Formalizing MLTL in Isabelle/HOL

Zili Wang and Katherine Kosaian and Alec Rosentrater

March 4, 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* $\Rightarrow$ *'a mltl-ext* $\Rightarrow$ *bool*
(- $\models_c$ - [*80*,*80*] *80*)
where $\pi \models_c \varphi = \pi \models_m$ (*to-mltl* φ)

definition *semantic-equiv-ext*:: *'a mltl-ext* $\Rightarrow$ *'a mltl-ext* $\Rightarrow$ *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* $\Rightarrow$ *nat* $\Rightarrow$ *'a set list set*
where *language-mltl-r* φ *r* =
$\{\pi$. *semantics-mltl* π φ $\wedge$ *length* $\pi \geq r\}$

fun *convert-nnf-ext*:: *'a mltl-ext* $\Rightarrow$ *'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*] $\Rightarrow$ *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*] $\Rightarrow$ *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*] $\Rightarrow$ *bool* **where**
is-composition n L = (($\forall$ *i*. *List.member L i* $\longrightarrow$ *i* > *0*) $\wedge$ (*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* $\Rightarrow$ *bool* **where**
is-composition-MLTL (φ And_c ψ) = ((*is-composition-MLTL* φ) $\wedge$ (*is-composition-MLTL* ψ))
| *is-composition-MLTL* (φ Or_c ψ) = ((*is-composition-MLTL* φ) $\wedge$ (*is-composition-MLTL* ψ))
| *is-composition-MLTL* (G_c[*a*,*b*] <*L*> φ) = ((*is-composition* (*b*−*a*+*1*) *L*) $\wedge$ (*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*) $\wedge$ (*is-composition-MLTL* φ))
| *is-composition-MLTL* (φ U_c[*a*,*b*] <*L*> ψ) = ((*is-composition* (*b*−*a*+*1*) *L*) $\wedge$ (*is-composition-MLTL* φ) $\wedge$ (*is-composition-MLTL* ψ))
| *is-composition-MLTL* (φ R_c[*a*,*b*] <*L*> ψ) = ((*is-composition* (*b*−*a*+*1*) *L*) $\wedge$ (*is-composition-MLTL* φ) $\wedge$ (*is-composition-MLTL* ψ))
| *is-composition-MLTL* - = *True*

definition *is-composition-allones*:: *nat* $\Rightarrow$ *nat list* $\Rightarrow$ *bool* **where**
is-composition-allones n L = ((*is-composition n L*) $\wedge$ ($\forall$ *i*<*length L*. *L*!*i* = *1*))

fun *is-composition-MLTL-allones*:: *'a mltl-ext* $\Rightarrow$ *bool* **where**
is-composition-MLTL-allones (φ And_c ψ) = ((*is-composition-MLTL-allones* φ) $\wedge$ (*is-composition-MLTL-allones* ψ))
| *is-composition-MLTL-allones* (φ Or_c ψ) = ((*is-composition-MLTL-allones* φ) $\wedge$ (*is-composition-MLTL-allones* ψ))
| *is-composition-MLTL-allones* (G_c[*a*,*b*] <*L*> φ) = ((*is-composition-allones* (*b*−*a*+*1*) *L*) $\wedge$ *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*) $\wedge$ (*is-composition-MLTL-allones* φ))
| *is-composition-MLTL-allones* (φ U_c[*a*,*b*] <*L*> ψ) = ((*is-composition-allones* (*b*−*a*+*1*) *L*) $\wedge$ (*is-composition-MLTL-allones* φ) $\wedge$ (*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 = [φ]
| 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 φ] D-ψ) @
   (And-mltl-list D-φ [(Not_c ψ)])))
| 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-φ ≤ 1) then ([G_c[a,b] <L> φ])
                        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
    (λi. (And-mltl-list [Global-mltl-ext (s!0) ((s!i)−1) [s!i − s!0] (Not_c φ)]
    (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
    (λi. (And-mltl-list [Global-mltl-ext (s!0) ((s!i)−1) [s!i − s!0] (And-mltl-ext φ
(Not_c ψ))]
                 (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*
(λ*i*. (*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* (λ*x*. *to-mltl x*)
(*map* (λ*x*. *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*⟩

