Formalizing MLTL in Isabelle/HOL

Katherine Kosaian Zili Wang Elizabeth Sloan

February 6, 2026

Abstract

We build on the Isabelle/HOL formalization of Mission-time Linear Temporal Logic (MLTL) to formalize a formula progression algorithm for MLTL formulas [1], a key algorithm in the FPROGG tool [2] for generating MLTL benchmarks. The formula progression algorithm takes a MLTL formula and steps through a given trace to partially evaluate a logically equivalent simpler formula at each step, ultimately checking whether or not the trace satisfies the original formula. Our formalization is executable and we export it to code in SML.

Contents

1 MLTL formula progresion

theory *MLTL-Formula-Progression*

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

begin

1.1 Algorithm

fun *weight-operators*:: *'a mltl* ⇒ *nat* **where**
weight-operators True$_m$ = *1*
| *weight-operators False*$_m$ = *1*
| *weight-operators* (*Prop*$_m$ (*p*)) = *1*
| *weight-operators* (*F1 And*$_m$ *F2*) = *weight-operators F1* + *weight-operators F2* + *1*
| *weight-operators* (*F1 Or*$_m$ *F2*) = *weight-operators F1* + *weight-operators F2* + *1*
| *weight-operators* (*Not*$_m$ *F*) = *1* + *weight-operators F*
| *weight-operators* (*F1 U*$_m$ [*a*,*b*] *F2*) = *weight-operators F1* + *weight-operators F2* + *1* + *a*+ *b*
| *weight-operators* (*F1 R*$_m$ [*a*,*b*] *F2*) = *10* + *weight-operators F1* + *weight-operators F2* + *1*+ *a*+ *b*
| *weight-operators* (*G*$_m$ [*a*,*b*] *F*) = *10* + *weight-operators F*+ *a*+ *b*
| *weight-operators* (*F*$_m$ [*a*,*b*] *F*) = *1* + *weight-operators F*+ *a*+ *b*

function *formula-progression-len1*:: *'a mltl* ⇒ *'a set* ⇒ *'a mltl* **where**
formula-progression-len1 True$_m$ *tr-entry* = *True*$_m$
| *formula-progression-len1 False*$_m$ *tr-entry* = *False*$_m$
| *formula-progression-len1* (*Prop*$_m$ (*p*)) *tr-entry* = (*if p* ∈ *tr-entry then True*$_m$ *else False*$_m$)
| *formula-progression-len1* (*Not*$_m$ *F*) *tr-entry* = *Not*$_m$ (*formula-progression-len1 F tr-entry*)
| *formula-progression-len1* (*F1 And*$_m$ *F2*) *tr-entry* = (*formula-progression-len1 F1 tr-entry*) *And*$_m$ (*formula-progression-len1 F2 tr-entry*)
| *formula-progression-len1* (*F1 Or*$_m$ *F2*) *tr-entry* = (*formula-progression-len1 F1 tr-entry*) *Or*$_m$ (*formula-progression-len1 F2 tr-entry*)
| *formula-progression-len1* (*F1 U*$_m$ [*a*,*b*] *F2*) *tr-entry* =
(*if* (*0*<*a* ∧ *a*≤*b*) *then* (*F1 U*$_m$ [(*a*−*1*), (*b*−*1*)] *F2*)
else (*if* (*0*=*a* ∧ *a*<*b*) *then* ((*formula-progression-len1 F2 tr-entry*) *Or*$_m$ ((*formula-progression-len1 F1 tr-entry*) *And*$_m$ (*F1 U*$_m$ [*0*, (*b*−*1*)] *F2*)))
else (*formula-progression-len1 F2 tr-entry*)))
| *formula-progression-len1* (*F1 R*$_m$ [*a*,*b*] *F2*) *tr-entry* = *Not*$_m$ (*formula-progression-len1* ((*Not*$_m$ *F1*) *U*$_m$ [*a*,*b*] (*Not*$_m$ *F2*)) *tr-entry*)
| *formula-progression-len1* (*G*$_m$ [*a*,*b*] *F*) *tr-entry* = *Not*$_m$ (*formula-progression-len1* (*F*$_m$ [*a*,*b*] (*Not*$_m$ *F*)) *tr-entry*)
| *formula-progression-len1* (*F*$_m$ [*a*,*b*] *F*) *tr-entry* =
(*if 0*< *a* ∧ *a*≤ *b then* (*F*$_m$ [(*a*−*1*), (*b*−*1*)] *F*)
else if (*0* = *a* ∧ *a* < *b*) *then* ((*formula-progression-len1 F tr-entry*) *Or*$_m$ (*F*$_m$ [*0*, (*b*−*1*)] *F*))
else (*formula-progression-len1 F tr-entry*))
⟨*proof*⟩
termination ⟨*proof*⟩

