Formalizing MLTL in Isabelle/HOL

Zili Wang and Elizabeth Sloan and Katherine Kosaian

January 28, 2025

Abstract

We formalize the syntax, semantics, and some useful properties of Mission-time Linear Temporal Logic (MLTL) [4, 3], following [2, 1]. MLTL is a variant of Linear Temporal Logic, which has already been formalized in Isabelle/HOL [6]. In contrast to LTL, MLTL includes finite discrete time bounds on the temporal operators. We do not directly build on the LTL entry, but aim to mirror its style; in particular, we found it useful when defining our syntactic sugar binding precedences. Another closely related AFP entry is [5].

Contents

1 MLTL Encoding

theory *MLTL-Encoding*

imports *Main*

begin

1.1 Syntax

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

definition *Implies-mltl* (- *Implies*$_m$ - [*81*, *81*] *80*)
  where φ *Implies*$_m$ $\psi \equiv$ *Not*$_m$ φ *Or*$_m$ ψ

definition *Iff-mltl* (- *Iff*$_m$ - [*81*, *81*] *80*)
  where φ *Iff*$_m$ $\psi \equiv (\varphi$ *Implies*$_m$ $\psi)$ *And*$_m$ $(\psi$ *Implies*$_m$ $\varphi)$

1.1.1 Binding Examples

value *Not*$_m$ *Prop*$_m$ (*p*) *And*$_m$ *Prop*$_m$ (*q*) =
  And-mltl (*Not-mltl* (*Prop-mltl p*)) (*Prop-mltl q*)

value *p And*$_m$ *q Or*$_m$ *r* = *Or-mltl* (*And-mltl p q*) *r*

value *F*$_m$ [*0*, *1*] *p And*$_m$ *q* = *And-mltl* (*Future-mltl 0 1 p*) *q*

value *p U*$_m$ [*0*,*1*] *q And*$_m$ *r* = *And-mltl* (*Until-mltl p 0 1 q*) *r*

1.2 Semantics

primrec *semantics-mltl* :: [*'a set list*, *'a mltl*] ⇒ *bool* (- ⊨$_m$ - [*80*,*80*] *80*)
where
  $\pi \models_m$ *True*$_m$ = *True*
| $\pi \models_m$ *False*$_m$ = *False*
| $\pi \models_m$ *Prop*$_m$ (*q*) = $(\pi \neq [] \wedge q \in (\pi$! *0*$))$
| $\pi \models_m$ *Not*$_m$ $\varphi = (\neg\ \pi \models_m \varphi)$
| $\pi \models_m \varphi$ *And*$_m$ $\psi = (\pi \models_m \varphi \wedge \pi \models_m \psi)$
| $\pi \models_m \varphi$ *Or*$_m$ $\psi = (\pi \models_m \varphi \vee \pi \models_m \psi)$
| $\pi \models_m$ (*F*$_m$ [*a*, *b*] φ) = $(a \leq b \wedge$ *length* $\pi > a \wedge$
    $(\exists\, i$::*nat*. $(i \geq a \wedge i \leq b) \wedge$ (*drop* $i\ \pi) \models_m \varphi))$
| $\pi \models_m$ (*G*$_m$ [*a*, *b*] φ) = $(a \leq b \wedge$ (*length* $\pi \leq a \vee$
    $(\forall\, i$::*nat*. $(i \geq a \wedge i \leq b) \longrightarrow$ (*drop* $i\ \pi) \models_m \varphi)))$
| $\pi \models_m (\varphi$ *U*$_m$ [*a*, *b*] $\psi) = \ (a \leq b \wedge$ *length* $\pi > a \wedge$
    $(\exists\, i$::*nat*. $(i \geq a \wedge i \leq b) \wedge ((drop\ i\ \pi) \models_m \psi$

