

## A ADDITIONAL BACKGROUND AND DEFINITIONS

**Temporal Logic.** Temporal logic is a formalism that describes properties of sequences over time, specifically in systems' behaviors. It employs logical operators that capture temporal aspects such as "always," "eventually," and "until" to specify constraints and requirements on system executions. Formally, temporal logic formulas are defined inductively as:  $\varphi := p \in P_{\mathcal{M}} \mid \neg\varphi \mid \varphi \vee \varphi \mid \circ\varphi \mid \varphi \mathbf{U} \varphi$ . Intuitively, a temporal logic formula consists of a set of atomic propositions, a set of temporal operators, and a set of logical connectives.

The common-used temporal operators are  $\diamond$  ("eventually"),  $\mathbf{U}$  ("until"),  $\circ$  ("next"), and  $\square$  ("always"). And the logical connectives includes  $\wedge$  ("and"),  $\vee$  ("or"),  $\neg$  ("not"), etc.

**Product Automaton.** Let a controller be  $\mathcal{C} := \langle \Sigma, A, Q, q_0, \delta \rangle$  with input alphabet  $\Sigma := 2^P$ , output alphabet  $A := 2^{P_A}$ , and non-deterministic transition function  $\delta : Q \times \Sigma \times A \times Q \rightarrow \{0, 1\}$ .

Let a model be a tuple  $\mathcal{M} := \langle \Gamma_{\mathcal{M}}, Q_{\mathcal{M}}, \delta_{\mathcal{M}}, \lambda_{\mathcal{M}} \rangle$  with output symbols  $\Gamma_{\mathcal{M}} = 2^{P \cup \{goal\}}$ , a non-deterministic transition function  $\delta_{\mathcal{M}} : Q_{\mathcal{M}} \times Q_{\mathcal{M}} \rightarrow \{0, 1\}$ , and a label function  $\lambda_{\mathcal{M}} : Q_{\mathcal{M}} \rightarrow \Gamma_{\mathcal{M}}$ .

We define the *product automaton* as a transition system  $\mathfrak{P} = \mathcal{M} \otimes \mathcal{C} := \langle Q_{\mathfrak{P}}, \delta_{\mathfrak{P}}, q_{init}^{\mathfrak{P}}, \lambda_{\mathfrak{P}} \rangle$  as follows:

$$\begin{aligned} Q_{\mathfrak{P}} &:= Q_{\mathcal{M}} \times Q \\ \delta_{\mathfrak{P}}((p, q)) &:= \{(p', q') \in Q_{\mathfrak{P}} \mid \delta(q, \lambda_{\mathcal{M}}(p) \cap \Sigma, a, q') = 1 \text{ and } \delta_{\mathcal{M}}(p, p') = 1, \text{ for some } a \in A\} \\ q_{init}^{\mathfrak{P}} &:= \{(p, q_0) \mid p \in Q_{\mathcal{M}}\} \\ \lambda_{\mathfrak{P}}((p, q), (p', q')) &:= \{\lambda_{\mathcal{M}}(p) \cup a \mid a \in A \text{ and } \delta(q, \lambda_{\mathcal{M}}(p) \cap P_{\mathcal{M}}, a, q') = 1 \text{ and } \delta_{\mathcal{M}}(p, p') = 1\} \end{aligned}$$

$\delta_{\mathfrak{P}} : Q_{\mathfrak{P}} \rightarrow 2^{Q_{\mathfrak{P}}}$  is a non-deterministic transition function, and  $\lambda_{\mathfrak{P}} : Q_{\mathfrak{P}} \times Q_{\mathfrak{P}} \rightarrow 2^{P \cup P_A}$  is a label function.

