

CDS 270 (Fall 09) - Lecture Notes for Assignment 8.

Because this part of the course has no slides or textbook, we will provide lecture supplements that include, hopefully, enough discussion to complete the exercises. Contact Ufuk or Andy if you have questions.

1 Introduction

This set of lecture notes gives a quick tutorial on regular languages and their applications to model checking. Linear Temporal Logic model checking is also presented.

Regular languages have nice properties that can be exploited in model checking.

All definitions used in these preliminaries are taken from [1].

2 Regular Languages

This section gives the definitions and some of the basic results for finite automata and regular languages. See [5] for a good introduction.

Definition 1. A *nondeterministic finite automaton* (NFA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ where

- Q is a finite set of *states*.
- Σ is a finite set called an *alphabet*.
- $\delta : Q \times \Sigma \rightarrow 2^Q$ is a *transition function*. Often we use the relation notation, $q \xrightarrow{A} q'$, to denote the fact that $q' \in \delta(q, A)$.
- $Q_0 \subseteq Q$ is a set of *initial* states.
- $F \subseteq Q$ is a set of *accepting* states.

Definition 2. Let $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ be an NFA and let $w = A_1 A_2 \dots A_n \in \Sigma^*$ be a finite string. A *run* for w in \mathcal{A} is a finite sequence of states, $q_0 q_1 \dots q_n$ such that

- $q_0 \in Q_0$
- $q_0 \xrightarrow{A_1} q_1 \xrightarrow{A_2} q_2 \xrightarrow{A_3} \dots \xrightarrow{A_n} q_n$

A run is *accepting* if $q_n \in F$. The string, $w \in \Sigma^*$, is *accepted* by \mathcal{A} if there is an accepting run of w in \mathcal{A} . The language *accepted* by \mathcal{A} is the set of strings, $\mathcal{L}(\mathcal{A})$, defined as

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* : \text{there is an accepting run of } w \text{ in } \mathcal{A}\}.$$

Definition 3. A set of finite strings $\mathcal{L} \subseteq \Sigma^*$ is called a *regular language* if there is a (nondeterministic) finite automaton \mathcal{A} such that $\mathcal{L} = \mathcal{L}(\mathcal{A})$.

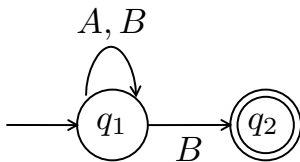


Figure 1: The NFA from Example 1. To signify that q_1 is an initial state, an arrow with no source is drawn to q_1 . To signify that q_2 is a final state, q_2 is drawn as a double circle. Transitions are given by arrows that are labeled by the corresponding letters.

Example 1. As with transition systems, we will typically represent finite automata pictorially by directed graphs. Consider the NFA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$, where

- $Q = \{q_1, q_2\}$,
- $\Sigma = \{A, B\}$,
- The transition function is given by

$$\begin{aligned} \delta(q_1, A) &= \{q_1\}, & \delta(q_2, A) &= \emptyset, \\ \delta(q_1, B) &= \{q_1, q_2\}, & \delta(q_2, B) &= \emptyset, \end{aligned}$$

- $Q_0 = \{q_1\}$,
- $F = \{q_2\}$.

The automaton is given pictorially in Figure 1.

The language accepted by \mathcal{A} is given by

$$\mathcal{L}(\mathcal{A}) = \{wB \in \{A, B\}^* : w \in \{A, B\}^*\}.$$

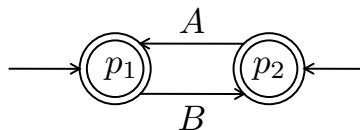


Figure 2: The automaton from Example 2.

Example 2. As another example, let $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ be the NFA given by

- $Q = \{p_1, p_2\}$,
- $\Sigma = \{A, B\}$,

- The transition function is given by

$$\begin{aligned}\delta(p_1, A) &= \{p_2\}, & \delta(p_2, A) &= \emptyset, \\ \delta(p_2, B) &= \emptyset, & \delta(p_1, B) &= \{p_1\},\end{aligned}$$

- $Q_0 = Q$,
- $F = Q$.

The NFA is depicted in Figure 2. In this case, \mathcal{A} accepts the language

$$\mathcal{L}(\mathcal{A}) = \{(AB)^n, (AB)^n A, (BA)^n, (BA)^n B : n \in \mathbb{N}\}$$

Theorem 1. *If \mathcal{L}_1 and \mathcal{L}_2 are regular languages (over the same alphabet Σ), then*

- $\mathcal{L}_1 \cup \mathcal{L}_2$ and
- $\mathcal{L}_1 \cap \mathcal{L}_2$

are regular languages.

If \mathcal{L} is a regular language over Σ , then so is $\Sigma^* \setminus \mathcal{L}$.

To see that $\mathcal{L}_1 \cup \mathcal{L}_2$ is a regular language, let $\mathcal{A}_1 = (Q^1, \Sigma, \delta^1, Q_0^1, F^1)$ and $\mathcal{A}_2 = (Q^2, \Sigma, \delta^2, Q_0^2, F^2)$ be automata that accept \mathcal{L}_1 and \mathcal{L}_2 , respectively. Let \mathcal{A} be the NFA defined by

$$\mathcal{A} = (Q^1 \sqcup Q^2, \Sigma, \delta, Q_0^1 \sqcup Q_0^2, F^1 \sqcup F^2),$$

where \sqcup denotes the disjoint union, and δ is defined by

$$\delta(q, A) = \begin{cases} \delta^1(q, A) & \text{if } q \in Q^1 \\ \delta^2(q, A) & \text{if } q \in Q^2 \end{cases}.$$

Then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$.