$\wedge$ ($\forall j.\ j \geq a \wedge j<i \longrightarrow (drop\ j\ \pi) \models_m \varphi$))))
| $\pi \models_m (\varphi\ R_m\ [a,\ b]\ \psi) = (a \leq b \wedge (length\ \pi \leq a \vee$
($\forall i{::}nat.\ (i \geq a \wedge i \leq b) \longrightarrow (((drop\ i\ \pi) \models_m \psi))) \vee$
($\exists j.\ j \geq a \wedge j \leq b{-}1 \wedge (drop\ j\ \pi) \models_m \varphi \wedge$
($\forall k.\ a \leq k \wedge k \leq j \longrightarrow (drop\ k\ \pi) \models_m \psi$))))

1.2.1 Examples

lemma
$[\{0{::}nat\}] \models_m Not_m\ (F_m\ [0,2]\ Prop_m\ (0)) = False$
⟨*proof*⟩

lemma
$[\{0{::}nat\}] \models_m F_m\ [0,2]\ (Not_m\ Prop_m\ (0)) = True$
⟨*proof*⟩

lemma
$[\{0{::}nat\}] \models_m G_m\ [0,2]\ Prop_m\ (0{::}nat) = False$
⟨*proof*⟩

end

2 Properties of MLTL

theory *MLTL-Properties*

imports *MLTL-Encoding*

begin

2.1 Useful Functions

We use the following function to assume that an MLTL formula is well-defined: i.e., that all intervals in the formula satisfy a is less than or equal to b

fun *intervals-welldef*:: *'a mltl* ⇒ *bool*
where *intervals-welldef* $True_m = True$
| *intervals-welldef* $False_m = True$
| *intervals-welldef* $(Prop_m\ (p)) = True$
| *intervals-welldef* $(Not_m\ \varphi) =$ *intervals-welldef* φ
| *intervals-welldef* $(\varphi\ And_m\ \psi) = (intervals\text{-}welldef\ \varphi \wedge intervals\text{-}welldef\ \psi)$
| *intervals-welldef* $(\varphi\ Or_m\ \psi) = (intervals\text{-}welldef\ \varphi \wedge intervals\text{-}welldef\ \psi)$
| *intervals-welldef* $(F_m\ [a,b]\ \varphi) = (a \leq b \wedge intervals\text{-}welldef\ \varphi)$
| *intervals-welldef* $(G_m\ [a,b]\ \varphi) = (a \leq b \wedge intervals\text{-}welldef\ \varphi)$
| *intervals-welldef* $(\varphi\ U_m\ [a,b]\ \psi) =$
$(a \leq b \wedge intervals\text{-}welldef\ \varphi \wedge intervals\text{-}welldef\ \psi)$
| *intervals-welldef* $(\varphi\ R_m\ [a,b]\ \psi) =$

$(a \leq b \wedge \mathit{intervals\text{-}welldef}\ \varphi \wedge \mathit{intervals\text{-}welldef}\ \psi)$

2.2 Semantic Equivalence

definition *semantic-equiv*:: *'a mltl* ⇒ *'a mltl* ⇒ *bool* (- $\equiv_m$ - [*80*, *80*] *80*)
where $\varphi \equiv_m \psi \equiv (\forall \pi.\ \pi \models_m \varphi = \pi \models_m \psi)$

fun *depth-mltl*:: *'a mltl* ⇒ *nat*
where *depth-mltl* $True_m$ = *0*
| *depth-mltl* $False_m$ = *0*
| *depth-mltl* $Prop_m$ (*p*) = *0*
| *depth-mltl* (Not_m φ) = *1* + *depth-mltl* φ
| *depth-mltl* (φ And_m ψ) = *1* + *max* (*depth-mltl* φ) (*depth-mltl* ψ)
| *depth-mltl* (φ Or_m ψ) = *1* + *max* (*depth-mltl* φ) (*depth-mltl* ψ)
| *depth-mltl* (G_m [*a*,*b*] φ) = *1* + *depth-mltl* φ
| *depth-mltl* (F_m [*a*,*b*] φ) = *1* + *depth-mltl* φ
| *depth-mltl* (φ U_m [*a*,*b*] ψ) = *1* + *max* (*depth-mltl* φ) (*depth-mltl* ψ)
| *depth-mltl* (φ R_m [*a*,*b*] ψ) = *1* + *max* (*depth-mltl* φ) (*depth-mltl* ψ)