The product automaton generates infinite-length trajectories in the form of  $(p_0, q_0), (p_1, q_1), \dots$  by beginning in an initial state  $q_{init}^{\mathfrak{P}}$  and following the nondeterministic transition function  $\delta_{\mathfrak{P}}$  thereafter. Labeled trajectories are then generated by applying the labeling function  $\lambda_{\mathfrak{P}}$  to these trajectories within the product automaton, i.e.  $\psi_0\psi_1, \dots \in (2^{P \cup P_A})^*$  where  $\psi_i \in \lambda_{\mathfrak{P}}((p_i, q_i), (p_{i+1}, q_{i+1}))$ . When using the product automaton to solve the model-checking problem, we check that all possible labeled trajectories generated by the product automaton belong to the language defined by the LTL specification.

## B ADDITIONAL EXPLANATIONS TO THE METHOD

**Formal Verification vs. Empirical Evaluation.** Formal verification provides a mathematical guarantee on whether a specification is satisfied, while empirical evaluation examines the controller's behavior in practical operations.

**Definition 1.** Let  $\mathcal{M}$  be the automaton-based model for the system  $\mathcal{S}$ . If  $\mathcal{M}$  captures all the transitions  $\{(\sigma, \neg\sigma) \mid \sigma \in P \cup P_A\}$  allowed by  $\mathcal{S}$ , then we say  $\mathcal{M}$  captures **complete information** from  $\mathcal{S}$ .

From the properties of formal verification, we can derive Theorem 1.

**Theorem 1.** If  $\mathcal{M}$  captures complete information from  $\mathcal{S}$ , then

$$\mathcal{M} \otimes \mathcal{C} \models \Phi \implies \mathbf{G}(\mathcal{C}, \mathcal{S}) \models \Phi. \quad (3)$$

*Proof.* Suppose  $\mathbf{G}(\mathcal{C}, \mathcal{S}) \not\models \Phi$ , then we can find at least one sequence  $(\sigma_i, a_i)^N \in (2^P \times 2^{P_A})^N$  that violates  $\Phi$ , i.e.,  $(\sigma_i, a_i)^N$  is a counter-example. Hence, we get  $\mathcal{M} \otimes \mathcal{C} \not\models \Phi$ . By contra-positive, Theorem 1 holds.  $\square$

Therefore, the formal verification results provide stronger guarantees than empirical results.

However, if the model  $\mathcal{M}$  does not capture complete information from the system, then the guarantees provided by formal verification are no longer valid. Hence, we can use empirical evaluation in place of formal guarantees. From another perspective, empirical evaluation can be used to check whether the model  $\mathcal{M}$  has captured complete information.

## C ADDITIONAL EMPIRICAL DEMONSTRATION

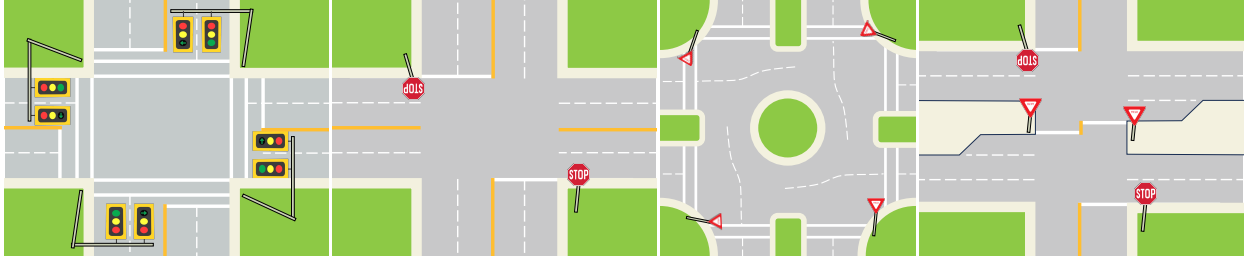


Figure 14. Illustration of different scenarios from the autonomous driving system. The figures from left to right show the scenarios for an intersection with a traffic light, a two-way stop sign, a roundabout, and an intersection with a wide median, respectively.