The intersection requires a slightly more complex construction, known as the product of two automata. We will see similar constructions later.

Definition 4. Let $\mathcal{A}_1 = (Q^1, \Sigma, \delta^1, Q_0^1, F^1)$ and $\mathcal{A}_2 = (Q^2, \Sigma, \delta^2, Q_0^2, F^2)$ be finite automata. Then their product is the automaton $\mathcal{A}_1 \otimes \mathcal{A}_2$ defined by

$$\mathcal{A}_1 \otimes \mathcal{A}_2 = (Q^1 \times Q^2, \Sigma, \delta, Q_0^1 \times Q_0^2, F^1 \times F^2),$$

where δ is defined as follows: If $q'_1 \in \delta^1(q_1, A)$ and $q'_2 \in \delta^2(q_2, A)$, then $\langle q'_1, q'_2 \rangle \in \delta(\langle q_1, q_2 \rangle, A)$. In other words $\delta(\langle q_1, q_2 \rangle, A) = \delta^1(q_1, A) \times \delta^2(q_2, A)$ (where $\emptyset \times S = \emptyset$ for any set S).

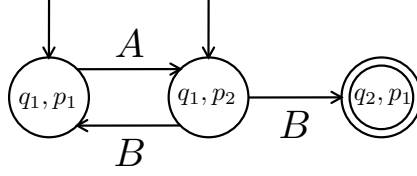


Figure 3: The product of the automata from Figures 1 and 2. As with transition systems, unreachable states are typically omitted.

It can be seen that $\mathcal{L}(\mathcal{A}_1 \otimes \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$, since $\langle q_0^1, q_0^2 \rangle \langle q_1^1, q_1^2 \rangle \dots \langle q_n^1, q_n^2 \rangle$ is a run of $\mathcal{A}_1 \otimes \mathcal{A}_2$ if and only if $q_0^1 q_1^1 \dots q_n^1$ is a run of \mathcal{A}_1 and $q_0^2 q_1^2 \dots q_n^2$ is a run of \mathcal{A}_2 . Further, given any string $w \in \Sigma^*$, there is an accepting run of w in $\mathcal{A}_1 \otimes \mathcal{A}_2$ if and only if there are accepting runs of w in both \mathcal{A}_1 and \mathcal{A}_2 .

Example 3. If \mathcal{A}_1 is the automaton from Example 1 and \mathcal{A}_2 is the automaton from Example 2, then their product is given in Figure 3.

Showing that the complement of a regular language is more involved still, and the standard technique involves showing that any regular language can be accepted by a deterministic finite automaton.

Definition 5. A finite automaton is called *deterministic* if $|Q_0| = 1$ and $|\delta(q, A)| = 1$ for all $q \in Q$ and all $A \in \Sigma$.

Recall that for any set, Ω , $|\Omega|$ is the number of elements that it contains.

Note that if $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ is a deterministic finite automaton, then the automaton $\neg\mathcal{A}$ defined by $\neg\mathcal{A} = (Q, \Sigma, \delta, Q_0, Q \setminus F)$ is such that $\mathcal{L}(\neg\mathcal{A}) = \Sigma^* \setminus \mathcal{L}(\mathcal{A})$.

Thus the proof that the complement of regular language is also a regular language can be completed by proving the following theorem:

Theorem 2. *If \mathcal{L} is a regular language, then there exists a deterministic finite automaton \mathcal{A} such that $\mathcal{L} = \mathcal{L}(\mathcal{A})$.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ be an NFA that accepts \mathcal{L} . Let $\hat{\mathcal{A}} = (2^Q, \Sigma, \hat{\delta}, \{Q_0\}, \hat{F})$ be the finite automaton constructed from \mathcal{A} , where

- $\hat{\delta}(\hat{Q}, A) = \{\bigcup_{q \in \hat{Q}} \delta(q, A)\}$ and
- $\hat{F} = \{\hat{Q} \subseteq Q : \hat{Q} \cap F \neq \emptyset\}$.

Then $\hat{\mathcal{A}}$ is a deterministic finite automaton such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\hat{\mathcal{A}})$. □

Example 4. Figure 4 depicts a deterministic automaton that accepts the same language as the automaton from Example 1.

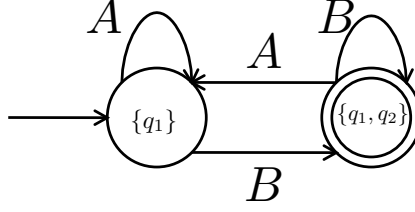


Figure 4: A deterministic automaton that accepts the same language as the NFA from Example 1. It was derived from the construction in the proof of Theorem 2. Again, unreachable states are omitted.

3 Regular Safety

Definition 6. A safety property $P_{safe} \subseteq (2^{\text{AP}})^\omega$ is called a *regular safety* property if $\text{BadPref}(P_{safe})$ is a regular language over 2^{AP} .