fun *subformulas*:: *'a mltl* ⇒ *'a mltl set*
where *subformulas* $True_m$ = {}
| *subformulas* $False_m$ = {}
| *subformulas* $Prop_m$ (*p*) = {}
| *subformulas* (Not_m φ) = {φ} ∪ *subformulas* φ
| *subformulas* (φ And_m ψ) = {φ, ψ} ∪ *subformulas* φ ∪ *subformulas* ψ
| *subformulas* (φ Or_m ψ) = {φ, ψ} ∪ *subformulas* φ ∪ *subformulas* ψ
| *subformulas* (G_m [*a*,*b*] φ) = {φ} ∪ *subformulas* φ
| *subformulas* (F_m [*a*,*b*] φ) = {φ} ∪ *subformulas* φ
| *subformulas* (φ U_m [*a*,*b*] ψ) = {φ, ψ} ∪ *subformulas* φ ∪ *subformulas* ψ
| *subformulas* (φ R_m [*a*,*b*] ψ) = {φ, ψ} ∪ *subformulas* φ ∪ *subformulas* ψ

2.3 Basic Properties

lemma *future-or-distribute*:
shows F_m [*a*,*b*] (*φ1* Or_m *φ2*) $\equiv_m$ (F_m [*a*,*b*] *φ1*) Or_m (F_m [*a*,*b*] *φ2*)
⟨*proof*⟩

lemma *global-and-distribute*:
shows G_m [*a*,*b*] (*φ1* And_m *φ2*) $\equiv_m$ (G_m [*a*,*b*] *φ1*) And_m (G_m [*a*,*b*] *φ2*)
⟨*proof*⟩

lemma *not-not-equiv*:
shows $\varphi \equiv_m (Not_m\ (Not_m\ \varphi))$
⟨*proof*⟩

lemma *demorgan-and-or*:
shows Not_m (φ And_m ψ) $\equiv_m$ (Not_m φ) Or_m (Not_m ψ)
⟨*proof*⟩

lemma *demorgan-or-and*:

shows *semantic-equiv* (*Not-mltl* (φ Or_m ψ))
(*And-mltl* (*Not*$_m$ φ) (*Not-mltl* ψ))
⟨*proof*⟩

lemma *future-as-until*:
fixes a b::*nat*
assumes $a \leq b$
shows (F_m [a,b] φ) $\equiv_m$ (*True*$_m$ U_m [a,b] φ)
⟨*proof*⟩

lemma *globally-as-release*:
fixes a b::*nat*
assumes $a \leq b$
shows (G_m [a,b] φ) $\equiv_m$ (*False*$_m$ R_m [a,b] φ)
⟨*proof*⟩

lemma *until-or-distribute*:
fixes a b ::*nat*
assumes $a \leq b$
shows φ U_m [a,b] (α Or_m β) $\equiv_m$
(φ U_m [a,b] α) Or_m (φ U_m [a,b] β)
⟨*proof*⟩

lemma *until-and-distribute*:
fixes a b ::*nat*
assumes $a \leq b$
shows (α And_m β) U_m [a,b] φ $\equiv_m$
(α U_m [a,b] φ) And_m (β U_m [a,b] φ)
⟨*proof*⟩

lemma *release-or-distribute*:
fixes a b ::*nat*
assumes $a \leq b$
shows (α Or_m β) R_m [a,b] φ $\equiv_m$
(α R_m [a,b] φ) Or_m (β R_m [a,b] φ)
⟨*proof*⟩

lemma *different-next-operators*:
shows $\neg$(G_m [*1*,*1*] φ $\equiv_m$ F_m [*1*,*1*] φ)
⟨*proof*⟩

2.4 Duality Properties

lemma *globally-future-dual*:
fixes a b::*nat*
assumes $a \leq b$
shows (G_m [a,b] φ) $\equiv_m$ *Not*$_m$ (F_m [a,b] (*Not*$_m$ φ))
⟨*proof*⟩

lemma *future-globally-dual*:
fixes a b::*nat*
assumes $a \leq b$
shows $(F_m\ [a,b]\ \varphi) \equiv_m Not_m\ (G_m\ [a,b]\ (Not_m\ \varphi))$
⟨*proof*⟩