**Additional System Modeling** We present more examples of the automaton-based models encoding other scenarios in the autonomous system. The scenarios include a traffic light with an explicit left-turn signal, a two-way stop sign, and a roundabout.

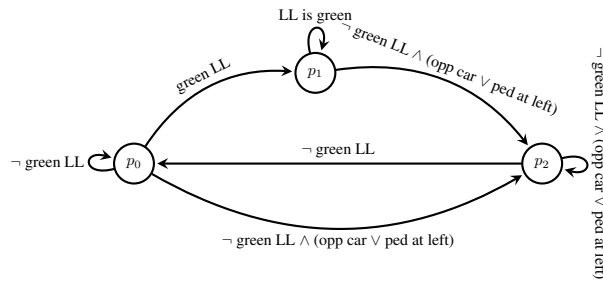


Figure 15. An automaton-based model represents a vehicle's environment dynamics at an intersection's left turn traffic signal. LL represents "Left-Turn Light," ped represents "pedestrian."

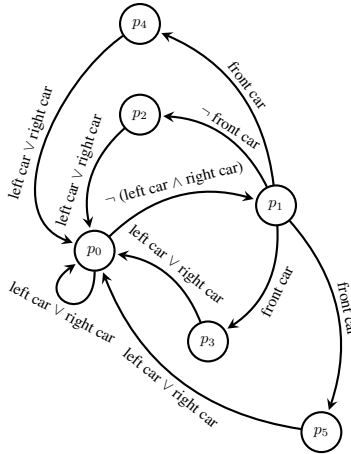


Figure 16. An automaton-based model represents a vehicle's environment dynamics at a two-way stop sign.

**The Complete Set of LTL Specifications.** We verify the LLM's outputs through the following 15 LTL specifications:

$$\Phi_1 = \square(\text{pedestrian} \rightarrow (\diamond \text{stop})),$$

$$\Phi_2 = \square(\text{opposite car} \wedge \neg \text{green left-turn light} \rightarrow \neg \text{turn left}),$$

$$\Phi_3 = \square(\neg \text{green traffic light} \rightarrow \neg \text{go straight}),$$

$$\Phi_4 = \square(\text{stop sign} \rightarrow \diamond \text{stop}),$$

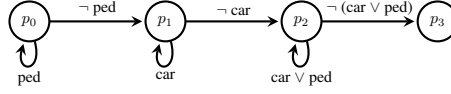


Figure 17. An automaton-based model representing the environment dynamics of a vehicle at a roundabout. The proposition “car” represents “car from left” and the proposition “ped” represents “pedestrian at left  $\wedge$  pedestrian at right.”

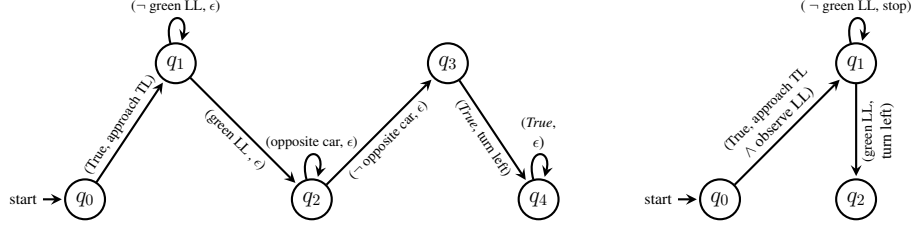


Figure 18. Automaton-based controllers for the task “turn left at the traffic light with the left-turn signal.” The left and right controllers are constructed from the language model before and after fine-tuning, respectively. TL and LL represent “traffic light” and “left-turn light.”