Note that any invariant is a regular safety property.

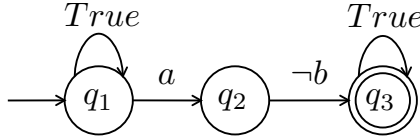


Figure 5: An NFA that accepts the set of bad prefixes for the safety property from Example 5. Note that we used a common shorthand for automata over 2^{AP} by labeling each transition with propositional formula, Φ , that represents all subsets $A \in 2^{\text{AP}}$ such that $A \models \Phi$. For instance a represents the subsets $\{a\}$ and $\{a, b\}$, while True represents all the subsets. Without this formula notation, the transitions would be cluttered with many different labels.

Example 5. Consider the following safety property of over $\text{AP} = \{a, b\}$:

$$P_{safe} = \{A_0A_1A_2 \dots \in (2^{\text{AP}})^\omega : \text{for all } j \geq 0, \text{ if } a \in A_j \text{ then } b \in A_{j+1}\},$$

with bad prefixes

$$\text{BadPref}(P_{safe}) = \{A_0A_1 \dots A_n \in (2^{\text{AP}})^* : a \in A_j \text{ but } b \notin A_{j+1} \text{ for some } j < n\}.$$

A finite automaton that accepts $\text{BadPref}(P_{safe})$ is given in Figure 5.

Example 6. The vending machine safety property over $\text{AP} = \{\text{coin}, \text{drink}\}$ with

$$P_{safe} = \{A_0A_1A_2 \dots \in (2^{\text{AP}})^\omega : \forall j \geq 0, |\{i \leq j : \text{coin} \in A_i\}| \geq |\{i \leq j : \text{drink} \in A_i\}|\},$$

and

$$\text{BadPref}(P_{safe}) = \left\{ A_0A_1 \dots A_n \in (2^{\text{AP}})^* : \left. \begin{array}{l} |\{i \leq j : \text{coin} \in A_i\}| < |\{i \leq j : \text{drink} \in A_j\}| \\ \text{for some } j \geq 0 \end{array} \right\} \right\}.$$

is not a regular safety property, since $BadPref(P_{safe})$ is not a regular language (see [5]). Roughly speaking, $BadPref(P_{safe})$ is not a regular language because any machine that accepts it requires some sort of counter, but counters cannot be constructed with finite automata.

Regular languages are useful in model checking because of the following argument. Say TS is a transition system and P_{safe} is a regular safety property, both over AP. Let \mathcal{A} be an NFA such that $\mathcal{L}(\mathcal{A}) = BadPref(P_{safe})$. Then

$$\begin{aligned}
TS \models P_{safe} & \text{ iff } Traces(TS) \subseteq P_{safe} \\
& \text{ iff } Traces(TS) \cap ((2^{AP})^\omega \setminus P_{safe}) = \emptyset \\
& \text{ iff } Traces(TS) \cap BadPref(P_{safe}) \cdot (2^{AP})^\omega = \emptyset \\
& \text{ iff } pref(Traces(TS)) \cap BadPref(P_{safe}) = \emptyset \\
& \text{ iff } pref(Traces(TS)) \cap \mathcal{L}(\mathcal{A}) = \emptyset,
\end{aligned}$$

where $pref(Traces(TS))$ are the finite prefixes of $Traces(TS)$.

Thus model checking regular languages is reduced to checking to see if finite path fragments give rise to bad prefixes. In order to check this, we utilize a product construction closely related to the product construction for finite automata.

Definition 7. Let $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system (with no terminal states) and let $\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$ be a finite automaton (with $Q \cap F = \emptyset$). Then the product of TS and \mathcal{A} is the transition system $TS \otimes \mathcal{A}$ defined by

$$TS \otimes \mathcal{A} = (S', Act, \rightarrow', I', AP', L'),$$

where

- $S' = S \times Q$
- \rightarrow' is defined by the following rule: If $s \xrightarrow{\alpha} t$ (with $s, t \in S$) and $q \xrightarrow{L(t)} p$ (with $q, p \in Q$), then $\langle s, q \rangle \xrightarrow{\alpha'} \langle t, p \rangle$.
- $I' = \{\langle s_0, q \rangle : s_0 \in I \text{ and } \exists q_0 \in Q_0 \text{ s.t. } q_0 \xrightarrow{L(s_0)} q\}$
- $AP' = Q$
- $L' : S \times Q \rightarrow 2^Q$ is given by $L'(\langle s, q \rangle) = \{q\}$.

Given a finite automaton \mathcal{A} , let $P_{inv(\mathcal{A})}$ be the invariant over 2^Q defined by

$$\begin{aligned}
P_{inv(\mathcal{A})} &= \left\{ A_0 A_1 A_2 \dots \in (2^Q)^\omega : \text{for all } j \geq 0, A_j \models \bigwedge_{q \in F} \neg q \right\} \\
&= \{A_0 A_1 A_2 \dots \in (2^Q)^\omega : \text{for all } j \geq 0, A_j \cap F = \emptyset\}.
\end{aligned}$$

Then, stepping through the definition of the product shows that $TS \otimes \mathcal{A} \models P_{inv(\mathcal{A})}$ if and only if $pref(Traces(TS)) \cap \mathcal{L}(\mathcal{A}) = \emptyset$. Therefore, if $\mathcal{L}(\mathcal{A}) = BadPref(P_{safe})$, then $TS \otimes \mathcal{A} \models P_{inv(\mathcal{A})}$ if and only if $TS \models P_{safe}$.

