Lecture Notes
Properties of Transition Systems: Linear-Time, Regular, ω-Regular

Assaf Kfoury

January 12, 2018 (last modified: January 24, 2018)

This handout is a guide through the main concepts in the first four chapters of the textbook *Principles of Model Checking* [PMC] by C. Baier and J.-P. Katoen. I leave out most of the motivational examples, though they are very important, and I mention just a few of the most crucial results and always without their proofs (this is only a guide!). My focus here is on [PMC, Chapters 2, 3, 4] and my presentation mostly follow, but not entirely, the order of the presentation in [PMC]. In later handouts I will discuss material from Chapters 5-6 and later chapters in [PMC].

Although there are notational variations elsewhere in the literature for the material in this handout, much of it is fairly standard and can be found by searching the Web, *e.g.*, start by consulting the Wikipedia article (click here), where they make a distinction between *labelled transition systems* and *unlabelled transition systems* (and both of which are minor variants of *transition systems* as defined below). When appropriate, I compare the notation in [PMC] with that in the textbook *Logic in Computer Science* [LCS] by M. Huth and M. Ryan.

My presentation presumes you have the standard background from undergraduate courses in computer science. You can refresh your memory by consulting the appendices in [PMC] on *formal languages* [PMC, Appendix A.2], *propositional logic* [PMC, Appendix A.3], *graphs* [PMC, Appendix A.4], and *computational complexity* [PMC, Appendix A.5].

1 Transition Systems

Using the conventions of [PMC, Definition 2.1, page 20], a *transition system* TS is a 6-tuple:

$$\mathsf{TS} \triangleq \big(S, Act, \to, I, AP, L\big)$$

where:

- S is a set, finite or countably infinite, of *states*,
- Act is a set, finite or countably infinite, of *actions*,
- $\to \subseteq S \times Act \times S$ is a ternary relation, called the *transition relation*,
- $I \subseteq S$ is a finite set of *initial states*,
- AP is a set, finite or countably infinite, of *atomic propositions*,
- $L : S \to 2^{AP}$ is a *labeling function.*

TS is called *finite* if the three sets S, Act, and AP, are all finite.

As usual, we write 2^{AP} to denote the *power set* of AP, *i.e.*, $2^{AP} \triangleq \{ A \mid A \subseteq AP \}$. Instead of writing formally $(s, \alpha, s') \in \to$, we write $s \xrightarrow{\alpha} s'$ to reflect our informal understanding of how the transition relation works.

Remark 1. The definition of *transition system* in [PMC] is more general than the definition in many other places in the literature. It all depends on how much system analysts want to abstract from the details of the real-world systems that they model.

For example, in the book [LCS, Definition 3.4, page 178], a transition system, typically called $\mathcal{M}$, is simplified in the form of a triple:

$$\mathcal{M} \triangleq (S, \to, L)$$

where S and L are as in [PMC], but $\to \subseteq S \times S$ is a binary relation instead of a ternary relation because it omits any mention of action. This definition in [LCS] is closer to the definition of *state graph* in [PMC, Definition 3.3, page 95], reproduced below to introduce Definition 3. $\square$

Different notions can be used to formalize the behavior of a transition system, depending on how much of that behavior needs to be observed in order to analyze a particular property of interest. We mention three and, informally, what they are meant to describe; their precise definitions follow later:

- **executions**, which describe system runs
  (denoted by the Greek letter ρ appropriately decorated),
- **paths**, which abstract/ignore some incidental parameters from system runs
  (denoted by the Greek letter π appropriately decorated),
- **traces**, which describe observable behaviors occurring during system runs
  (denoted by the Greek letter σ appropriately decorated),

and each of these three notions can be further qualified as being a *fragment* or *maximal*.

The next definition is identical to [PMC, Definition 2.6, page 24].

Definition 2 (**Executions, Execution Fragments, Maximal Executions**). A *finite execution fragment* ρ of a transtion system $\mathsf{TS} \triangleq (S, Act, \to, I, AP, L)$ is an alternating sequence of states and actions ending with a state:

$$\rho \;\triangleq\; s_0 \; \alpha_1 \; s_1 \; \alpha_2 \; s_2 \; \alpha_3 \; \cdots \; \alpha_n \; s_n$$

where $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ for every $0 \leqslant i < n$. An *infinite execution* ρ' of TS is an alternating sequence of states and actions of the form:

$$\rho' \;\triangleq\; s_0 \; \alpha_1 \; s_1 \; \alpha_2 \; s_2 \; \alpha_3 \; \cdots$$

where $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ for every $0 \leqslant i$.