$$\Phi_5 = \square(\text{car from left} \vee \text{pedestrian at right} \rightarrow \neg \text{turn right}),$$

$$\Phi_6 = \square(\text{stop} \vee \text{go straight} \vee \text{turn left} \vee \text{turn right}),$$

$$\Phi_7 = \diamond(\text{green traffic light} \vee \text{green left-turn light}) \rightarrow \diamond \neg \text{stop},$$

$$\Phi_8 = \square(\neg \text{green traffic light} \rightarrow \diamond \text{stop}),$$

$$\Phi_9 = \square(\text{car from left} \rightarrow \neg(\text{turn left} \vee \text{turn right})),$$

$$\Phi_{10} = \square(\text{green traffic light} \rightarrow \diamond \neg \text{stop}),$$

$$\Phi_{11} = \square((\text{turn right} \wedge \neg \text{green traffic light}) \rightarrow \neg \text{car from left}),$$

$$\Phi_{12} = \square((\text{turn left} \wedge \neg \text{green left-turn light}) \rightarrow (\neg \text{car from right} \wedge \neg \text{car from left} \wedge \neg \text{opposite car})),$$

$$\Phi_{13} = \square((\text{stop sign} \wedge \neg \text{car from left} \wedge \neg \text{car from right}) \rightarrow (\diamond \neg \text{stop})),$$

$$\Phi_{14} = \square((\text{go straight} \rightarrow \neg \text{pedestrian in front})),$$

$$\Phi_{15} = \square((\text{turn right} \wedge \text{stop sign}) \rightarrow \neg \text{car from left}).$$

**Example on Controller Construction and Verification: Left-Turn.** We provide an example of the controllers for the task “turn left at the traffic light” and a model representing the traffic light environment. When implemented in the model, we can verify whether the controllers satisfy the provided specifications.

The responses from the language model before fine-tuning:

```

1 Steps for "turn left at traffic light"
2 1. Approach the traffic light with a left-turn light.
3 2. Wait for the left-turn light to turn green.
4 3. When the left-turn light turns green, wait for oncoming traffic to clear before turning left.
5 4. Turn left and proceed through the intersection.
    
```

The responses from the language model after fine-tuning:

```

1 Steps for "turn left at traffic light"
2 1. Approach the traffic light and observe the left turn light.
3 2. If the left turn light is not green, then stop.
4 3. If the left turn light is green, then turn left.
    
```

We present constructed controllers in Figure 18.

We implement both controllers in the model presented in Figure 15, i.e., get the product automaton of the controller and the model. Then, we can verify both controllers against the 15 specifications.

The controller obtained before fine-tuning fails specification  $\Phi_{12}$ , while the one after fine-tuning passes all the specifications.

## D VERIFICATION USING NUSMV

```

1  MODULE turn_left_before_finetune
2
3
4  VAR
5    green_traffic_light : boolean;
6    green_left_turn_light : boolean;
7    opposite_car : boolean;
8    car_from_left : boolean;
9    car_from_right : boolean;
10   pedestrian_at_left : boolean;
11   pedestrian_at_right : boolean;
12   side_car : boolean;
13   stop_sign : boolean;
14   action : {stop, turn_left, turn_right, go_straight};
15
16  ASSIGN
17    init(action) := stop;
18
19  TRANS
20    case
21      !green_left_turn_light : next(action) = stop;
22      green_left_turn_light & !opposite_car & !car_from_left & !car_from_right & !pedestrian_at_left
23      & !pedestrian_at_right : next(action) = turn_left;
24      opposite_car | car_from_left | car_from_right | pedestrian_at_left | pedestrian_at_right :
25      next(action) = stop;
26      action = turn_left : next(action) = go_straight;
27      TRUE : next(action) = stop;
28    esac;
29
30  MODULE turn_left_after_finetune
31
32  VAR
33    green_traffic_light : boolean;
34    green_left_turn_light : boolean;
35    opposite_car : boolean;
36    car_from_left : boolean;
37    car_from_right : boolean;
38    pedestrian_at_left : boolean;
39    pedestrian_at_right : boolean;
40    side_car : boolean;
41    stop_sign : boolean;
42    action : {stop, turn_left, turn_right, go_straight};
43
44  ASSIGN
45    init(action) := stop;
46
47  TRANS
48    case
49      !green_left_turn_light : next(action) = stop;
50      green_left_turn_light : next(action) = turn_left;
51    esac;
52
53  LTLSPEC NAME sample_phi_1 :=
54    G( action=turn_left -> (left_turn_light in {flashing, green}) );
55
56  LTLSPEC NAME sample_phi_2 :=
57    G( F action=turn_left );