Thus, using finite automata, model checking regular safety properties can be reduced to model checking invariants, which as we saw in the last set of notes, can be done using a depth first search. Note, however, that the system we must check is now larger than the original system, so the complexity of the model checking algorithm depends on both the size of TS and the size of \mathcal{A} .

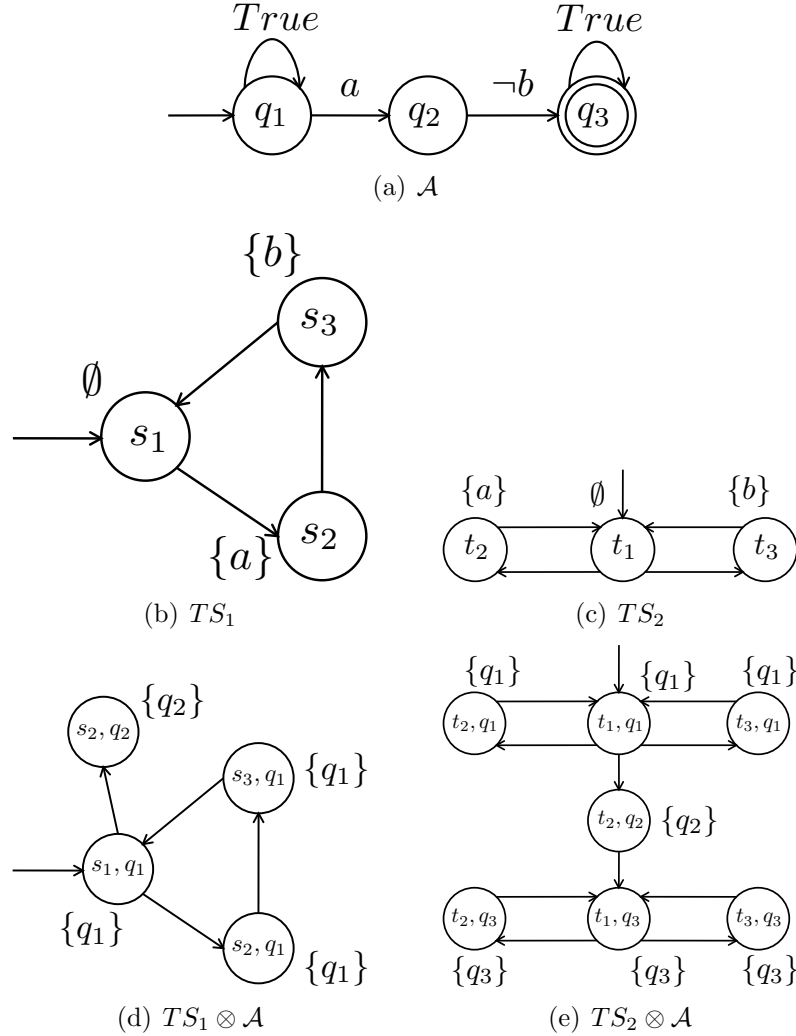


Figure 6: Two transition systems and their products with the automaton from Example 5. As usual, unreachable states are omitted. Furthermore, action labels are ignored, because they are not used in the verification. Note that $TS_1 \models P_{safe}$ but $TS_2 \not\models P_{safe}$, as shown by their products.

Example 7. Let \mathcal{A} be the automaton from Example 5. Figure 6 depicts two transition

systems and their products with \mathcal{A} . The product constructions show that $TS_1 \models P_{safe}$, while $TS_2 \not\models P_{safe}$, where P_{safe} is the safety property from Example 5.

Note how the product construction introduced a terminal state in $TS_1 \otimes \mathcal{A}$. As before, the terminal state could be removed by adding an extra “trap state,” but from the standpoint of verification, this is unnecessary. Indeed, the verification algorithm simply performs a depth first search through the graph, looking to see if an accept state is ever encountered. In this case, terminal states pose no problem.

4 ω -Regular Languages

Definition 8. A *nondeterministic Büchi Automaton* (NBA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$, where all components are defined the same as for nondeterministic finite automata. The only difference comes in what objects an NBA accepts.

Definition 9. Let $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ be an NBA and let $w = A_1A_2\dots \in \Sigma^\omega$ be an infinite string. A *run* for w in \mathcal{A} is an infinite sequence of states, $q_0q_1\dots$ such that

- $q_0 \in Q_0$
- $q_0 \xrightarrow{A_1} q_1 \xrightarrow{A_2} q_2 \xrightarrow{A_3} \dots$

A run is *accepting* if $q_j \in F$ for infinitely many j . The string, w , is *accepted* by \mathcal{A} if there is an accepting run of w in \mathcal{A} . The language *accepted* by \mathcal{A} is the set of infinite strings $\mathcal{L}_\omega(\mathcal{A})$ defined as

$$\mathcal{L}_\omega(\mathcal{A}) = \{w \in \Sigma^\omega : \text{there is an accepting run of } w \text{ in } \mathcal{A}\}.$$

Definition 10. A set of infinite strings $\mathcal{L}_\omega \subseteq \Sigma^\omega$ is called an *ω -regular language* if there is a (nondeterministic) Büchi automaton \mathcal{A} such that $\mathcal{L}_\omega = \mathcal{L}_\omega(\mathcal{A})$.