Note that formula progression needs to be defined when the length of the trace is 0. In this case, we define it to just return the original formula.

fun *formula-progression*:: *'a mltl* ⇒ *'a set list* ⇒ *'a mltl*
where *formula-progression F tr* =

(*if length tr = 0 then F*
else (*if length tr = 1 then* (*formula-progression-len1 F* (*tr*!*0*))
else (*formula-progression* (*formula-progression-len1 F* (*tr* ! *0*)) (*drop 1 tr*))))

value *take 2* ([*0*::*nat*, *1*, *2*, *3*]::*nat list*)
value *drop 2* ([*0*::*nat*, *1*, *2*, *3*]::*nat list*)

lemma *formula-progression-alt*:
formula-progression F xs = *fold* (λ*x F*. *formula-progression-len1 F x*) *xs F*
⟨*proof*⟩

1.2 Proofs

1.2.1 Empty Trace Semantics of MLTL

lemma *semantics-global*:
shows [] ⊨$_m$ (G_m [*0*,*1*] φ)
⟨*proof*⟩

lemma *semantics-future*:
shows [] ⊨$_m$ (Not_m (F_m [*0*,*1*] (Not_m φ)))
⟨*proof*⟩

1.2.2 Well-definedness Properties

lemma *formula-progression-well-definedness-preserved-len1*:
assumes *intervals-welldef* φ
shows *intervals-welldef* (*formula-progression-len1* φ π)
⟨*proof*⟩

lemma *formula-progression-well-definedness-preserved*:
assumes *intervals-welldef* φ
shows *intervals-welldef* (*formula-progression* φ π)
⟨*proof*⟩

1.2.3 Theorem 1

Helper lemma for Theorem 1

lemma *formula-progression-identity*:
fixes φ::*'a mltl*
fixes *k*::*nat*
assumes *k* < *length* π
shows *formula-progression* (*formula-progression* φ (*take k* π)) [π ! *k*]
= *formula-progression* φ (*take* (*k*+*1*) π)
⟨*proof*⟩

Theorem 1

theorem *formula-progression-decomposition*:
fixes φ::*'a mltl*

assumes $k \geq 1$
assumes $k \leq$ *length* π
shows *formula-progression* (*formula-progression* φ (*take* k π)) (*drop* k π)
$=$ *formula-progression* φ π
⟨*proof*⟩

1.2.4 Theorem 2

Base case for Theorem 2

lemma *satisfiability-preservation-len1*:
fixes φ::*'a mltl*
assumes *1* $<$ *length* π
assumes *intervals-welldef* φ
shows *semantics-mltl* (*drop 1* π) (*formula-progression-len1* φ (π ! *0*))
$\longleftrightarrow$ *semantics-mltl* π φ
⟨*proof*⟩

Theorem 2

theorem *satisfiability-preservation*:
fixes φ::*'a mltl*
assumes $k \geq 1$
assumes $k <$ *length* π
assumes *intervals-welldef* φ
shows *semantics-mltl* (*drop* k π) (*formula-progression* φ (*take* k π))
$\longleftrightarrow$ *semantics-mltl* π φ
⟨*proof*⟩

Counter example to Theorem 2 showing how the theorem can fail if the trace length condition is removed. **lemma** *theorem2-cexa*:
fixes φ::*nat mltl*
assumes $k = 1$
assumes $\pi = [\{1::nat\}]$
assumes $\varphi = G_m$ [*0*,*3*] (*Prop-mltl* (*1*::*nat*))
assumes *intervals-welldef* φ
shows (*drop* k π) $\models_m$ (*formula-progression* φ (*take* k π)) $=$ *True*
⟨*proof*⟩

value (*take 1* [{*1*::*nat*}])
value *formula-progression* (G_m [*0*,*3*] (*Prop-mltl* (*1*::*nat*))) (*take 1* [{*1*::*nat*}])

lemma *theorem2-cexb*:
fixes φ::*nat mltl*
assumes $\pi = [\{1::nat\}]$
assumes $\varphi = G_m$ [*0*,*1*] (*Prop-mltl* (*1*::*nat*))
assumes *intervals-welldef* φ
shows *semantics-mltl* π φ $=$ *False*
⟨*proof*⟩