Proof altered from source material in the last case.

lemma *release-until-dual1*:
fixes a b::*nat*
assumes $\pi \models_m (\varphi\ R_m\ [a,b]\ \psi)$
shows $\pi \models_m (Not_m\ ((Not_m\ \varphi)\ U_m\ [a,b]\ (Not_m\ \psi)))$
⟨*proof*⟩

lemma *release-until-dual2*:
fixes a b::*nat*
assumes *a-leq-b*: $a \leq b$
assumes $\pi \models_m (Not_m\ ((Not_m\ \varphi)\ U_m\ [a,b]\ (Not_m\ \psi)))$
shows *semantics-mltl* $\pi\ (\varphi\ R_m\ [a,b]\ \psi)$
⟨*proof*⟩

lemma *release-until-dual*:
fixes a b::*nat*
assumes *a-leq-b*: $a \leq b$
shows $(\varphi\ R_m\ [a,b]\ \psi) \equiv_m (Not_m\ ((Not_m\ \varphi)\ U_m\ [a,b]\ (Not_m\ \psi)))$
⟨*proof*⟩

lemma *until-release-dual*:
fixes a b::*nat*
assumes *a-leq-b*: $a \leq b$
shows $(\varphi\ U_m\ [a,b]\ \psi) \equiv_m (Not_m\ ((Not_m\ \varphi)\ R_m\ [a,b]\ (Not_m\ \psi)))$
⟨*proof*⟩

2.5 Additional Basic Properties

lemma *release-and-distribute*:
fixes a b ::*nat*
assumes $a \leq b$
shows $(\varphi\ R_m\ [a,b]\ (\alpha\ And_m\ \beta)) \equiv_m$
$((\varphi\ R_m\ [a,b]\ \alpha)\ And_m\ (\varphi\ R_m\ [a,b]\ \beta))$
⟨*proof*⟩

2.6 NNF Transformation and Properties

fun *convert-nnf*:: *'a mltl* ⇒ *'a mltl*
where *convert-nnf* $True_m = True_m$
| *convert-nnf* $False_m = False_m$
| *convert-nnf* $Prop_m\ (p) = Prop_m\ (p)$
| *convert-nnf* $(\varphi\ And_m\ \psi) = ((convert\text{-}nnf\ \varphi)\ And_m\ (convert\text{-}nnf\ \psi))$
| *convert-nnf* $(\varphi\ Or_m\ \psi) = ((convert\text{-}nnf\ \varphi)\ Or_m\ (convert\text{-}nnf\ \psi))$
| *convert-nnf* $(F_m\ [a,b]\ \varphi) = (F_m\ [a,b]\ (convert\text{-}nnf\ \varphi))$

| *convert-nnf* (G_m [*a*,*b*] φ) = (G_m [*a*,*b*] (*convert-nnf* φ))
| *convert-nnf* (φ U_m [*a*,*b*] ψ) = ((*convert-nnf* φ) U_m [*a*,*b*] (*convert-nnf* ψ))
| *convert-nnf* (φ R_m [*a*,*b*] ψ) = ((*convert-nnf* φ) R_m [*a*,*b*] (*convert-nnf* ψ))

