LTL Satisfaction as Language Containment

Recall that, given a Kripke structure \( A = (A, \rightarrow_A, \models_A) \) on atomic propositions \( \Pi \), and choosing an initial state \( a \in A \) and an LTL formula \( \varphi \in LTL(\Pi) \), the satisfaction relation is defined by the chain of equivalences:

\[
A, a \models_{LTL} \varphi \iff \forall \pi \in \text{Paths}(A)_a \pi \models_{LTL} \varphi \iff \forall \pi \in \text{Paths}(\mathcal{K})_a \pi; \tilde{\Pi}_A \models_{LTL} \varphi.
\]

If we define the set \( \text{Traces}(A)_a \) of the traces of \( A \) from initial state \( a \) as:

\[
\text{Traces}(A)_a = \{ \pi; \tilde{\Pi}_A | \pi \in \text{Paths}(A)_a \},
\]

we can rephrase the above definition of satisfaction as the simpler equivalence:

\[
A, a \models_{LTL} \varphi \iff \forall \tau \in \text{Traces}(A)_a \tau \models_{LTL} \varphi.
\]

But we can view \( \text{Traces}(A)_a \) as a language of infinite words on the alphabet \( \mathcal{P}(\Pi) \). Specifically, an infinite word on an alphabet \( \Lambda \) is just a function \( \tau : \mathbb{N} \rightarrow \Lambda \), where we suggestively denote \([\mathbb{N} \rightarrow \Lambda]\) as \( \Lambda^\omega \) (\( \omega \) denotes the set of natural numbers viewed as an “ordinal set” with its \( < \) order), to emphasize that this is the language of infinite words on \( \Lambda \), just as \( \Lambda^* \) is the language of finite words on \( \Lambda \). Therefore, we have the language containment: \( \text{Traces}(A)_a \subseteq \mathcal{P}(\Pi)^\omega \).

Now observe that the relation \( \tau \models_{LTL} \varphi \) between an infinite word \( \tau \in \mathcal{P}(\Pi)^\omega \) and an LTL formula \( \varphi \in LTL(\Pi) \) is defined independently of any Kripke structure, since the inductive semantic definition of \( \tau \models_{LTL} \varphi \) is given in terms of the syntactic structure of \( \varphi \) and can be expressed in terms of traces, regardless of where such traces come from. Therefore, an LTL formula \( \varphi \in LTL(\Pi) \) also defines a language of infinite words, namely, the set of all traces \( \tau \) that satisfy \( \varphi \). Call this language \( L(\varphi) \) the language of \( \varphi \), i.e., \( L(\varphi) = \{ \tau \in \mathcal{P}(\Pi)^\omega | \tau \models_{LTL} \varphi \} \).

Using this notation, we can express the satisfaction relation \( A, a \models_{LTL} \varphi \) in an even simpler, language-theoretic way by the equivalence:

\[
A, a \models_{LTL} \varphi \iff \text{Traces}(A)_a \subseteq L(\varphi).
\]

The intuitive meaning is that, semantically, the property \( \varphi \) specifies a set of allowable traces, so \( A \) starting at \( a \) satisfies property \( \varphi \) iff all its traces are among those allowed by \( \varphi \).