lemma *convert-nnf-ext-to-mltl-commute*:
shows (*convert-nnf* (*to-mltl* φ)) = (*to-mltl* (*convert-nnf-ext* φ))
⟨*proof*⟩

lemma *convert-nnf-ext-preserves-semantics*:
assumes *intervals-welldef* (*to-mltl* φ)
shows (*convert-nnf-ext* φ) $\equiv_c$ φ
⟨*proof*⟩

lemma *convert-nnf-ext-convert-nnf-ext*:
shows *convert-nnf-ext* φ = *convert-nnf-ext* (*convert-nnf-ext* φ)
⟨*proof*⟩

4.1 Cases where to mltl is bijective

lemma *to-mltl-true-bijective*:
assumes *to-mltl* $\varphi = True_m$
shows $\varphi = True_c$
⟨*proof*⟩

lemma *to-mltl-false-bijective*:
assumes *to-mltl* $\varphi = False_m$
shows $\varphi = False_c$
⟨*proof*⟩

lemma *to-mltl-prop-bijective*:
assumes *to-mltl* $\varphi = Prop_m$ (p)

shows $\varphi = Prop_c\ (p)$
⟨*proof*⟩

lemma *to-mltl-not-prop-bijective*:
assumes *to-mltl* $\varphi = Not_m\ (Prop_m\ (p))$
shows $\varphi = Not_c\ (Prop_c\ (p))$
⟨*proof*⟩

5 Lemmas about Integer Composition

lemma *composition-length-ub*:
fixes *n*::*nat* **and** *L*::*nat list*
assumes *is-composition n L*
shows *length* $L \leq n$
⟨*proof*⟩

lemma *composition-length-lb*:
fixes *n*::*nat* **and** *L*::*nat list*
assumes *is-composition n L*
assumes $n > 0$
shows $0 <$ *length L*
⟨*proof*⟩

lemma *interval-times-length*:
fixes *a*::*nat* **and** *L*::*nat list*
shows *length* (*interval-times a L*) = *length L* + *1*
⟨*proof*⟩

lemma *interval-times-first*:
fixes *a*::*nat* **and** *L*::*nat list*
shows (*interval-times a L*)!*0* = *a*
⟨*proof*⟩

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*⟩

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*⟩

lemma *interval-times-diff-ge*:
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 $s!(i+1) > s!(i)$
⟨*proof*⟩

lemma *interval-times-diff-ge-general*:
fixes $a\ b\ i\ j{::}nat$ **and** $L{::}nat\ list$
assumes *int-welldef*: $a \leq b$
assumes *composition*: *is-composition* $(b-a+1)\ L$
assumes *j-index*: $j \leq$ *length* L
assumes *i-le-j*: $i < j$
assumes *s-is*: $s =$ *interval-times* $a\ L$
shows $s!j > s!i$
⟨*proof*⟩

lemma *trivial-composition*:
assumes $n > 0$
shows *is-composition* $n\ [n]$
⟨*proof*⟩

lemma *sum-list-pos*: $(\bigwedge x.\ x \in$ *set* $(xs{::}nat\ list) \Longrightarrow 0 < x)$
$\Longrightarrow$ *length* $xs > 0 \Longrightarrow 0 <$ *sum-list* xs
⟨*proof*⟩

lemma *take-prefix*:
assumes $L = H@[t]$
assumes $k \leq$ *length* $L - 1$
shows *take* $k\ H =$ *take* $k\ L$
⟨*proof*⟩

lemma *take-interval-times*:
assumes *length* $L \geq k$
shows *take* $(k+1)$ (*interval-times* $a\ L$) $=$ *interval-times* a (*take* $k\ L$)
⟨*proof*⟩

