CDS 270 (Fall 09) - Lecture Notes for Assignment 7.

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 assignment and the next few will run through a really bare bones presentation of automata based model checking. Model checking is a powerful (if rather brute force) method used extensively in industry for hardware and software verification. While model checking applications in computer science are relatively mature, more research is required to understand the potential and limitations of model checking ideas in control. Our aim in this part of the course is to explain some of the basic techniques used in current research at the intersection of model checking and control systems.

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

2 Transition System Basics

Definition 1. A *transition system* is a tuple $TS = (S, \text{Act}, \rightarrow, I, \text{AP}, L)$ such that

- S is a set of *states* (the state space)
- Act is a set of *actions*
- $\rightarrow \subseteq S \times \text{Act} \times S$ is a transition relation. $(s, \alpha, s') \in \rightarrow$ is denoted by $s \xrightarrow{\alpha} s'$
- $I \subseteq S$ is a set of *initial states* or *initial conditions*
- AP is a set of *atomic propositions*
- $L : S \rightarrow 2^{\text{AP}}$ is a *labeling function*

Often, transition systems are drawn as directed graphs with states represented by vertices and transitions represented by edges.

Example 1 (Traffic Light)**.** Consider a simplified model of a traffic light, in which all we care about is if the light is green or not:

$$TS_{L1} = (Q_{L1}, \text{Act}_{L1}, \rightarrow_{L1}, I_{L1}, A_{L1}, L_{L1})$$

with

- $Q_{L1} = \{q_1, q_2\}$
- $\text{Act}_{L1} = \{\alpha\}$
- $\rightarrow_{L1} = \{(q_1, \alpha, q_2), (q_2, \alpha, q_1)\}$
- $I_{L1} = \{q_1\}$
- $AP_{L1} = \{g_1\}$
- $L_{L1}(q_1) = \emptyset$ and $L_{L1}(q_2) = \{g_1\}$.

See Figure 1 for a pictorial representation of TS_{L1}. The label $L_{L1}(q_1) = \emptyset$ is interpreted to mean that if the system is in state q_1, then the light is not green. Whereas, $L_{L1}(q_2) = \{g_1\}$ indicates that the light is green when the system is in state q_2.

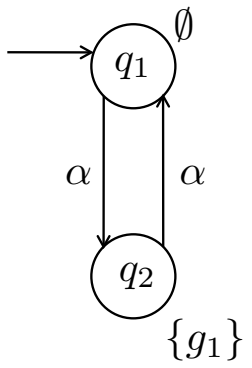

Figure 1: Pictorial representation of the traffic light transition system TS_{L1}. The initial condition, q_1, is specified by the incoming arrow which does not originate from another state. Edges are labeled with the corresponding actions, and the states are labeled by the subsets of $\text{AP}_{L1} = \{g_1\}$ as specified by the labeling function L_{L1}.

Example 2 (Logical Dynamical Systems)**.** For formal definitions of the propositional logic concepts used in this example, see Subsection 3.1.

Consider the following logical dynamical system with state variable x, input u, and output y:

$$\begin{aligned} x[k+1] &= x[k] \oplus u[k], \qquad x[0] = 0 \\ y[k] &= x[k], \end{aligned}$$

where $x[k], u[k] \in \{0, 1\}$ for all $k \geq 0$ and $\oplus$ denotes the XOR operator.

We model this dynamical system as transition system as
$TS_{LOG1} = (S_{LOG1}, \text{Act}_{LOG1}, \rightarrow_{LOG1}, I_{LOG1}, \text{AP}_{LOG1}, L_{LOG1})$, where

- The state space $S_{LOG1} = \{0, 1\}$ corresponds to the x variable.

- The actions $\mathrm{Act}_{LOG1} = \{0, 1\}$ correspond to the inputs u.
- $a \xrightarrow{b}_{LOG1} c$ iff $c = a \oplus b$.
- $I_{LOG1} = \{0\} \subseteq S_{LOG1}$ corresponds to the fact that $x[0] = 0$.
- $\mathrm{AP}_{LOG1} = \{y\}$ and $2^{\mathrm{AP}_{LOG1}} = \{\emptyset, \{y\}\}$ corresponds to the possible values of the output.
- $L_{LOG1}(0) = \emptyset$ which is interpreted as $y = 0$ and $L_{LOG1}(1) = \{y\}$ which is interpreted as $y = 1$.