Büchi Automata and Decidability of \( \omega \)-Regular Languages

Recall that regular languages are languages recognized by finite automata; and that Boolean operations on such languages, such as union, intersection and complement, as well as properties such as language containment or language emptiness, can be effectively computed, resp. decided, by means of automata. Thanks to the work of the Swiss mathematician Richard Büchi, finite automata on an input alphabet \( \Lambda \) can also recognize \( \omega \)-regular languages as subsets of the set...
The definition of a finite automaton\(^1\) \(B\) on an input alphabet \(\Lambda\) remains the same: we specify its input alphabet, finite set of states, initial state (or set of initial states), \(\Lambda\)-labeled transition relation, and a subset of final/accepting states. The only thing that changes is the notion of acceptance. A finite word \(w \in \Lambda^*\) is accepted by an automaton \(B\) if the input \(w\) can reach a state in the set \(F\) of accepting states of \(A\). Instead, \(B\) will accept an infinite word \(\tau \in \Lambda^\omega\) iff \(F \cap \text{inf}_B(\tau) \neq \emptyset\), where \(\text{inf}_B(\tau) = \{b \in B \mid \{n \in \mathbb{N} \mid \text{init } \stackrel{\tau|_{\leq n}}{\rightarrow}^*_{B} b\} \congm \mathbb{N}\}\), where \(\tau|_{\leq n}\) denotes the finite word \(\tau(0)\ldots\tau(n)\), and where \(\congm\) is the equinumerosity (sometimes also called equicardinality) equivalence relation on sets (see §8 of STACS), so that \(\{b \in B \mid \{n \in \mathbb{N} \mid \text{init } \stackrel{\tau|_{\leq n}}{\rightarrow}^*_{B} b\} \congm \mathbb{N}\) just means that \(\{b \in B \mid \{n \in \mathbb{N} \mid \text{init } \stackrel{\tau|_{\leq n}}{\rightarrow}^*_{B} b\}\) is an infinite set. Therefore, \(\text{inf}^B(\tau)\) is the set of states of \(B\) that are visited infinitely often by the infinite input \(\tau\). Although automata remain the same, when this new interpretation of input acceptance is given to them, they are called \(\text{B"uchi automata}\), in honor of Richard B"uchi.

For our current purposes, we just need to use two facts about \(\text{B"uchi automata}\) and \(\omega\)-regular languages: (1) \((\text{Language Intersection})\) If two \(\omega\)-regular languages, \(L_1\) and \(L_2\) on \(\Lambda\) are respectively recognized by \(\text{B"uchi automata}\) \(B_1\) and \(B_2\), then their intersection \(L_1 \cap L_2\) is also an \(\omega\)-regular language recognized by a \(\text{B"uchi automaton}\) \(B_1 \circ B_2\) called the \(\text{synchronous product}\) of \(B_1\) and \(B_2\) (see §9.2 of [1] for a detailed construction of \(B_1 \circ B_2\)). (2) \((\text{Language Emptiness})\) Given a \(\text{B"uchi automaton}\) \(B\), there is an algorithm to effectively decide whether the language \(\mathcal{L}(B)\) recognized by \(B\) is empty or not. Specifically, the procedure deciding the \(\omega\)-regular language emptiness problem answers “empty” when \(\mathcal{L}(B)\) is empty, but in case \(\mathcal{L}(B)\) is non-empty, it effectively computes\(^2\) a witness \(\tau \in \mathcal{L}(B)\) proving its non-emptiness.

\section*{Model Checking LTL Properties with \(\text{B"uchi Automata}\)}

We now have almost all the ingredients needed to obtain a model checking \textit{decision procedure} for deciding the LTL satisfaction problem \(\mathcal{A}, a \models \varphi\) in case the set of states \(\text{Reach}_\mathcal{A}(a)\) reachable from \(a\) is \textit{finite}, except for two remaining technical details.

First, we need to associate to \((\mathcal{A}, a)\) a \(\text{B"uchi automaton}\) \(\mathcal{B}(\mathcal{A}, a)\) such that \(\mathcal{L}(\mathcal{B}(\mathcal{A}, a)) = \text{Traces}(\mathcal{A})_a\). This is easy: we can build \(\mathcal{B}(\mathcal{A}, a)\) with input alphabet \(P(\Pi)\) so that it exactly mimics the behavior of \(\mathcal{A}\) from the initial state \(a\) as follows: (1) its set of states \textit{and} its set of accepting states are both \(\{i\} \cup \text{Reach}_\mathcal{A}(a)\), (2) its initial state is the new added state \(i\), and (3) its labeled transition relation is the union:

\[\{i \stackrel{L(a)}{\rightarrow} a\} \cup \{b \stackrel{L(c)}{\rightarrow} c \mid b, c \in \text{Reach}_\mathcal{A}(a) \land b \rightarrow_{K} c\}\].