| *convert-nnf* (Not_m $True_m$) = $False_m$
| *convert-nnf* (Not_m $False_m$) = $True_m$
| *convert-nnf* (Not_m $Prop_m$ (*p*)) = (Not_m $Prop_m$ (*p*))
| *convert-nnf* (Not_m (Not_m φ)) = *convert-nnf* φ
| *convert-nnf* (Not_m (φ And_m ψ)) = ((*convert-nnf* (Not_m φ)) Or_m (*convert-nnf* (Not_m ψ)))
| *convert-nnf* (Not_m (φ Or_m ψ)) = ((*convert-nnf* (Not_m φ)) And_m (*convert-nnf* (Not_m ψ)))
| *convert-nnf* (Not_m (F_m [*a*,*b*] φ)) = (G_m [*a*,*b*] (*convert-nnf* (Not_m φ)))
| *convert-nnf* (Not_m (G_m [*a*,*b*] φ)) = (F_m [*a*,*b*] (*convert-nnf* (Not_m φ)))
| *convert-nnf* (Not_m (φ U_m [*a*,*b*] ψ)) = ((*convert-nnf* (Not_m φ)) R_m [*a*,*b*] (*convert-nnf* (Not_m ψ)))
| *convert-nnf* (Not_m (φ R_m [*a*,*b*] ψ)) = ((*convert-nnf* (Not_m φ)) U_m [*a*,*b*] (*convert-nnf* (Not_m ψ)))

lemma *convert-nnf-preserves-semantics*:
assumes *intervals-welldef* φ
shows (π $\models_m$ (*convert-nnf* φ)) $\longleftrightarrow$ (π $\models_m$ φ)
⟨*proof*⟩

lemma *convert-nnf-form-Not-Implies-Prop*:
assumes Not_m *F* = *convert-nnf init-F*
shows $\exists$ *p*. *F* = $Prop_m$ (*p*)
⟨*proof*⟩

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

lemma *nnf-subformulas*:
assumes *F* = *convert-nnf init-F*
assumes *G* ∈ *subformulas F*
shows ∃ *init-G*. *G* = *convert-nnf init-G*
⟨*proof*⟩

2.7 Computation Length and Properties

fun *complen-mltl*:: *'a mltl* ⇒ *nat*
where *complen-mltl* $False_m$ = *1*
| *complen-mltl* $True_m$ = *1*
| *complen-mltl* $Prop_m$ (*p*) = *1*
| *complen-mltl* (Not_m φ) = *complen-mltl* φ
| *complen-mltl* (φ And_m ψ) = *max* (*complen-mltl* φ) (*complen-mltl* ψ)
| *complen-mltl* (φ Or_m ψ) = *max* (*complen-mltl* φ) (*complen-mltl* ψ)

| *complen-mltl* (G_m [*a*,*b*] φ) = *b* + (*complen-mltl* φ)
| *complen-mltl* (F_m [*a*,*b*] φ) = *b* + (*complen-mltl* φ)
| *complen-mltl* (φ R_m [*a*,*b*] ψ) = *b* + (*max* ((*complen-mltl* φ)−*1*) (*complen-mltl* ψ))
| *complen-mltl* (φ U_m [*a*,*b*] ψ) = *b* + (*max* ((*complen-mltl* φ)−*1*) (*complen-mltl* ψ))

lemma *complen-geq-one*: *complen-mltl F* ≥ *1*
⟨*proof*⟩

2.7.1 Capture (not (a <= b)) in an MLTL formula

fun *make-empty-trace*:: *nat* ⇒ *'a set list*
where *make-empty-trace 0* = []
| *make-empty-trace n* = [{}]@ *make-empty-trace* (*n*−*1*)

lemma *length-make-empty-trace*:
shows *length* (*make-empty-trace n*) = *n*
⟨*proof*⟩

lemma *semantics-of-not-a-lteq-b*:
shows (*make-empty-trace* (*a*+*1*)) ⊨$_m$ (*Global-mltl a b True*$_m$) = (*a* ≤ *b*)
⟨*proof*⟩

lemma *semantics-of-not-a-lteq-b2*:
shows (*make-empty-trace* (*a*+*1*)) ⊨$_m$ (*Not-mltl* (*Global-mltl a b True*$_m$)) = (¬ (*a* ≤ *b*))
⟨*proof*⟩

2.8 Custom Induction Rules

In some cases, it is sufficient to consider just a subset of MLTL operators when proving a property. We facilitate this with the following custom induction rules.