See Figure 2 for a pictorial representation.

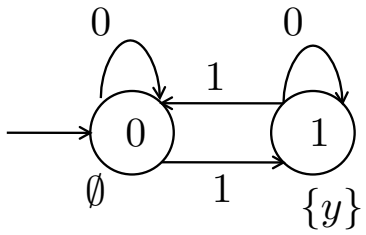

Figure 2: Pictorial representation of TS_{LOG1}, the transitsition system associated with the XOR dynamical system.

As another example of a logical dynamical system, consider the following system with state variables x_1, x_2, input u, and output y:

$$\begin{aligned} x_1[k+1] &= x_2[k] \vee u[k] & x_1[0] = 0 \\ x_2[k+1] &= x_1[k] \wedge u[k] & x_2[0] = 1 \\ y[k] &= x_1[k] \oplus x_2[k], \end{aligned}$$

where $\vee$ is the logical OR operator and $\wedge$ is the logical AND operator. We represent this system as a transition system $TS_{LOG2} = (S_{LOG2}, \mathrm{Act}_{LOG2}, \rightarrow_{LOG2}, I_{LOG2}, \mathrm{AP}_{LOG2}, L_{LOG2})$, where

- $S_{LOG2} = \{0, 1\}^2$ and corresponds to the possible values of (x_1, x_2).
- $\mathrm{Act}_{LOG2} = \{0, 1\}$ and corresponds to the inputs.
- $\rightarrow_{LOG2}$ is the transition relation specified by the dynamics above, $(x_1, x_2) \xrightarrow{u}_{LOG2} (x_1', x_2')$ iff $x_1' = x_2 \vee u$ and $x_2' = x_1 \wedge u$. See Figure 3.
- $I_{LOG2} = \{(0, 1)\}$ because of the initial condition $x_1[0] = 0$, $x_2[0] = 1$.
- $\mathrm{AP}_{LOG2} = \{y\}$ as in the logical system above.

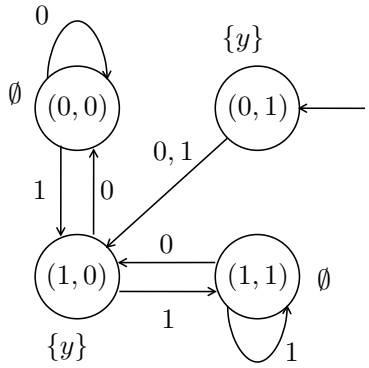

Figure 3: Transition system, TS_{LOG2}. Notice that since $(0,1) \xrightarrow{0}_{LOG2} (1,0)$ and $(0,1) \xrightarrow{1}_{LOG2} (1,0)$, only one edge labeled with both 0 and 1 is drawn from vertex $(0,1)$ to vertex $(1,0)$.

- $L_{LOG2}(x_1, x_2) = \emptyset$ if $x_1 \oplus x_2 = 0$, to indicate output 0. Otherwise $L_{L2}(x_1, x_2) = \{y\}$, to indicate output 1.

See the homework for another example of a logical dynamical system.

2.1 Paths and Traces

For the following definitions, fix a transition system $TS = (S, \text{Act}, \to, I, \text{AP}, L)$.

Definition 2. For $s \in S$ and $\alpha \in \text{Act}$, the set of *α-successors* of s is defined as

$$Post(s, \alpha) = \{s' \in S : s \xrightarrow{\alpha} s'\},$$

and the set of *successors* of s is defined as

$$Post(s) = \bigcup_{\alpha \in \text{Act}} Post(s, \alpha).$$

Similarly, define the sets of *α-predecessors* and *predecessors* of s as

$$Pre(s, \alpha) = \{s' \in S : s' \xrightarrow{\alpha} s\}, \qquad Pre(s) = \bigcup_{\alpha \in \text{Act}} Pre(s, \alpha).$$

Definition 3. A state $s \in S$ is called *terminal* or *blocking* if and only if $Post(s) = \emptyset$.