Definition 11. An NBA is called *deterministic* if $|Q_0| = 1$ and $|\delta(q, A)| = 1$ for all $q \in Q$ and all $A \in \Sigma$

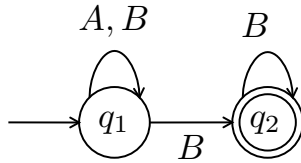


Figure 7: The NBA from Example 8. Note that NBAs are drawn exactly like NFAs (since all their parts are the same).

Example 8. Let \mathcal{A} be the NBA depicted in Figure 7. Then the language accepted by \mathcal{A} is given by

$$\mathcal{L}_\omega(\mathcal{A}) = \{A, B\}^* \cdot \{B^\omega\}.$$

That is, any finite string can occur in the beginning, but the tail is an infinite string of B s.

In sharp contrast with the case of finite strings, it can be shown that there is no deterministic Büchi automaton that accepts $\mathcal{L}_\omega(\mathcal{A})$ (see [1]).

Theorem 3. If $\mathcal{L}_{\omega,1}$ and $\mathcal{L}_{\omega,2}$ are ω -regular languages (over the same alphabet Σ), then

- $\mathcal{L}_{\omega,1} \cup \mathcal{L}_{\omega,2}$ and
- $\mathcal{L}_{\omega,1} \cap \mathcal{L}_{\omega,2}$

are ω -regular languages.

If \mathcal{L}_ω is an ω -regular language over Σ , then so is $\Sigma^\omega \setminus \mathcal{L}_\omega$.

The union works exactly the same as in the case of NFAs. Intersections and complements are harder.

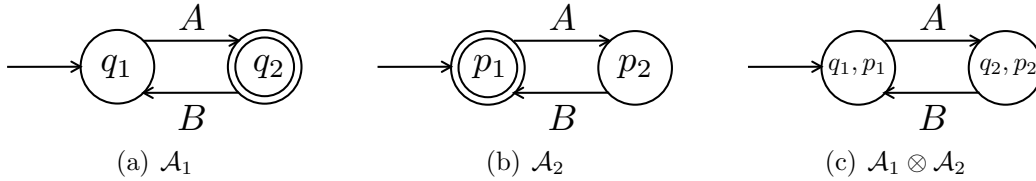


Figure 8: Two automata with $\mathcal{L}_\omega(\mathcal{A}_1) = \mathcal{L}_\omega(\mathcal{A}_2) = \{(AB)^\omega\}$, but $\mathcal{L}_\omega(\mathcal{A}_1 \otimes \mathcal{A}_2) = \emptyset$.

Example 9. Consider the automata depicted in Figure 8. They accept exactly the same ω -language, $\{(AB)^\omega\}$, but their product has no accepting states. This is because the accepting states of both automata cannot occur simultaneously.

Definition 12. Let $\mathcal{A}_1 = (Q^1, \Sigma, \delta^1, Q_0^1, F^1)$ and $\mathcal{A}_2 = (Q^2, \Sigma, \delta^2, Q_0^2, F^2)$ be NBAs. Define their ω -product (I don't think this is standard terminology, but ...) to be the automaton $\mathcal{A}_1 \otimes_\omega \mathcal{A}_2$ defined by

$$\mathcal{A}_1 \otimes_\omega \mathcal{A}_2 = (Q^1 \times Q^2 \times \{1, 2\}, \Sigma, \delta, Q_0^1 \times Q_0^2 \times \{1\}, F^1 \times Q^2 \times \{1\}),$$

where the transition function is defined by the following rules:

For any $q_1, q'_1 \in Q^1$, any $q_2, q'_2 \in Q^2$ and any $A \in \Sigma$,

- if $q_1 \notin F^1$, $q'_1 \in \delta^1(q_1, A)$, and $q'_2 \in \delta^2(q_2, A)$, then $\langle q'_1, q'_2, 1 \rangle \in \delta(\langle q_1, q_2, 1 \rangle, A)$
- if $q_1 \in F^1$, $q'_1 \in \delta^1(q_1, A)$, and $q'_2 \in \delta^2(q_2, A)$, then $\langle q'_1, q'_2, 2 \rangle \in \delta(\langle q_1, q_2, 1 \rangle, A)$

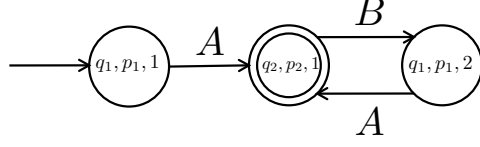


Figure 9: $\mathcal{A}_1 \otimes_\omega \mathcal{A}_2$ for the automata from Example 9. The ω -product accepts the correct language.