lemma *index-list-index*:
fixes $k{::}nat$
assumes $j < k$
shows $[0\ ..{<}\ k]\ !\ j = j$
⟨*proof*⟩

lemma *interval-times-obtain-aux*:
assumes $a \leq b$
assumes *is-composition* $(b - a + 1)$ L
assumes $s =$ *interval-times* a L
assumes $(s \,!\, 1) \leq t \wedge t \leq b$
shows $\exists i.\ s \,!\, i \leq t \wedge t \leq s \,!\, (i + 1) - 1 \wedge 1 \leq i \wedge i <$ *length* L
⟨*proof*⟩

lemma *interval-times-obtain*:
assumes $a \leq b$
assumes *is-composition* $(b - a + 1)$ L
assumes $s =$ *interval-times* a L
assumes $a \leq t \wedge t \leq b$
shows $\exists i.\ s \,!\, i \leq t \wedge t \leq s \,!\, (i + 1) - 1 \wedge 0 \leq i \wedge i <$ *length* L
⟨*proof*⟩

lemma *list-allones*:
assumes $\forall i<$*length* $L.\ L!i = 1$
shows $L =$ *map* $(\lambda i.\ 1)$ $[0 ..<$ *length* $L]$
⟨*proof*⟩

lemma *sum-list-constants*:
fixes L::*nat list* **and** k::*nat*
assumes $\forall i<$*length* $L.\ L \,!\, i = k$
shows *sum-list* $L = k*($*length* $L)$
⟨*proof*⟩

lemma *length-is-composition-allones*:
assumes *is-composition-allones* n L
shows *length* $L = n$
⟨*proof*⟩

lemma *partial-sum-allones*:
assumes $(\forall i<$*length* $L.\ L \,!\, i = 1)$
assumes $i \leq$ *length* L
shows *partial-sum* L $i = i$
⟨*proof*⟩

lemma *interval-times-allones*:
assumes $a \leq b$
assumes *is-composition-allones* $(b - a + 1)$ L
assumes $i <$ *length* (*interval-times* a L)
shows (*interval-times* a L)$!i = a+i$
⟨*proof*⟩

lemma *allones-implies-is-composition*:

assumes *is-composition-allones n L*
shows *is-composition n L*
⟨*proof*⟩

lemma *allones-implies-is-composition-MLTL*:
assumes *is-composition-MLTL-allones* φ
shows *is-composition-MLTL* φ
⟨*proof*⟩

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*⟩

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*))
⟨*proof*⟩

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*⟩

lemma *list-concat-set-union*:
shows *set*(*A*@*B*) = *set A* $\cup$ *set B*
⟨*proof*⟩

lemma *pairs-empty-list*:
shows *pairs A* [] = []
⟨*proof*⟩

7.1 Forward Direction Proofs

lemma *pairs-member-fst-forward*:
assumes *List.member* (*pairs A B*) *x*
shows *List.member A* (*fst x*)
⟨*proof*⟩

lemma *pairs-member-snd-forward*:

assumes *List.member* (*pairs A B*) *x*
shows *List.member B* (*snd x*)
⟨*proof*⟩

lemma *pairs-member-forward*:
assumes *List.member* (*pairs A B*) *x*
shows *List.member A* (*fst x*) ∧ *List.member B* (*snd x*)
⟨*proof*⟩

lemma *And-mltl-list-member-forward*:
assumes *List.member* (*And-mltl-list D-x D-y*) ψ
shows $\exists \psi 1\ \psi 2.\ \psi = \textit{And-mltl-ext}\ \psi 1\ \psi 2$
$\wedge$ *List.member D-x* $\psi 1$ $\wedge$ *List.member D-y* $\psi 2$
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *And-mltl-list-member-converse*:
assumes $\exists \psi 1\ \psi 2.\ \psi = \textit{And-mltl-ext}\ \psi 1\ \psi 2$
$\wedge$ *List.member D-x* $\psi 1$ $\wedge$ *List.member D-y* $\psi 2$
shows *List.member* (*And-mltl-list D-x D-y*) ψ
⟨*proof*⟩