Definition 4. Let π be a sequence of states, so either $\pi = s_0 s_1 s_2 \dots s_n$ (if the sequence is finite) or $\pi = s_0 s_1 s_2 \dots$ (if the sequence if infinite). The sequence, π, is called:

- a *path fragment* if $s_{i+1} \in Post(s_i)$ for all $i \geq 0$.

- an *initial* path fragment if π is a path fragment and $s_0 \in I$.
- a *maximal* path fragment if π is a path fragment and either $\pi = s_0 s_1 s_2 \dots s_n$ with $Post(s_n) = \emptyset$ or π is infinite.
- a *path* if π is an initial, maximal path fragment.

Let $Paths(TS)$ denote the set of all paths in TS.

Definition 5. Let TS be a transition system with no terminal states (so all paths are infinite). If $\pi = s_0 s_1 s_2 \dots$ is an infinite path fragment, its trace is defined by

$$trace(\pi) = L(s_0)L(s_1)L(s_2)\dots.$$

The set of traces of TS, denoted by $Traces(TS)$, is defined to be the set of traces of all the paths in TS:

$$Traces(TS) = \{trace(\pi) : \pi \in Paths(TS)\}.$$

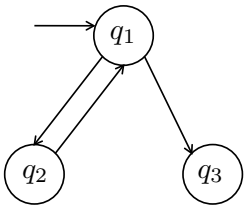

Figure 4: A transition system with a terminal state. Note that actions and atomic proposition labels are omitted. These labels are often omitted in the diagrams, if they are not used in the analysis.

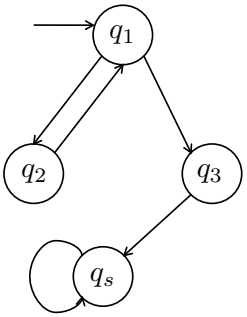

Figure 5: The transition system from Figure 4 modified to have no terminal states.

Example 3. Let TS be the transition system depicted in Figure 4. Note that q_3 is a terminal state. The paths are given by

$$Paths(TS) = \{(q_1 q_2)^\omega\} \cup \{(q_1 q_2)^n q_1 q_3 : n \geq 0\}.$$

(For a string, x, define x^n to be the string $xx \dots x$ (with n repetitions), and define x^ω to be the infinite string, $xx \dots$.)

Usually, we deal with transition systems that have no terminal states, because the properties that we wish to verify will be defined by sets of infinite strings (see Section 3). In order to avoid problems imposed by the terminal state, a new transition system without terminal states can be defined, based on TS, by adding an extra state q_s, such that $q_s \in Post(q_3)$ and q_s has a self-loop ($Post(q_s) = \{q_s\}$). See Figure 5.

Example 4. Recall the traffic light system, TS_{L1}, from Example 1. In this case, $Paths(TS_{L1}) = \{(q_1 q_2)^\omega\}$ and $Traces(TS_{L1}) = \{(\emptyset\{g_1\})^\omega\}$.

2.2 Parallel Composition by Handshaking

Many times complex transition systems are built up from simpler ones by a method known as parallel composition. The composition method described below is often referred to as "handshaking."

Definition 6. Let $TS_1 = (S_1, \mathrm{Act}_1, \rightarrow_1, I_1, \mathrm{AP}_1, L_1)$ and $TS_2 = (S_2, \mathrm{Act}_2, \rightarrow_2, I_2, \mathrm{AP}_2, L_2)$ be transition systems. Their parallel composition, $TS_1 \parallel TS_2$, is the transition system defined by

$$TS_1 \parallel TS_2 = (S_1 \times S_2, \mathrm{Act}_1 \cup \mathrm{Act}_2, \rightarrow, I_1 \times I_2, \mathrm{AP}_1 \cup \mathrm{AP}_2, L)$$

where $L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$ and $\rightarrow$ is defined by the following rules:

- If $\alpha \in \mathrm{Act}_1 \cap \mathrm{Act}_2$, $s_1 \xrightarrow{\alpha}_1 s_1'$, and $s_2 \xrightarrow{\alpha}_2 s_2'$, then $\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2' \rangle$.
- If $\alpha \in \mathrm{Act}_1 \setminus \mathrm{Act}_2$ and $s_1 \xrightarrow{\alpha}_1 s_1'$, then $\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle$.
- If $\alpha \in \mathrm{Act}_2 \setminus \mathrm{Act}_1$ and $s_2 \xrightarrow{\alpha}_2 s_2'$, then $\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle$.

