Simple LTL Model Checker for Finite-State Models

Mariam Tchubabria

February 2026

Abstract

In this project, I created a simple Linear Temporal Logic (LTL) model checker in Python for small finite-state models. The program can parse LTL formulas, find loops in state machines (lassos), and check whether the formulas hold along these paths. I tested it on several examples, including safety, liveness, next, until, and mutual exclusion properties, to show that it works correctly.

1 Introduction

Model checking is a way to verify whether a system behaves as expected. Formally, given a model M and a property ϕ, we want to know if $M \models \phi$ (i.e., ϕ is true in the model).

In this project, I focus on Linear Temporal Logic (LTL), which lets us express properties that depend on sequences of states over time.

My approach has a few main steps (I tried to keep it simple so it's easier for me to understand):

1. Represent the system as a finite-state model (a Kripke structure).
2. Parse LTL formulas into a tree-like structure so we can evaluate them.
3. Find all infinite paths in the system using the lasso method (a simple loop detection).
4. Evaluate the formula on each path to see if it holds.
5. Report whether the property is satisfied globally.

2 Kripke Structure

A Kripke structure is basically a state machine with labels. We define it as $M = (S, S_0, R, L)$, where:

- S is the set of states.
- $S_0 \subseteq S$ is the set of initial states.
- $R \subseteq S \times S$ is the transition relation.
- $L : S \to 2^{AP}$ maps each state to the set of atomic propositions that are true there.

In Python, I implemented it like this:

```python
@dataclass
class KripkeStructure:
    states: list
    init:   list
    trans:  dict   # {state: list of successors}
    label:  dict   # {state: set of atomic propositions}
```

Listing 1: Kripke Structure Definition

3 LTL Formulas

I represent LTL formulas as trees, where each node is either an operator (like `!`, `X`, `G`) or an atomic proposition.

```python
@dataclass
class LTLFormula:
    op:    str
    name:  Optional[str] = None
    left:  Optional['LTLFormula'] = None
    right: Optional['LTLFormula'] = None
```

Listing 2: LTL Formula Node

Logical operators: `!`, `&&`, `||`, `->` Temporal operators: `X` (next), `F` (finally), `G` (globally), `U` (until)

4 Parsing LTL Formulas

I wrote a parser that converts an LTL string into the tree structure. Operator precedence (lowest → highest) is:

Lowest	$->$
	||
	&&
	`U`
	Unary operators: `!`, `X`, `F`, `G`
Highest	Atoms (`p`, `q`, etc.)

The parser uses a recursive descent method:

```python
def parse_ltl(src: str) -> LTLFormula:
    return Parser(tokenize(src)).parse()
```

Listing 3: Parsing LTL Formulas

5 Finding Lasso Paths

Since our models are finite, every infinite path eventually loops. I used a DFS to find all paths that form a lasso. A lasso is a path like:

$$s_0 \to s_1 \to \cdots \to s_n, \quad s_m \to \cdots \to s_n \to s_m \ldots$$

Python code:

```python
def enumerate_lassos(model: KripkeStructure) -> list:
    lassos = []
    def dfs(state, path):
        if state in path:
            loop_start = path.index(state)
            lassos.append((list(path), loop_start))
            return
        if len(path) > len(model.states) * 2:
            return
        succs = model.trans.get(state, [])
        if not succs:
            lassos.append((path + [state], len(path)))
            return
        for succ in succs:
            dfs(succ, path + [state])
    for s in model.init:
        dfs(s, [])
    return lassos
```

Listing 4: Enumerate Lasso Paths

6 Evaluating LTL on a Lasso

The evaluation function checks if a formula holds along the infinite lasso path. Important points:

- `X` ϕ: ϕ should hold in the next state.
- `F` ϕ: ϕ should hold in some future state in the loop.
- `G` ϕ: ϕ should hold in all future states in the loop.
- ϕ `U` ψ: ψ eventually holds, and ϕ holds until then.

```python
def eval_ltl(phi: LTLFormula, seq: list, loop_start: int,
             model: KripkeStructure, pos: int) -> bool:
    # recursive evaluation for all operators
```

Listing 5: Evaluate LTL Formula

7 Model Checking

The model checker just goes through all lasso paths and evaluates the formula. If it holds on all paths, we say the property is satisfied.

```python
def model_check(model: KripkeStructure, phi: LTLFormula, verbose: bool=
    True) -> bool:
    lassos = enumerate_lassos(model)
    all_hold = True
    for seq, loop_start in lassos:
        result = eval_ltl(phi, seq, loop_start, model, pos=0)
        if verbose:
            loop_state = seq[loop_start]
            path_str = ' -> '.join(seq) + f' -> [{loop_state}...] (loop
    )'
```

```
            status = 'done' if result else 'nope'
            print(f"  {status} Path: {path_str}")
        if not result:
            all_hold = False
    return all_hold
```

Listing 6: Model Checker

8 Examples

I tested the checker on six examples:

1. Response property: $G(p \rightarrow Fq)$
2. Safety property: $G\neg error$
3. Liveness property: $GFdone$
4. Until property: pUq
5. Next property: Xq
6. Mutual exclusion: $G\neg(crit1 \wedge crit2)$

Each example defines a small Kripke structure and prints whether the formula is true or false.

9 Conclusion

In this project, I built a small Python program that can check LTL properties on finite-state models. The program shows the main ideas behind model checking: representing states, parsing formulas, finding loops, and evaluating temporal properties. While it is simple and not optimized for big systems, it works well for study examples.