- if $q_2 \notin F^2$, $q'_1 \in \delta^1(q_1, A)$, and $q'_2 \in \delta^2(q_2, A)$, then $\langle q'_1, q'_2, 2 \rangle \in \delta(\langle q_1, q_2, 2 \rangle, A)$
- if $q_2 \in F^2$, $q'_1 \in \delta^1(q_1, A)$, and $q'_2 \in \delta^2(q_2, A)$, then $\langle q'_1, q'_2, 1 \rangle \in \delta(\langle q_1, q_2, 2 \rangle, A)$

By stepping through the definition, it can be shown that $\mathcal{L}_\omega(\mathcal{A}_1 \otimes_\omega \mathcal{A}_2) = \mathcal{L}_\omega(\mathcal{A}_1) \cap \mathcal{L}_\omega(\mathcal{A}_2)$.

Example 10. Recall \mathcal{A}_1 and \mathcal{A}_2 from Example 9. Their ω -product is shown in Figure 9. From the figure it can be seen that $\mathcal{L}_\omega(\mathcal{A}_1 \otimes_\omega \mathcal{A}_2) = \mathcal{L}_\omega(\mathcal{A}_1) \cap \mathcal{L}_\omega(\mathcal{A}_2) = \{(AB)^\omega\}$.

4.1 Remark on Complementation

Example 8 gives an ω -regular language that cannot be accepted by a deterministic Büchi automaton. This implies that a Büchi automaton that accepts the complement of an ω -regular language cannot be constructed using a determinization procedure, as in the case of regular languages. All known procedures for finding a Büchi automata that accepts the complement of an ω -regular language require sophisticated constructions beyond the scope of this course (see [2] or [3] for such procedures). As in the case of regular languages, the complement automata may be exponentially larger than the original automata.

5 Regular Properties

Definition 13. A property $P \subseteq (2^{\text{AP}})^\omega$ is called a *regular property* if P is an ω -regular language over 2^{AP} .

Note that if P is a regular property, then $(2^{\text{AP}})^\omega \setminus P$ is also a regular language. Thus, there is an NBA, \mathcal{A} such that $(2^{\text{AP}})^\omega \setminus P = \mathcal{L}_\omega(\mathcal{A})$.

Let TS be a transition system over AP. Then, similar to the case of regular safety, we have the following chain of equivalences:

$$\begin{aligned}
 TS \models P & \text{ iff } \text{Traces}(TS) \subseteq P \\
 & \text{ iff } \text{Traces}(TS) \cap ((2^{\text{AP}})^\omega \setminus P) = \emptyset \\
 & \text{ iff } \text{Traces}(TS) \cap \mathcal{L}_\omega(\mathcal{A}) = \emptyset.
 \end{aligned}$$

Definition 14. Let TS be a transition system and let \mathcal{A} be an NBA (both over AP). Then their product, $TS \otimes \mathcal{A}$, is the transition system defined in exactly the same way as for transition systems and finite automata (Definition 7).

Definition 15. A property $P_{pers} \subseteq 2^{AP}$ is called a *persistence* property if there is a propositional formula Φ , over AP, such that

$$P_{pers} = \{A_0A_1A_2 \dots \in (2^{AP})^\omega : \text{there exists } k \geq 0 \text{ s.t. for all } j \geq k, A_j \models \Phi\}$$

The formula Φ is called a persistence condition for P_{pers} .

If $\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$, let $P_{pers(\mathcal{A})}$ be the persistence property defined by the propositional formula (over Q) $\bigwedge_{q \in F} \neg q$.

Stepping through the definitions, one can show that $Traces(TS) \cap \mathcal{L}_\omega(\mathcal{A}) = \emptyset$ if and only if $TS \otimes \mathcal{A} \models P_{pers(\mathcal{A})}$ (since satisfying the persistence property corresponds to having all runs in \mathcal{A} only reach the set of accepting states finitely many times). Therefore $TS \models P$ if and only if $TS \otimes \mathcal{A} \models P_{pers(\mathcal{A})}$.

Thus, given an algorithm to check persistence properties, we have an algorithm to check any regular property. Keep in mind, however, that the complement automaton, \mathcal{A} may be exponentially larger than the automaton that accepts P .

Example 11. Let $AP = \{a\}$ and consider the property P defined by

$$P = \{A_0A_1A_2 \dots \in (2^{\{a\}})^\omega : \forall j \geq 0, a \in A_j\} \\ \cup \{A_0A_1A_2 \dots \in (2^{\{a\}})^\omega : a \notin A_j \text{ infinitely often}\}.$$

Then P is a regular property and $(2^{\{a\}})^\omega \setminus P$ is accepted by the automaton, \mathcal{A} , shown in Figure 10(a). Figure 10 shows two different transition systems over AP, as well as their products with \mathcal{A} . To see whether or not the transition systems satisfy P , one just needs to check that their product transition systems satisfy the persistence property defined by the formula $\neg q_3$. Inspection of the products shows that $TS_1 \models P$ but $TS_2 \not\models P$.

5.1 Checking Persistence Properties

Let $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system and let P_{pers} be a persistence property given by the propositional formula Φ . Note that TS violates the persistence property if and only if there is a state s such that

- s is reachable from I
- $L(s) \not\models \Phi$ and
- s is on a directed cycle.