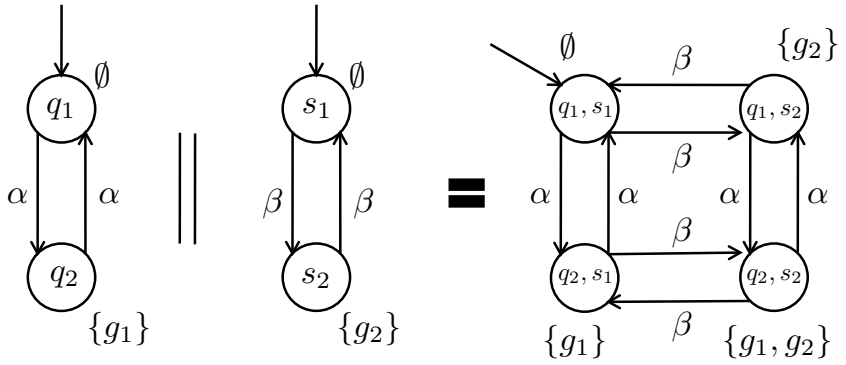

Figure 6: The parallel composition of two traffic lights TS_{L1} and TS_{L2}. Note how the two systems have disjoint action sets, and thus each transition in the composition, $TS_{L1} \parallel TS_{L2}$, corresponds to a transition in either light 1 or light 2.

Example 5. Returning to the traffic light example, consider the original traffic light system TS_{L1} composed with another traffic light system TS_{L2} as in Figure 6.

Note that in Example 5 above, the new system $TS_{L1} \parallel TS_{L2}$ can have some undesirable behaviors. For instance, there are traces of $TS_{L1} \parallel TS_{L2}$ such that g_1 and g_2 hold simultaneously. This would correspond to both lights being green at the same time. Parallel composition is often used to construct controllers that eliminate undesired behaviors from transition system.

In discussing following discussion, the definition of reachability will be useful.

Definition 7. Let $TS = (S, \text{Act}, \rightarrow, I, \text{AP}, L)$ be a transition system. A state s' is said to be *reachable from* s if there is a path fragment of TS, $s_0 s_1 \ldots s_n$ such that $s_0 = s$ and $s_n = s'$.

A state $s \in S$ is called *reachable* if it is reachable from some state $s_0 \in I$. A state is called *unreachable* if it is not reachable.

Denote the set of all reachable states by $Reach(TS)$.

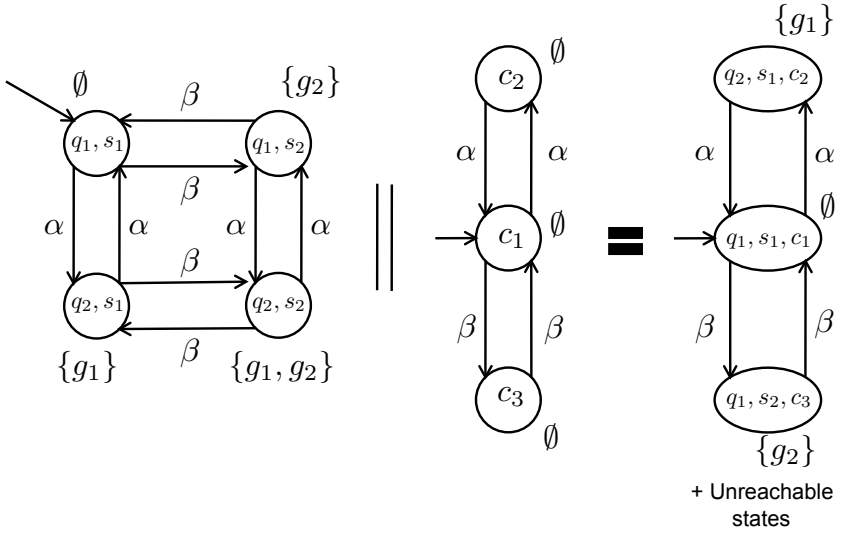

Figure 7: Parallel composition of $TS_{L1} \parallel TS_{L2}$ with the controller C_1. Note that the unreachable states in $T_{L1} \parallel T_{L2} \parallel C_1$ are not drawn.