7.3 Biconditional Lemmas

lemma *pairs-member*:
shows (*List.member A* (*fst x*) ∧ *List.member B* (*snd x*)) $\longleftrightarrow$
List.member (*pairs A B*) *x*
⟨*proof*⟩

lemma *And-mltl-list-member*:
shows ($\exists \psi 1\ \psi 2.\ \psi = \textit{And-mltl-ext}\ \psi 1\ \psi 2$
$\wedge$ *List.member D-x* $\psi 1$ $\wedge$ *List.member D-y* $\psi 2$) $\longleftrightarrow$
List.member (*And-mltl-list D-x D-y*) ψ
⟨*proof*⟩

8 MLTL Decomposition Top Level Correctness

fun *wpd-mltl*:: *'a mltl* ⇒ *nat*
where *wpd-mltl* $\textit{False}_m$ = *1*
| *wpd-mltl* $\textit{True}_m$ = *1*
| *wpd-mltl* ($\textit{Prop}_m$ (*p*)) = *1*

| *wpd-mltl* (Not_m φ) = *wpd-mltl* φ
| *wpd-mltl* (φ And_m ψ) = *max* (*wpd-mltl* φ) (*wpd-mltl* ψ)
| *wpd-mltl* (φ Or_m ψ) = *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* (φ R_m $[a,b]$ ψ) = b + (*max* ((*wpd-mltl* φ)) (*wpd-mltl* ψ))
| *wpd-mltl* (φ U_m $[a,b]$ ψ) = b + (*max* ((*wpd-mltl* φ)) (*wpd-mltl* ψ))

8.1 Helper Lemmas

lemma *wpd-geq-one*:
shows *wpd-mltl* $\varphi \geq 1$
⟨*proof*⟩

lemma *wpd-convert-nnf*:
fixes φ::*'a mltl*
shows *wpd-mltl* (*convert-nnf* φ) = *wpd-mltl* φ
⟨*proof*⟩

lemma *convert-nnf-ext-preserves-wpd*:
shows *wpd-mltl* (*to-mltl* (*convert-nnf-ext* φ)) =
wpd-mltl (*to-mltl* φ)
⟨*proof*⟩

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

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)
⟨*proof*⟩

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)
⟨*proof*⟩

function *Ands-mltl-ext*:: *'a mltl-ext list* ⇒ *'a mltl-ext*
where *Ands-mltl-ext* [] = *True-mltl-ext*

| *Ands-mltl-ext* $(H@[t]) = (if\ (length\ H = 0)\ then\ t$
  else $(And\text{-}mltl\text{-}ext\ (Ands\text{-}mltl\text{-}ext\ H)\ t))$
⟨*proof*⟩
termination ⟨*proof*⟩

lemma *Ands-mltl-semantics*:
assumes $length\ X \geq 1$
shows $semantics\text{-}mltl\text{-}ext\ \pi\ (Ands\text{-}mltl\text{-}ext\ X) \longleftrightarrow$
$(\forall x \in set\ X.\ semantics\text{-}mltl\text{-}ext\ \pi\ x)$
⟨*proof*⟩

lemma *in-Global-mltl-decomp*:
assumes $length\ D\text{-}\varphi > 1$
assumes $\psi \in set\ (Global\text{-}mltl\text{-}decomp\ D\text{-}\varphi\ a\ n\ L)$
shows $\exists X.\ ((\psi = Ands\text{-}mltl\text{-}ext\ X \land$
$(\forall x.\ List.member\ X\ x \longrightarrow$
$(\exists y \in set\ D\text{-}\varphi.\ (\exists k.\ a \leq k \land k \leq (a+n) \land x = Global\text{-}mltl\text{-}ext\ k\ k\ [1]$
$y)))) \land$
$(length\ X = Suc\ n))$
⟨*proof*⟩