```

```

1  MODULE turn_right_before_finetune
2
3  VAR
4    green_traffic_light : boolean;
5    green_left_turn_light : boolean;
6    opposite_car : boolean;
7    car_from_left : boolean;
8    car_from_right : boolean;
9    pedestrian_at_left : boolean;
10   pedestrian_at_right : boolean;
11   side_car : boolean;
12   stop_sign : boolean;
13   action : {stop, turn_left, turn_right, go_straight};
14
15  ASSIGN
16    init(action) := stop;
17    next(action) :=
18      case
19        green_traffic_light & !car_from_right : {go_straight, turn_right};
20        green_traffic_light & car_from_right : go_straight;
21        !green_traffic_light : stop;
22      esac;

```

```

23 |
24 | MODULE turn_right_after_finetune
25 | VAR
26 |   green_traffic_light : boolean;
27 |   green_left_turn_light : boolean;
28 |   opposite_car : boolean;
29 |   car_from_left : boolean;
30 |   car_from_right : boolean;
31 |   pedestrian_at_left : boolean;
32 |   pedestrian_at_right : boolean;
33 |   side_car : boolean;
34 |   stop_sign : boolean;
35 |   action : {stop, turn_left, turn_right, go_straight};
36 | ASSIGN
37 |   init(action) := stop;
38 | TRANS
39 |   case
40 |     !car_from_left & !pedestrian_at_right : next(action) = turn_right;
41 |     car_from_left | pedestrian_at_right : next(action) = stop;
42 |   esac
43 |
44 | LTLSPEC NAME sample_phi_1 :=
45 |   G( pedestrian_at_right -> ! action=turn_right );
46 |
47 | LTLSPEC NAME sample_phi_2 :=
48 |   G( car_from_left -> ! action=turn_right );

```

```

1 | #!NuSMV -source
2 | read_model -i right_turn.smv # file name
3 | go
4 |
5 | check_ltlspec -P "phi_1" -o result1.txt
6 |
7 | check_ltlspec -P "phi_2" -o result2.txt
8 |
9 | quit

```

## E FINE-TUNING LLAMA-2: IMPLEMENTATION DETAILS

**Llama-2 Prompts** Llama-2 has particular implementation requirements for the prompt. Certain tokens are required which delineate system and user messages. System messages describe what role the language model should act as, while user messages describe the task at hand. We use the following prompt for Llama-2, where italicized symbols represent special input tokens, and the last sentence is a given task:

$\langle s \rangle$  [*INST*]  $\langle\langle$  *SYS*  $\rangle\rangle$

You are a helpful assistant. Always answer as helpfully as possible, while being safe. Your answers should be detailed.  $\langle\langle$  /*SYS*  $\rangle\rangle$

Steps for “turn right at traffic light”: [/*INST*]

**Fine-tuning Efficiency** Due to hardware limitations, fine-tuning all model parameters is impractical. Instead, it is possible to fine-tune a low-rank approximation of a given matrix within the model (Hu et al., 2021). For example, instead of updating a matrix  $W \in \mathcal{R}^{d \times d}$ , it is more memory efficient to update two matrices  $A \in \mathcal{R}^{d \times k}$ ,  $B \in \mathcal{R}^{k \times d}$  with  $k \ll d$ , by holding  $W$  constant and defining  $\tilde{W} = W + AB$ . Then,  $A$  and  $B$  can be updated using gradient descent with a smaller memory profile than updating  $W$  itself, since the combined number of parameters in  $A$  and  $B$  is much less than  $W$ .