Example 6. To rule out the possibility that g_1 and g_2 occur simultaneously, we compose $TS_{L1} \parallel TS_{L2}$ with another transition system, C_1, (called a controller) that eliminates the possibility that both occur. See Figure 7. As in the figure, the unreachable states resulting from parallel composition are typically ignored.

Note that some undesired behavior can still occur in $TS_{L1} \parallel TS_{L2} \parallel C_1$ (parallel composition is associative (see the homework). For example $(\emptyset\{g_2\})^\omega \in Traces(TS_{L1} \parallel TS_{L2} \parallel C_1)$, which shows that it is possible that the first light may never turn green. The controller, C_2, depicted in Figure 8, avoids these problems.

3 Linear Time Properties

Definition 8. A linear-time property P over atomic propositions AP is a set of infinite strings over 2^{AP}. In other words, $P \subseteq (2^{\text{AP}})^\omega$.

Definition 9. Let $TS = (S, \text{Act}, \rightarrow, I, \text{AP}, L)$ be a transition system, and let $P \subseteq (2^{\text{AP}})^\omega$ be a linear-time property over AP. We say that TS satisfies P, denoted by $TS \vDash P$, if and only if $Traces(TS) \subseteq P$. If $Traces(TS) \nsubseteq P$, we write $TS \nvDash P$.

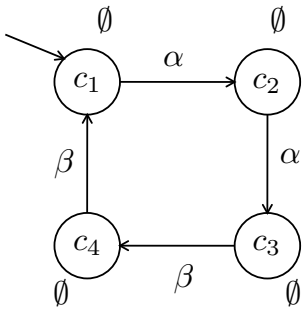

Figure 8: C_2, a controller that guarantees that g_1 and g_2 occur infinitely often, but never simultaneously.

Example 7. Let $\mathrm{AP} = \{g_1, g_2\}$, and consider the following linear-time properties:

$$\begin{aligned} P_1 &= \{A_0A_1A_2\ldots \in (2^{\mathrm{AP}})^\omega : \forall j \quad A_j \neq \{g_1, g_2\}\} \\ P_2 &= \{A_0A_1A_2\ldots \in (2^{\mathrm{AP}})^\omega : g_1 \in A_j \text{ for infinitely many } j\}. \end{aligned}$$

Now recall the controlled traffic lights from Example 6. In that case, $TS_{L1} \parallel TS_{L2} \parallel C_1 \vDash P_1$ but $TS_{L1} \parallel TS_{L2} \parallel C_1 \nvDash P_2$. On the other hand $TS_{L1} \parallel TS_{L2} \parallel C_2 \vDash P_1 \cap P_2$, since $Traces(TS_{L1} \parallel TS_{L2} \parallel C_2) = \{(\emptyset\{g_1\}\emptyset\{g_2\})^\omega\}$.

3.1 Propositional Logic

In order to discuss interesting classes of linear-time properties, we review some concepts from propositional logic.

Let AP be a set of atomic propositions, the set of *propositional logic formulae* is inductively constructed from the following rules:

- True is a formula.
- If $a \in \mathrm{AP}$, then a is a formula.
- If Φ is a formula, then so is $\neg\Phi$.
- If Φ_1 and Φ_2 are formulae, then so is $\Phi_1 \wedge \Phi_2$.
- Nothing else is a formula.