In order to use the MLTL-induct rule, one must establish IntervalsWellDef, which states that the input formula is well-formed, and also prove PProp, which states that the property being established is not dependent on the syntax of the input formula but only on its semantics.

lemma *MLTL-induct*[*case-names IntervalsWellDef PProp True False Prop Not And Until*]:
assumes *IntervalsWellDef*: *intervals-welldef F*
and *PProp*: (⋀ *F G*. ((∀ π. *semantics-mltl* π *F* = *semantics-mltl* π *G*) ⟶ *P F* = *P G*))
and *True*: *P True*$_m$
and *False*: *P False*$_m$
and *Prop*: ⋀ *p*. *P Prop*$_m$ (*p*)
and *Not*: ⋀ *F G*. ⟦*F* = *Not*$_m$ *G*; *P G*⟧ ⟹ *P F*

and *And*: $\bigwedge F\ F1\ F2.\ [\![F = F1\ And_m\ F2;$
$P\ F1;\ P\ F2]\!] \Longrightarrow P\ F$
and *Until*:$\bigwedge F\ F1\ F2\ a\ b.\ [\![F = F1\ U_m\ [a,b]\ F2;$
$P\ F1;\ P\ F2]\!] \Longrightarrow P\ F$
shows $P\ F$ ⟨*proof*⟩

In order to use the nnf-induct rule, one must establish that the input formula (i.e. the formula being inducted on) is in NNF format.

lemma *nnf-induct*[*case-names nnf True False Prop And Or Final Global Until Release NotProp*]:
assumes *nnf*: $\exists$ *init-F*. F = *convert-nnf init-F*
and *True*: $P\ True_m$
and *False*: $P\ False_m$
and *Prop*: $\bigwedge p.\ P\ Prop_m\ (p)$
and *And*: $\bigwedge F\ F1\ F2.\ [\![F = F1\ And_m\ F2;$
$P\ F1;\ P\ F2]\!] \Longrightarrow P\ F$
and *Or*: $\bigwedge F\ F1\ F2.\ [\![F = F1\ Or_m\ F2;$
$P\ F1;\ P\ F2]\!] \Longrightarrow P\ F$
and *Final*: $\bigwedge F\ F1\ \ a\ b.\ [\![F = F_m\ [a,b]\ F1;$
$P\ F1]\!] \Longrightarrow P\ F$
and *Global*: $\bigwedge F\ F1\ \ a\ b.\ [\![F = G_m\ [a,b]\ F1;$
$P\ F1]\!] \Longrightarrow P\ F$
and *Until*: $\bigwedge F\ F1\ F2\ \ a\ b.\ [\![F = F1\ U_m\ [a,b]\ F2;$
$P\ F1;\ P\ F2]\!] \Longrightarrow P\ F$
and *Release*: $\bigwedge F\ F1\ F2\ a\ b.\ [\![F = F1\ R_m\ [a,b]\ F2;$
$P\ F1;\ P\ F2]\!] \Longrightarrow P\ F$
and *Not-Prop*: $\bigwedge F\ p.\ F = Not_m\ Prop_m\ (p) \Longrightarrow P\ F$
shows $P\ F$ ⟨*proof*⟩

end

References

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

[2] J. Li and K. Y. Rozier. MLTL benchmark generation via formula progression. In C. Colombo and M. Leucker, editors, *RV*, volume 11237 of *LNCS*, pages 426–433. Springer, 2018.

[3] J. Li, M. Y. Vardi, and K. Y. Rozier. Satisfiability checking for mission-time LTL. In I. Dillig and S. Tasiran, editors, *CAV*, volume 11562 of *LNCS*, pages 3–22. Springer, 2019.

[4] T. Reinbacher, K. Y. Rozier, and J. Schumann. Temporal-logic based runtime observer pairs for system health management of real-time sys-

tems. In *TACAS*, volume 8413 of *LNCS*, pages 357–372. Springer-Verlag, April 2014.

[5] J. Schneider and D. Traytel. Formalization of a monitoring algorithm for metric first-order temporal logic. *Archive of Formal Proofs*, July 2019. https://isa-afp.org/entries/MFOTL_Monitor.html, Formal proof development.

[6] S. Sickert. Linear temporal logic. *Archive of Formal Proofs*, March 2016. https://isa-afp.org/entries/LTL.html, Formal proof development.