lemma *in-Global-mltl-decomp-exact-forward*:
assumes $length\ D\text{-}\varphi > 1$
assumes $\psi \in set\ (Global\text{-}mltl\text{-}decomp\ D\text{-}\varphi\ a\ n\ L)$
shows $\exists X.\ ((\psi = Ands\text{-}mltl\text{-}ext\ X \land$
$(\forall i < length\ X.\ (\exists y \in set\ D\text{-}\varphi.\ (X!i) = Global\text{-}mltl\text{-}ext\ (a+i)\ (a+i)\ [1]$
$y)))) \land$
$(length\ X = Suc\ n)$
⟨*proof*⟩

lemma *in-Global-mltl-decomp-exact-converse*:
fixes $n::nat$ **and** $X::'a\ mltl\text{-}ext\ list$
assumes $length\ D\text{-}\varphi > 1$
assumes $\psi = Ands\text{-}mltl\text{-}ext\ X$
assumes $(\forall i < length\ X.\ (\exists y \in set\ D\text{-}\varphi.$
$(X!i) = Global\text{-}mltl\text{-}ext\ (a+i)\ (a+i)\ [1]\ y))$
assumes $length\ X = n+1$
shows $\psi \in set\ (Global\text{-}mltl\text{-}decomp\ D\text{-}\varphi\ a\ n\ L)$
⟨*proof*⟩

lemma *case-split-helper*:
assumes $x \in A \cup B \cup C$
assumes $x \in A \Longrightarrow P\ x$ **and** $x \in B \Longrightarrow P\ x$ **and** $x \in C \Longrightarrow P\ x$
shows $P\ x$
⟨*proof*⟩

lemma *LP-mltl-aux-intervals-welldef*:

fixes φ ψ::'*a mltl-ext*
assumes *intervals-welldef* (*to-mltl* φ)
assumes $\psi \in$ *set* (*LP-mltl-aux* (*convert-nnf-ext* φ) *k*)
assumes *is-composition-MLTL* φ
shows *intervals-welldef* (*to-mltl* ψ)
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *And-mltl-list-nonempty*:
assumes $A \neq []$ **and** $B \neq []$
shows *And-mltl-list* A $B \neq []$
⟨*proof*⟩

lemma *Global-mltl-decomp-nonempty*:
assumes $D \neq []$
shows *Global-mltl-decomp* D a n $L \neq []$
⟨*proof*⟩

lemma *LP-mltl-aux-nonempty*:
assumes $\exists\,\varphi$-*init*. φ = *convert-nnf-ext* φ-*init*
assumes *intervals-welldef* (*to-mltl* φ)
assumes *is-composition-MLTL* φ
shows *LP-mltl-aux* φ $k \neq []$
⟨*proof*⟩

8.2 Union Theorem

Forward Direction **lemma** *exist-first*:
fixes *lb* *i*::*nat*
assumes *lowerbound*: $lb \leq i$ **and** *iprop*: $(P\ i)$
shows $\exists j.\ (lb \leq j \land j \leq i \land (P\ j)$
$\land (\forall l.\ (lb \leq l \land l < j) \longrightarrow \neg(P\ l)))$
⟨*proof*⟩

lemma *exist-bound-split*:
fixes *a m b*::*nat*
assumes $a \leq b$
assumes $\exists i.\ a \leq i \land i \leq b \land P\ i$
shows $(\exists i.\ a \leq i \land i \leq m{-}1 \land P\ i) \lor$
$(\exists i.\ m \leq i \land i \leq b \land P\ i \land \neg(\exists j.\ a \leq j \land j < m \land P\ j))$