A state $s \in S$ is *terminal* if there is no action $\alpha \in Act$ and no state $s' \in S$ such that $s \xrightarrow{\alpha} s'$. An execution ρ is *maximal* if either ρ is infinite or ρ is a finite fragment ending in a terminal state.

Note that if there is no terminal state in TS, then every maximal execution of TS is necessarily an infinite execution. $\square$

Let $\mathsf{TS} \triangleq (S, Act, \rightarrow, I, AP, L)$ be a transition system. If we do not need to keep track of the actions in Act and the atomic propositions in AP encountered during executions of TS, we consider instead the 'state graph' of TS which abstracts further details from the real-world systems that we are interested in modeling and analyzing.

The following is taken from [PMC, Definition 3.3, page 95]. The *state graph* of TS, denoted $G(\mathsf{TS})$, is the directed graph (V, E) where:

- the set of vertices is $V = S$, and
- the set of edges is $E = \{\, (s, s') \mid s \xrightarrow{\alpha} s' \text{ for some } \alpha \in Act \,\}$.

If we write $s \rightarrow s'$, this means there is a directed edge from s to s' in the graph $G(\mathsf{TS})$. Informally, you can think that $G(\mathsf{TS})$ is the representation of TS as a directed graph where anything related to actions in Act or initial states in I has been erased.

The next definition is identical to [PMC, Definition 3.4, page 95].

Definition 3 (**Paths, Path Fragments, Maximal Paths**). Let $\mathsf{TS} \triangleq (S, Act, \rightarrow, I, AP, L)$ be a transition system and $G(\mathsf{TS})$ its state graph. A *finite path fragment* π of $G(\mathsf{TS})$ (or of TS) is a finite sequence of states of the form:

$$\pi \triangleq s_0\ s_1\ s_2\ \cdots\ s_n$$

where $s_i \rightarrow s_{i+1}$ for every $0 \leqslant i < n$. An *infinite path* π' of $G(\mathsf{TS})$ (or of TS) is an infinite sequence of states of the form:

$$\pi' \triangleq s_0\ s_1\ s_2\ \cdots$$

where $s_i \rightarrow s_{i+1}$ for every $0 \leqslant i$. A path π is *maximal* if either π is infinite or π is a finite path fragment that ends in a terminal state.

If there is no terminal state in TS, then every maximal path of TS is necessarily an infinite path. $\square$

In some analyses, what is considered 'observable' are the atomic propositions that may or may not be true in the states visited during execution. The sequence of states themselves may not be 'observable' during execution, nor the sequence of actions inducing the execution, only the atomic propositions are 'observable'. So, instead of focusing on the execution:

$$\rho \triangleq s_0\ \alpha_1\ s_1\ \alpha_2\ s_2\ \alpha_3\ \cdots$$

or on the path:

$$\pi \triangleq s_0\ s_1\ s_2\ \cdots$$

we need to consider what is called the *trace* of the execution ρ or the path π.

The next definition is adapted from [PMC, Definition 3.8, page 98], though it is not exactly the same.

Definition 4 (**Traces, Trace Fragments, Maximal Traces**). Let $\mathsf{TS} \triangleq (S, Act, \rightarrow, I, AP, L)$ be a transition system and $G(\mathsf{TS})$ its state graph. Let

$$\pi \triangleq s_0\ s_1\ s_2\ s_3\ \cdots$$

be an infinite path (resp. a finite path fragment, resp. a maximal path) of TS. The sequence:

$$trace(\pi) \triangleq L(s_0)\ L(s_1)\ L(s_2)\ L(s_3)\ \cdots$$

is called the *trace* of π, which is further qualified as an *infinite trace* (resp. a *finite trace fragment*, resp. a *maximal trace*). Given that $L : S \to 2^{AP}$, traces are therefore finite on infinite words (*i.e.*, sequences) over the alphabet 2^{AP}.

If there is no terminal state in TS, then every maximal trace of TS is necessarily an infinite trace. □

Make sure you understand how traces are defined. For example, if $AP = \{a, b, c\}$, then the power set 2^{AP} contains 8 elementss:

$$\varnothing \quad \{a\} \quad \{b\} \quad \{c\} \quad \{a,b\} \quad \{a,c\} \quad \{b,c\} \quad \{a,b,c\}$$

and a trace will be a finite or infinite sequence whose entries are all drawn from those 8 elements.