Other familiar logical operators can be defined from $\wedge$ and $\neg$:

$$\begin{aligned} \Phi_1 \vee \Phi_2 &:= \neg(\neg\Phi_1 \wedge \neg\Phi_2) \\ \Phi_1 \implies \Phi_2 &:= \neg\Phi_1 \vee \Phi_2 \\ \Phi_1 \iff \Phi_2 &:= (\Phi_1 \implies \Phi_2) \wedge (\Phi_2 \implies \Phi_1) \\ \Phi_1 \oplus \Phi_2 &:= (\neg\Phi_1 \wedge \Phi_2) \vee (\Phi_1 \wedge \neg\Phi_2). \end{aligned}$$

An *assignment* or *evaluation* for AP is a function $\mu : \text{AP} \to \{0, 1\}$. We say that μ satisfies Φ, (written $\mu \vDash \Phi$, if Φ evaluates to 1 when the values specified by μ are substituted into Φ. Otherwise, we write $\mu \nvDash \Phi$.

For example, if $AP = \{a, b\}$, and $\mu(a) = 0$ and $\mu(b) = 1$, then $\mu \vDash a \oplus b$, but $\mu \nvDash a \wedge b$.

Formally, satisfaction is defined inductively as follows:

- $\mu \vDash \text{True}$.
- If $a \in \text{AP}$, $\mu \vDash a$ iff $\mu(a) = 1$.
- $\mu \vDash \neg\Phi$ iff $\mu \nvDash \Phi$.
- $\mu \vDash \Phi_1 \wedge \Phi_2$ iff $\mu \vDash \Phi_1$ and $\mu \vDash \Phi_2$.

Note that the evaluations for AP are in one-to-one correspondence with the subsets of AP. Indeed, if $\mu : \text{AP} \to \{0, 1\}$, let $A_\mu = \{a \in \text{AP} : \mu(a) = 1\}$, and if $A \subseteq \text{AP}$, let $\mu_A : \text{AP} \to \{0, 1\}$ be such that $\mu_A(a) = 1$ iff $a \in A$. We say that A satisfies Φ ($A \vDash \Phi$) whenever $\mu_A \vDash \Phi$.

Example 8. Let $\text{AP} = \{a, b\}$. Then $\{a\}$, $\{b\}$, and $\{a, b\}$ satisfy $a \vee b$, but $\emptyset \nvDash a \vee b$.

Let $\text{AP} = \{a, b, c, d\}$. Then $\emptyset$ and $\{a, b, c\}$ satisfy $(a \wedge b) \implies (c \vee d)$, but $\{a, b\}$ does not.

3.2 Invariants

Now we come to the first interesting class of linear-time properties, the invariants. Roughly speaking, an invariant encodes the property that some propositional formula must hold at all times.

Definition 10. A linear-time property P_{inv} over AP is an *invariant* if there is a propositional logic formula Φ over AP such that

$$P_{inv} = \{A_0 A_1 A_2 \ldots \in (2^{\text{AP}})^\omega : \forall j \geq 0, \quad A_j \vDash \Phi\}.$$

The formula Φ is called an invariant condition of P_{inv}.

Note that if $TS = (S, \text{Act}, \to, I, \text{AP}, L)$ is a transition system and P_{inv} is an invariant with propositional logic formula Φ, then $TS \vDash P_{inv}$ if and only if $L(s) \vDash \Phi$ for all reachable $s \in S$. Thus if the set of reachable states, $Reach(TS)$, can be computed efficiently (using an algorithm that scales polynomially in $|S|$ and $|\to|$), then invariant properties can be checked efficiently, as long as Φ can be evaluated efficiently. The reachable set of a directed graph is indeed efficiently computable using depth-first search, (see [2] for more on graph search algorithms).

More generally, all the modl checking algorithms we will see in this course exploit the simple fact that depth-first search provides an efficient algorithm to decide if there is a path fragment from one state to another.

Example 9. The property $P_1 = \{A_0A_1A_2\ldots \in (2^{\{g_1,g_2\}})^\omega : \forall j \geq 0, \quad A_j \neq \{g_1, g_2\}\}$ (from Example 7) is an invariant with formula $\neg(g_1 \wedge g_2)$.

3.3 Safety Properties

Safety properties capture the intuitive notion that "nothing bad happens."

First we give some notation.