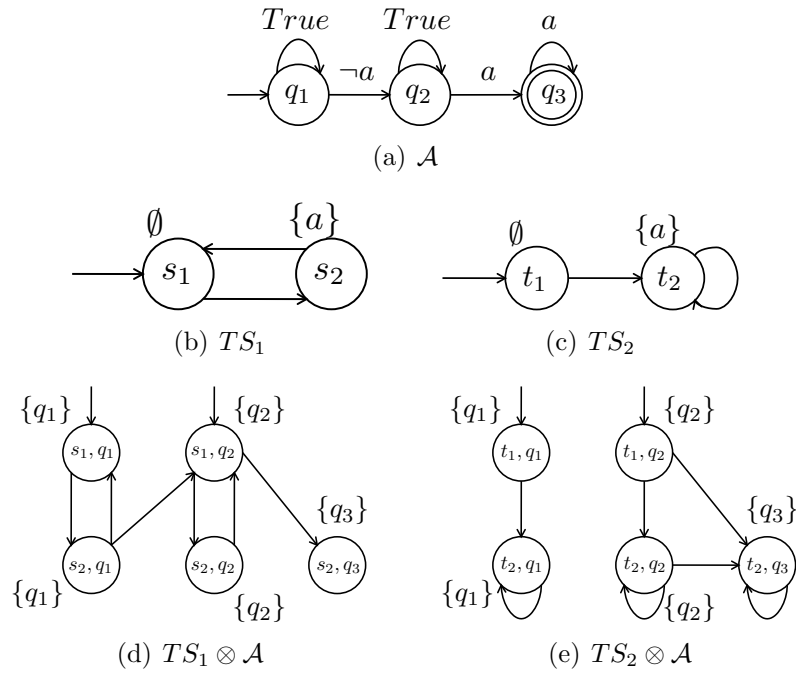


Figure 10: The automaton, \mathcal{A} and transition systems, TS_1 and TS_2 , for Example 11, along with their product transition systems. Note that since $F = \{q_3\}$, the product construction show that $TS_1 \models P$, since the only state of $TS_1 \otimes \mathcal{A}$ for which q_3 holds is terminal. On the other hand $TS_2 \not\models P$, since there is a self-loop on a state for which q_3 holds.

Indeed, since S is finite, in order for Φ to be violated infinitely often, there must be a state s that can be reached infinitely many times on a path. Thus s must lie on a directed cycle.

This observation suggests the following algorithm:

```

for each reachable state  $s$  do
  if  $L(s) \not\models \Phi$  then
    if there is a cycle from  $s$  to itself then
      return  $TS \not\models P_{pers}$  {Counterexample Found}
    end if
  end if
end for
return  $TS \models P_{pers}$  {No Counterexample Found}

```

When both the search for states to examine and the search for loops are handled by depth-first search, the above algorithm is called “nested depth-first search.” Since depth-first search is efficient (polynomial time in the size of TS), and there is only a single level of nesting, the nested depth-first search algorithm is also efficient.

6 Linear Temporal Logic

Linear temporal logic consists of propositional logic augmented by two temporal operators

- \bigcirc - the Next Operator.
- \mathcal{U} - the Until Operator.

Linear temporal logic (LTL) gives an expressive and intuitive way to specify high level properties of transition systems. We will see that LTL model checking can be reduced to the automata based model checking presented for regular properties.

Now we give a formal definition of linear temporal logic.

Definition 16. Let AP be a set of atomic propositions. The set of *linear temporal logic* (LTL) formulas is inductively constructed from the following rules:

- *True* is an LTL formula.
- If $a \in \text{AP}$, then a is an LTL formula.
- If Φ is an LTL formula, then so is $\neg\Phi$.
- If Φ_1 and Φ_2 are LTL formulas, then so is $\Phi_1 \wedge \Phi_2$.

- If Φ is an LTL formula, then so is $\bigcirc\Phi$.
- If Φ_1 and Φ_2 are LTL formulas, then so is $\Phi_1\mathcal{U}\Phi_2$.
- Nothing else is an LTL formula.

Rather than being interpreted over subsets of AP (i.e. elements of 2^{AP}), as in propositional logic, LTL formulas are interpreted on infinite strings over 2^{AP} .

Definition 17. Let $\sigma = A_0A_1A_2\dots \in (2^{\text{AP}})^\omega$. The satisfaction relation is defined inductively on LTL formulas as follows:

- $\sigma \models \text{True}$
- $\sigma \models a$ iff $a \in A_0$ (for $a \in \text{AP}$)
- $\sigma \models \neg\Phi$ iff $\sigma \not\models \Phi$
- $\sigma \models \Phi_1 \wedge \Phi_2$ iff $\sigma \models \Phi_1$ and $\sigma \models \Phi_2$
- $\sigma \models \bigcirc\Phi$ iff $A_1A_2\dots \models \Phi$
- $\sigma \models \Phi_1\mathcal{U}\Phi_2$ iff there exists $j \geq 0$ s.t. $A_jA_{j+1}\dots \models \Phi_2$ and for all i such that $0 \leq i < j$, $A_iA_{i+1}\dots \models \Phi_1$.

