psi 2$
assumes *r-wpd*: $r \geq$ *wpd-mltl* (*to-mltl* φ)
shows (*language-mltl-r* $\psi 1$ r) $\cap$ (*language-mltl-r* $\psi 2$ r) = {}
proof−
let *?D* = *LP-mltl-aux* (*convert-nnf-ext* φ) *1*
let *?*φ = *convert-nnf-ext* φ
have *cond1*: *intervals-welldef* (*to-mltl* (*convert-nnf-ext* φ))
using *intervals-welldef*
by (*metis convert-nnf-ext-to-mltl-commute nnf-intervals-welldef*)
have *cond2*: $\exists \varphi$*-init*. *convert-nnf-ext* φ = *convert-nnf-ext* φ*-init*
by *blast*
have *cond3*: *is-composition-MLTL* (*convert-nnf-ext* φ)
using *composition*
by (*simp add*: *intervals-welldef is-composition-convert-nnf-ext*)
have *cond4*: *set* (*LP-mltl-aux* (*convert-nnf-ext* φ) *1*) =
set (*LP-mltl-aux* (*convert-nnf-ext* φ) *1*)
by *blast*
obtain $\psi 1'$ $\psi 2'$ **where** $\psi 1$: $\psi 1$ = *to-mltl* (*convert-nnf-ext* $\psi 1'$)
and $\psi 1'$*-in*: $\psi 1' \in$ *set ?D*
and $\psi 2$: $\psi 2$ = *to-mltl* (*convert-nnf-ext* $\psi 2'$)
and $\psi 2'$*-in*: $\psi 2' \in$ *set ?D*
using *D-decomp* **unfolding** *LP-mltl.simps*
using *diff-formulas* **by** *auto*
have ψ'*s-neq*: $\psi 1' \neq \psi 2'$
using *diff-formulas* $\psi 1$ $\psi 2$ **by** *blast*
have $\psi 1$*-welldef*: *intervals-welldef* $\psi 1$
using *assms*(*4*) *D-decomp* **unfolding** *LP-mltl.simps*
using *LP-mltl-aux-intervals-welldef*
by (*metis* $\psi 1$ $\psi 1'$*-in composition convert-nnf-ext-to-mltl-commute intervals-welldef nnf-intervals-welldef*)
then have $\psi 1'$*-welldef*: *intervals-welldef* (*to-mltl* $\psi 1'$)
using $\psi 1$
using *LP-mltl-aux-intervals-welldef* $\psi 1'$*-in allones-implies-is-composition-MLTL composition intervals-welldef* **by** *auto*
have $\psi 2$*-welldef*: *intervals-welldef* $\psi 2$
using *assms*(*4*) *D-decomp* **unfolding** *LP-mltl.simps*
using *LP-mltl-aux-intervals-welldef*
by (*metis* $\psi 2$ $\psi 2'$*-in composition convert-nnf-ext-to-mltl-commute intervals-welldef nnf-intervals-welldef*)
then have $\psi 2'$*-welldef*: *intervals-welldef* (*to-mltl* $\psi 2'$)
using $\psi 2$
using *LP-mltl-aux-intervals-welldef* $\psi 2'$*-in allones-implies-is-composition-MLTL composition intervals-welldef* **by** *auto*
have *intersect*: *language-mltl-r* (*to-mltl* $\psi 1'$) r $\cap$
language-mltl-r (*to-mltl* $\psi 2'$) r = {}

using *LP-mltl-language-disjoint-aux-k1*[*OF cond1 cond2 cond3 cond4*, *of* $\psi 1'$ $\psi 2'$ *r*]
using $\psi 1'$*-in* $\psi 2'$*-in* ψ'*s-neq r-wpd*
by (*metis convert-nnf-ext-preserves-wpd*)
have *semantics-mltl* π (*to-mltl* (*convert-nnf-ext* φ)) =
semantics-mltl π (*to-mltl* φ)
if *intervals-welldef* (*to-mltl* φ)
for φ::'*a mltl-ext* **and** π
using *that* **unfolding** *semantic-equiv-ext-def*
by (*metis convert-nnf-ext-to-mltl-commute convert-nnf-preserves-semantics*)
then show *?thesis* **using** *intersect*
unfolding *language-mltl-r-def* $\psi 1$ $\psi 2$
using $\psi 1'$*-welldef* $\psi 2'$*-welldef*
by *auto*
qed

end
theory *MLTL-Language-Partition-Codegen*

imports *MLTL-Language-Partition-Algorithm Show.Shows-Literal*

begin

9 Pretty Parsing

fun *nat-to-string*:: *nat* ⇒ *string* **where**
nat-to-string n = *String.explode* (*Shows-Literal.showl n*)

fun *mltl-to-literal-aux*:: *nat mltl* ⇒ *string* **where**
mltl-to-literal-aux True$_m$ = ''*true*''
| *mltl-to-literal-aux False*$_m$ = ''*false*''
| *mltl-to-literal-aux* (*Prop*$_m$ (*p*)) = ''*p*''@(*nat-to-string p*)
| *mltl-to-literal-aux* (*Not*$_m$ φ) = ''(!''@(*mltl-to-literal-aux* φ)@'')''
| *mltl-to-literal-aux* (φ *And*$_m$ ψ) = ''('' @ (*mltl-to-literal-aux* φ) @ '' & '' @ (*mltl-to-literal-aux* ψ) @ '')''
| *mltl-to-literal-aux* (φ *Or*$_m$ ψ) = ''('' @ (*mltl-to-literal-aux* φ) @ '' | '' @ (*mltl-to-literal-aux* ψ) @ '')''
| *mltl-to-literal-aux* (*G*$_m$ [*a*,*b*] φ) = ''(G['' @ (*nat-to-string a*) @ '','' @ (*nat-to-string b*) @ '']'' @ (*mltl-to-literal-aux* φ) @ '')''
| *mltl-to-literal-aux* (*F*$_m$ [*a*,*b*] φ) = ''(F['' @ (*nat-to-string a*) @ '','' @ (*nat-to-string b*) @ '']'' @ (*mltl-to-literal-aux* φ) @ '')''
| *mltl-to-literal-aux* (φ *R*$_m$ [*a*,*b*] ψ) = ''('' @ (*mltl-to-literal-aux* φ) @ '' R['' @ (*nat-to-string a*) @ '','' @ (*nat-to-string b*) @ '']'' @ (*mltl-to-literal-aux* ψ) @ '')''
| *mltl-to-literal-aux* (φ *U*$_m$ [*a*,*b*] ψ) = ''('' @ (*mltl-to-literal-aux* φ) @ '' U['' @ (*nat-to-string a*) @ '','' @ (*nat-to-string b*) @ '']'' @ (*mltl-to-literal-aux* ψ) @ '')''

fun *mltl-to-literal*:: *nat mltl* ⇒ *String.literal*
where *mltl-to-literal* φ = *String.implode* (*mltl-to-literal-aux* φ)

value *mltl-to-literal* $((Prop_m\ (3)\ And_m\ True_m)\ U_m[3,4]\ False_m) =$
STR ''((p3 & true) U[3,4] false)''

10 Code Export

export-code *LP-mltl mltl-to-literal* **in** *Haskell* **module-name** *LP-mltl*

end

References

[1] K. Kosaian, Z. Wang, and E. Sloan. Mission-time linear temporal logic. *Archive of Formal Proofs*, January 2025. https://isa-afp.org/entries/Mission_Time_LTL.html, Formal proof development.