The equality \(\mathcal{L}(\mathcal{B}(\mathcal{A}, a)) = \text{Traces}(\mathcal{A})_a\) follows trivially from this construction, since there is a one-to-one correspondence between the infinite executions of \(\mathcal{A}\) from \(a\) and the infinite computations of \(\mathcal{B}(\mathcal{A}, a)\) having the exact same traces by construction.

Second, we need to observe the fact that the language \(\mathcal{L}(\varphi)\) is \(\omega\)-regular. This is because \(\mathcal{L}(\varphi)\) is the language recognized by a \(\text{B"uchi automaton}\) \(\mathcal{B}_{\varphi}\) that can be effectively constructed

\(^1\)See Def. 5 in §7.2 of STACS, where \(\Lambda\) is denoted \(L\) and is called the labeled set. But here we need two more pieces of information: In STACS, \(B\) is a triple \(B = (B, \Lambda, \rightarrow_B)\), with \(B\) a finite set; but here \(B\) is a 5-tuple \(B = (B, \text{init}, \Lambda, \rightarrow_B, F)\), with \(\text{init} \in B\) the initial state, and \(F \subseteq B\) the set of accepting states.

\(^2\)The reader might wonder how \(\tau\), being an infinite object, can be effectively specified. The reason is that the set \(B\) of states is \textit{finite}. Therefore, \(\tau\), viewed as an infinite path on a finite graph, will necessarily have \textit{cycles}, allowing a \textit{finite} cycle description of \(\tau\).
from the LTL formula $\varphi$. Since the details of the construction $\varphi \rightarrow B \varphi$ are rather involved, I refer to Section 9.4 of [1] (or, alternatively, to Section 6.8 of [4]), where this construction is described in full detail.

We are now ready to prove the main theorem of this Appendix:

**Theorem** (Decidability of LTL Model Checking). When the set of states $\text{Reach}_A(a)$ reachable from state $a$ is finite, the LTL satisfaction problem $A, a \models_{\text{LTL}} \varphi$ is decidable. Furthermore, when $A, a \models_{\text{LTL}} \varphi$, the decision procedure returns a (finite representation of) a trace $\tau \in \text{Traces}(A)_a$ such that $\tau \models_{\text{LTL}} \varphi$.

**Proof:** Since we have the equivalence $A, a \models_{\text{LTL}} \varphi \iff \text{Traces}(A)_a \subseteq \mathcal{L}(\varphi)$, we just need to have a decision procedure for effectively checking the set containment $\text{Traces}(A)_a \subseteq \mathcal{L}(\varphi)$. But this is equivalent to checking the emptiness problem $\text{Traces}(A)_a \cap \mathcal{L}(\varphi)^c = \emptyset$, where $\mathcal{L}(\varphi)^c$ denotes the complement of $\mathcal{L}(\varphi)$ in $P(\Pi)^c$. But by the semantic definition $\tau \models_{\text{LTL}} \neg \varphi \iff \text{def} \tau \models_{\text{LTL}} \varphi$, we have the language identity $\mathcal{L}(\varphi)^c = \mathcal{L}(\neg \varphi)$. So we just need a decision procedure for the emptiness problem $\text{Traces}(A)_a \cap \mathcal{L}(\neg \varphi) = \emptyset$. But this is just the emptiness problem $\mathcal{L}(B(A, a)) \cap \mathcal{L}(B_{\neg \varphi}) = \emptyset$; that is, the Büchi automata language emptiness problem $\mathcal{L}(B(A, a) \otimes B_{\neg \varphi}) = \emptyset$, which is decidable and returns a “witness trace” (a counterexample) $\tau \in \text{Traces}(A)_a$ in such a language intersection if the intersection is non-empty, as desired. $\square$

**Further Reading**

The already cited Chapter 9 of [1] contains a detailed description of all the concepts presented here. In particular, Section 9.5 describes an on the fly LTL model checking algorithm to efficiently decide the emptiness problem $\mathcal{L}(B(A, a) \otimes B_{\neg \varphi}) = \emptyset$ using double depth first search. This is the explicit-state model checking algorithm used by both the Spin model checker [3] and the Maude LTL model checker [2]. Another useful reference for the automata-theoretic approach to model checking is provided in Chapters 5 and 6 of [4].

**References**