1.2.5 Theorem 3

Setup: Properties of Computation Length **lemma** *complen-geq-1*:
shows *complen-mltl* $\varphi \geq 1$
⟨*proof*⟩

This is a key property that makes the base case of Theorem 3 work: Constraining the computation length of the formula means that the formula progression is either globally true or false. This is a very strong structural property that lets us use the inductive hypotheses in, e.g., the Or case and the Not case of the base case of Theorem 3.

lemma *complen-bounded-by-1*:
assumes *intervals-welldef* φ
assumes $1 \geq$ *complen-mltl* φ
shows $(\forall \xi.\ \xi \models_m (\textit{formula-progression-len1}\ \varphi\ \pi)) \vee$
$(\forall \xi.\ \neg\ (\xi \models_m (\textit{formula-progression-len1}\ \varphi\ \pi)))$
⟨*proof*⟩

lemma *complen-temporal-props*:
shows $(\textit{complen-mltl}\ (F_m\ [a,b]\ \varphi) = 1 \implies (b = 0))$
$(\textit{complen-mltl}\ (G_m\ [a,b]\ \varphi) = 1 \implies (b = 0))$
$(\textit{complen-mltl}\ (\varphi 1\ U_m\ [a,b]\ \varphi 2) = 1 \implies (b = 0))$
$(\textit{complen-mltl}\ (\varphi 1\ R_m\ [a,b]\ \varphi 2) = 1 \implies (b = 0))$
⟨*proof*⟩

lemma *complen-one-implies-one-base*:
assumes *intervals-welldef* φ
assumes *complen-mltl* $\varphi = 1$
shows *complen-mltl* (*formula-progression-len1* $\varphi\ k$) $= 1$
⟨*proof*⟩

lemma *complen-one-implies-one*:
assumes *intervals-welldef* φ
assumes *complen-mltl* $\varphi = 1$
shows *complen-mltl* (*formula-progression* $\varphi\ \pi$) $= 1$
⟨*proof*⟩

lemma *formula-progression-decreases-complen-base*:
assumes *intervals-welldef* φ
shows *complen-mltl* $\varphi = 1 \vee$ *complen-mltl* (*formula-progression-len1* $\varphi\ k$) $\leq$ *complen-mltl* $\varphi - 1$
⟨*proof*⟩

Key helper lemma — relates computation length and formula progression. Intuitively, the formula progression usually decreases the computation length.

lemma *formula-progression-decreases-complen*:
assumes *intervals-welldef* φ
shows *complen-mltl* $\varphi = 1 \vee$ *complen-mltl* (*formula-progression* $\varphi\ \pi$) $= 1 \vee$

complen-mltl (*formula-progression* φ π) $\leq$ *complen-mltl* φ $-$ (*length* π)
⟨*proof*⟩

Base case **lemma** *formula-progression-correctness-len1-helper*:
fixes φ::*'a mltl*
assumes *length* π = *1*
assumes *intervals-welldef* φ
assumes *length* π $\geq$ *complen-mltl* φ
shows (*semantic-equiv* (*formula-progression-len1* φ (π ! *0*)) *True-mltl*) $\longleftrightarrow$ *semantics-mltl* [π!*0*] φ
⟨*proof*⟩

lemma *formula-progression-correctness-len1*:
fixes φ::*'a mltl*
assumes *length* π = *1*
assumes *intervals-welldef* φ
assumes *length* π $\geq$ *complen-mltl* φ
shows (*formula-progression* φ π $\equiv_m$ $True_m$) $\longleftrightarrow$ π $\models_m$ φ
⟨*proof*⟩

Top-Level Result and Corollary **theorem** *formula-progression-correctness*:
fixes φ::*'a mltl*
assumes *intervals-welldef* φ
assumes *length* π $\geq$ *complen-mltl* φ
shows (*formula-progression* φ π $\equiv_m$ $True_m$) $\longleftrightarrow$ π $\models_m$ φ
⟨*proof*⟩

Adds the crucial assumption that the length of the trace is greater than or equal to the computation length of the formula.