Given an LTL formula, Φ , the linear-time property induced by Φ is

$$\text{Words}(\Phi) = \{\sigma \in (2^{\text{AP}})^\omega : \sigma \models \Phi\}.$$

Definition 18. If TS is a transition system and Φ is an LTL formula (both over AP), then TS satisfies Φ , denoted $TS \models \Phi$, if $\text{Traces}(TS) \subseteq \text{Words}(\Phi)$.

Intuitively, $\bigcirc\Phi$ means that Φ holds in the next step. Also, $\Phi_1\mathcal{U}\Phi_2$ means that at some point Φ_2 holds, and Φ_1 holds for all times leading up to that point.

As before, the propositional operators \vee , \Rightarrow , \Leftrightarrow , and \oplus can be derived from the operators \neg and \wedge . We will also be interested in two more temporal operators: \diamond - the Eventually Operator, and \square - the Globally Operator.

The \diamond operator is defined as

$$\diamond\Phi \equiv \text{True}\mathcal{U}\Phi$$

and is interpreted as “eventually Φ holds.”

The \square operator is defined as

$$\square\Phi \equiv \neg\diamond\neg\Phi$$

and is interpreted as “ Φ always holds.”

With these operators, several important types of formulas can be built.

Example 12. If P_{inv} is an invariant, with invariant condition Φ , then $P_{inv} = Words(\Box\Phi)$.

Example 13. If P_{pers} is a persistence property, with persistence condition Φ , then $P_{pers} = Words(\Diamond\Box\Phi)$

Example 14. The safety property from Example 5,

$$P_{safe} = \{A_0A_1A_2\dots \in (2^{AP})^\omega : \text{for all } j \geq 0, \text{ if } a \in A_j \text{ then } b \in A_{j+1}\},$$

is induced by the LTL formula $\Box(a \Rightarrow \bigcirc b)$.

Example 15. The property from Example 11,

$$P = \{A_0A_1A_2\dots \in (2^{\{a\}})^\omega : \forall j \geq 0, a \in A_j\} \\ \cup \{A_0A_1A_2\dots \in (2^{\{a\}})^\omega : a \notin A_j \text{ infinitely often}\}.$$

is induced by the formula the LTL $(\Box a) \vee (\Box\Diamond\neg a)$.

Theorem 4. *There exists an algorithm that takes an LTL formula, Φ , as an input and returns a Büchi automaton \mathcal{A} such that*

$$Words(\Phi) = \mathcal{L}_\omega(\mathcal{A}).$$

So, in particular, we see that $Words(\Phi)$ is always an ω -regular language. One should note, however, that the automaton \mathcal{A} (with size measured as number of states), may be exponentially larger than the formula Φ (where the size of Φ is measured as the number of operators in Φ). See [1] for more details.

Example 16. The converse to Theorem 4 does not hold. Indeed for $AP = \{a\}$ let P be the following linear-time property:

$$P = \{A_0A_1A_2\dots \in (2^{\{a\}})^\omega : a \in A_j \text{ for all even } j\}.$$

Then P is a regular property, but there is no LTL formula that induces P . See [8] for a proof.

As before, the relation to model checking comes from the realization that

$$TS \models \Phi \quad \text{iff} \quad Traces(TS) \subseteq Words(\Phi) \\ \text{iff} \quad Traces(TS) \cap Words(\neg\Phi) = \emptyset.$$

Combining this idea with the fact that LTL formulas always induce ω -regular languages, gives the following simple algorithm for LTL model checking:

- Given a transition system TS and an LTL formula Φ
- Construct an NBA, \mathcal{A} such that $\mathcal{L}_\omega(\mathcal{A}) = Words(\Phi)$
- Construct $TS \otimes \mathcal{A}$
- Use the persistence checking algorithm check if $TS \otimes \mathcal{A} \models P_{pers(\mathcal{A})}$
- Conclude that $TS \models \Phi$ if and only if $TS \otimes \mathcal{A} \models P_{pers(\mathcal{A})}$

The only place where exponential blowup can occur in this algorithm is in constructing \mathcal{A} . Typically, however, Φ is very small compared to TS , and this blowup is not nearly as problematic as simply writing down TS .

6.1 Model Checking Tools

Some LTL model checking tools include SPIN [6], NuSMV [4], and TLC (for the logical extension, TLA) [7] .

References

- [1] Christel Baier and Joost-Pieter Katoen. Principles of Model Checking (Representation and Mind Series). The MIT Press, 2008.
- [2] Felix Klaedtke. Complementation of Büchi automata using alternation. In E. Grädel, W. Thomas, and T. Wilke, editors, Automata, Logics, and Infinite Games (A Guide to Current Research), volume 2500 of Lecture Notes in Computer Science, chapter 4, pages 61–77. Springer-Verlag, 2002.
- [3] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. ACM Trans. Comput. Logic, 2(3):408–429, 2001.
- [4] <http://nusmv.irst.itc.it>.
- [5] M. Sipser. Introduction to the Theory of Computation. PWS Publishing Company, 1997.
- [6] <http://spinroot.com>.
- [7] <http://research.microsoft.com/en-us/um/people/lamport/tla/tla.html>.
- [8] Pierre Wolper. Temporal logic can be more expressive. In Foundations of Computer Science, 1981. SFCS '81. 22nd Annual Symposium on, pages 340–348, Oct. 1981.