Definition 11. If Σ is a set, then Σ^* is the set of finite strings of Σ.

If $\Sigma = \{a, b\}$, then $a \in \Sigma^*$ and $abbaababa \in \Sigma^*$. Note that for any set Σ, the empty string, ϵ, is an element of Σ.

Definition 12. If $\sigma \in (2^{\mathrm{AP}})^\omega$ is an infinite string, then its set of prefixes is a set of finite strings defined by

$$pref(\sigma) = \{\hat{\sigma} \in (2^{\mathrm{AP}})^* : \sigma = \hat{\sigma}w \text{ for some } w \in (2^{\mathrm{AP}})^\omega\}.$$

The prefixes of a linear-time property P are given by

$$pref(P) = \bigcup_{\sigma \in P} pref(\sigma).$$

Note that the empty string ϵ is a prefix of every infinite string (i.e. $\epsilon \in pref(\sigma)$ for any σ).

Definition 13. A linear-time property $P_{safe} \subseteq (2^{\mathrm{AP}})^\omega$ is called a *safety* property if there exists a set of finite strings $\mathcal{L} \subseteq (2^{\mathrm{AP}})^*$ such that

$$(2^{\mathrm{AP}})^\omega \setminus P_{safe} = \mathcal{L}.(2^{\mathrm{AP}})^\omega.$$

Here $\mathcal{L}.(2^{\mathrm{AP}})^\omega = \{\hat{\sigma}w \in (2^{\mathrm{AP}})^\omega : \hat{\sigma} \in \mathcal{L} \text{ and } w \in (2^{\mathrm{AP}})^\omega\}$. The largest such $\mathcal{L}$ is denoted by $BadPref(P_{safe})$.

For an infinite string σ we can show that $\sigma \notin P_{safe}$ by finding a finite prefix, $\hat{\sigma} \in pref(\sigma)$, such that $\hat{\sigma} \in BadPref(P_{safe})$.

Note that any invariant is also a safety property. Indeed if Φ is an invariant condition of P_{inv} then

$$BadPref(P_{inv}) = \{A_0A_1\ldots A_n \in (2^{\mathrm{AP}})^* : A_j \not\models \Phi \text{ for some } j\}.$$

Example 10. Our good friend, Dr. G has a special problem. He thinks that eating is a very dirty enterprise, and thus he must shower after every meal. This personal constraint can be modeled as the following safety property over $\text{AP} = \{eat, shower\}$:

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

with bad prefixes

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

Example 11. Given a vending machine that accepts coins and dispenses drinks, a natural requirement is that the machine has always received at least as many coins as the number of drinks it has dispensed. This can be formalized as a safety property over $\text{AP} = \{coin, drink\}$:

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

with bad prefixes

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

3.4 Liveness Properties

Liveness properties capture the intuitive notion that "eventually something happens."

Definition 14. A linear-time property $P_{live} \subseteq (2^{\text{AP}})^\omega$ is called a *liveness* property if $pref(P_{live}) = (2^{\text{AP}})^*$.

Example 12. The property stating that g_1 occurs infinitely often (P_2 from Example 7) is a liveness property, since there are no conditions stating when g_1 must hold.

Example 13. The property that informally states "I will eventually sleep" can be formalized as a liveness property over $\text{AP} = \{sleep\}$:

$$P_{live} = \{A_0A_1A_2 \ldots \in (2^{\text{AP}})^\omega : sleep \in A_j \text{ for some } j \geq 0\}.$$

Interestingly enough, every linear-time property can be formed by the intersection of a safety property and a liveness property.

Theorem 1. *If $P \subseteq (2^{\text{AP}})^\omega$ is a linear-time property, then there is a safety property P_{safe} and a liveness property P_{live} (both over* AP*), such that*

$$P = P_{safe} \cap P_{live}$$

The proof is outlined in the homework.

References

[1] Christel Baier and Joost-Pieter Katoen. Principles of Model Checking (Representation and Mind Series). The MIT Press, 2008.

[2] T. H. Cormen C. E. Leiserson R. L. Rivest and C. Stein. Introduction to Algorithms. MIT Press, second edition edition, 2001.