corollary *formula-progression-append*:
fixes φ::*'a mltl*
assumes *intervals-welldef* φ
assumes π $\models_m$ φ
assumes *length* π $\geq$ *complen-mltl* φ
shows (π @ ζ) $\models_m$ φ
⟨*proof*⟩

Converse of Corollary and Combined Statement Alternate statement of the formula progression correctness lemma that asserts formula progression on a trace of length one is semantically equivalent to False mltl when the formula is not satisfied

lemma *formula-progression-correctness-len1-helper-alt*:
fixes φ::*'a mltl*
assumes *length* π = *1*
assumes *intervals-welldef* φ
assumes *length* π $\geq$ *complen-mltl* φ
shows ((*formula-progression-len1* φ (π ! *0*)) $\equiv_m$ $False_m$) $\longleftrightarrow$ $\neg$ ([π!*0*] $\models_m$ φ)
⟨*proof*⟩

Alternate statement of the formula-progression-correctness lemma with False in the case that the semantics are not satisfied.

lemma *formula-progression-correctness-len1-alt*:
fixes φ::*'a mltl*
assumes *length* π = *1*
assumes *intervals-welldef* φ
assumes *length* $\pi \geq$ *complen-mltl* φ
shows ((*formula-progression* φ π) $\equiv_m$ *False-mltl*) $\longleftrightarrow$ $\neg$ π $\models_m$ φ
⟨*proof*⟩

theorem *formula-progression-correctness-alt*:
fixes φ::*'a mltl*
assumes *intervals-welldef* φ
assumes *length* $\pi \geq$ *complen-mltl* φ
shows ((*formula-progression* φ π) $\equiv_m$ *False-mltl*) $\longleftrightarrow$ $\neg$ (π $\models_m$ φ)
⟨*proof*⟩

lemma *formula-progression-true-or-false*:
fixes φ::*'a mltl*
assumes *intervals-welldef* φ
assumes *length* $\pi \geq$ *complen-mltl* φ
shows ((*formula-progression* φ π) $\equiv_m$ $False_m$) $\vee$
((*formula-progression* φ π) $\equiv_m$ $True_m$)
⟨*proof*⟩

The inverse statement of formula-progression-append lemma

corollary *formula-progression-append-converse*:
fixes φ::*'a mltl*
assumes *intervals-welldef* φ
assumes $\neg$ π $\models_m$ φ
assumes *length* $\pi \geq$ *complen-mltl* φ
shows $\neg$ (π @ ζ) $\models_m$ φ
⟨*proof*⟩

An important property of complen-mltl that says states in the trace after the computation length does not affect the semantic satisfaction of the formula.

corollary *complen-property*:
fixes φ::*'a mltl*
assumes *intervals-welldef* φ
assumes *length* $\pi \geq$ *complen-mltl* φ
shows π $\models_m$ φ $\longleftrightarrow$ ($\forall \zeta$. (π @ ζ) $\models_m$ φ)
⟨*proof*⟩

1.3 Formula Progression Examples

value *formula-progression*
((G_m [*0*,*2*] (*Prop-mltl 0*))::*nat mltl*)

[{0::nat}, {0}, {1}]

value [{0::nat}, {0}, {1}] ! 0
value drop 1 ([{0::nat}, {0}, {1}])
value formula-progression-len1 ((G$_m$ [0,2] (Prop-mltl 0))::nat mltl) {0}

value formula-progression
(formula-progression-len1
((G$_m$ [0,2] (Prop-mltl 0))::nat mltl)
{0}
)
[{0}, {1}]

value formula-progression
((G$_m$ [0,1] (Prop-mltl 0))::nat mltl)
[{0}, {1}]

value formula-progression
(formula-progression-len1
((Global-mltl 0 1 (Prop-mltl 0))::nat mltl)
{0})
[{1}]

value formula-progression-len1 ((G$_m$ [0,1] (Prop-mltl 0))::nat mltl) {0}

value formula-progression
((G$_m$ [0,0] (Prop-mltl 0))::nat mltl)
[{1}]

value formula-progression-len1
((G$_m$ [0,0] (Prop-mltl 0))::nat mltl)
{1}

1.4 Code Export

export-code
formula-progression
in SML **module-name** FP

end

References

[1] 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.

[2] A. Rosentrater and K. Y. Rozier. FPROGG: A formula progression-based MLTL benchmark generator. To appear; emailed to authors, 2025.