⟨*proof*⟩

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* α)
assumes *ih*: $\bigwedge$*trace. semantics-mltl-ext trace* $\alpha \Longrightarrow$
wpd-mltl (*to-mltl* α) $\leq$ *length trace* $\Longrightarrow$
$\exists$ *x*∈*set D. semantics-mltl-ext trace x*
shows $\exists$ *X*. (*length X* = *b*−*a*+*1*) ∧
(∀ *i*<*length X*. (*X*!*i* ∈ *set D*) ∧ *semantics-mltl-ext* (*drop* (*a*+*i*) π) (*X*!*i*))
⟨*proof*⟩

lemma *Release-semantics-split*:
assumes (∀ *i*. $a \leq i \wedge i \leq b \longrightarrow$ *semantics-mltl* (*drop i* π) (*to-mltl* β)) ∨
(∃ *j*≥*a*. $j \leq b - 1$ ∧ *semantics-mltl* (*drop j* π) (*to-mltl* α) ∧
(∀ *k*. $a \leq k \wedge k \leq j \longrightarrow$
semantics-mltl (*drop k* π) (*to-mltl* β)))
shows ((∀ *i*. $a \leq i \wedge i \leq b \longrightarrow$ *semantics-mltl* (*drop i* π) (*to-mltl* β))
∧(∀ *i*. $a \leq i \wedge i \leq b \longrightarrow$ *semantics-mltl* (*drop i* π) (Not_m (*to-mltl* α))))
∨ (∃ *j*≥*a*. $j \leq b$ ∧
semantics-mltl (*drop j* π) (*to-mltl* α) ∧
(∀ *k*. $a \leq k \wedge k \leq j \longrightarrow$
semantics-mltl (*drop k* π) (*to-mltl* β)))
⟨*proof*⟩

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* $\pi \geq$ *wpd-mltl* (*to-mltl* φ)
shows ∃ ψ ∈ *set D*. *semantics-mltl-ext* π ψ
⟨*proof*⟩

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* $\pi \geq$ *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
⟨*proof*⟩

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*: $\exists\,\varphi$*-init*. φ = *convert-nnf-ext* φ*-init*
assumes *trace-length*: *length* $\pi \geq$ *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)
⟨*proof*⟩

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* $\pi \geq$ *wpd-mltl* (*to-mltl* φ)
shows *semantics-mltl-ext* π φ $\longleftrightarrow$ ($\exists\,\psi\in$*D*. *semantics-mltl* π ψ)
⟨*proof*⟩

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*
= ($\bigcup$ $\psi\in$*D*. *language-mltl-r* ψ *r*)
⟨*proof*⟩

8.3 Disjointedness Theorem

lemma *LP-mltl-language-disjoint-aux-helper*:
fixes φ $\psi 1$ $\psi 2$::′*a mltl-ext* **and** *k*::*nat* **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-allones*: *is-composition-MLTL-allones* φ
assumes *tracelen*: *length* $\pi \geq$ *wpd-mltl* (*to-mltl* φ)
assumes *D-decomp*: *D* = *set* (*LP-mltl-aux* φ *k*)
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*⟩

lemma *LP-mltl-language-disjoint-aux*:

fixes φ::'a *mltl-ext* **and** $\psi 1$ $\psi 2$::'a *mltl-ext* **and** k::*nat*
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-allones* φ
assumes *D-decomp*: D = *set* (*LP-mltl-aux* φ k)
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* (*to-mltl* $\psi 1$) r)
$\cap$ (*language-mltl-r* (*to-mltl* $\psi 2$) r) = {}
⟨*proof*⟩

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*: ($\psi 1 \in D$) $\wedge$ ($\psi 2 \in D$) $\wedge$ $\psi 1 \neq \psi 2$
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*⟩

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*⟩

lemma *LP-mltl-language-disjoint-aux-k1*:
fixes φ::'a *mltl-ext* **and** $\psi 1$ $\psi 2$::'a *mltl-ext* **and** k::*nat*
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 *D-decomp*: D = *set* (*LP-mltl-aux* 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* (*to-mltl* $\psi 1$) r)
$\cap$ (*language-mltl-r* (*to-mltl* $\psi 2$) r) = {}
⟨*proof*⟩

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*⟩

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.