We extend the notion of 'trace' to a single state and to a set of states, as follows:[1]

- If $s \in S$, then $Traces(s) \triangleq \{\, trace(\pi) \mid \pi$ is a maximal path that starts in state $s \,\}$.
- $Traces(\mathsf{TS}) \triangleq \bigcup_{s\in I}\ Traces(s)$.

Make sure you understand the definition of $Traces(\mathsf{TS})$: This is the set of all maximal traces observable when the transition system TS is started in one of its initial states. The book also considers the case of *finite* trace fragments (presented somewhat differently in [PMC, page 98]):

- If $s \in S$, then $Traces_{\text{fin}}(s) \triangleq \{\, trace(\pi) \mid \pi$ is a finite path fragment that starts in state $s \,\}$.
- $Traces_{\text{fin}}(\mathsf{TS}) \triangleq \bigcup_{s\in I}\ Traces_{\text{fin}}(s)$.

Keep in mind that, if there is no terminal state in TS, then $Traces(\mathsf{TS})$ is a set of infinite words while $Traces_{\text{fin}}(\mathsf{TS})$ is a set of finite words.

Remark 5. Our presentation is rather dry and abstract. To get a better grasp of the definitions and why they are useful for a rigorous analysis of system behavior, we need to go through examples. Some of these will be presented in lectures, in handouts, and in homework assignments. □

Remark 6. Much of what follows makes sense, or best sense, when we deal with several transition systems (or processes) running in *parallel* and *communicating* with each other. In the presence of several transition systems, whether cooperating or competing, issues of *parallelism*, *concurrency* and *communication* have to be formalized and amenable to a rigorous analysis. Parallelism can be *asynchronous* or *synchronous*. Communication between parallel processes can be by *interleaving* and via *shared variables*, or *handshaking*, or via *channels*.

These notions are discussed in [PMC, Section 2.2, pages 35-75]. I skip this material in this handout, but you should try to read as much as you can of it, and at least a couple of examples (out of the many in this section); on a first reading, you will have some difficulty in understanding many of the notational conventions, but don't despair.

[1] Although the book [PMC] does not do it, we can equivalently define $Traces(s)$ relative to *executions* rather than *paths*, *i.e.*, we can set $Traces(s) \triangleq \{\, trace(\rho) \mid \rho$ is a maximal execution that starts in state $s \,\}$ where we can write "$trace(\rho)$" instead of "$trace(\pi)$", because traces consider only the labels of states and ignore the actions.

One concept I do not skip here is *nondeterminism* because it is fundamental, whose relevance exceeds the scope of topics covered in this course, and because system properties we discuss later (*e.g.*, *fairness* in Section 4) are meaningful only in the presence of *nondeterminism*.[2] $\square$

The next definition is equivalent to [PMC, Definition 2.5, page 24], though written differently and more simply.

Definition 7 (**Deterministic Transition Systems**). Let $\mathsf{TS} \triangleq (S, Act, \to, I, AP, L)$ be a transition system. We say:

1. TS is *Act-deterministic* iff $|I| \leqslant 1$ and for every $s \in S$ and every $\alpha \in Act$ there is at most one $s' \in S$ such that $s \xrightarrow{\alpha} s'$.
2. TS is *AP-deterministic* iff $|I| \leqslant 1$ and for every $s \in S$ and every $A \in 2^{AP}$ there is at most one $s' \in S$ such that $s \xrightarrow{\alpha} s'$ for some $\alpha \in Act$ and $L(s') = A$.

In general, if we just say TS is *deterministic*, we mean TS is *Act-deterministic*; in words, for every combination (s, α) there is at most one state s' to which state s connects under action α. If TS is not deterministic, then TS is *nondeterministic*, naturally enough. Note that *deterministic* is a just special case of *nondeterministic*, so that *nondeterminism* is a common feature of all systems we analyze. $\square$

2 Linear-Time Properties

Linear-time properties of transition systems are named suggestively: **invariance**, **safety**, **liveness**, **fairness**, **persistence** (which are among the most common), and there are several others. Before we define these concepts precisely, we need to introduce some notational conventions.

Suppose X is a set of elements, say, $X = \{\, x_j \mid j \in J \,\}$ indexed with the elements in the set J. If X is finite with n elements, then the index set $J = \{0, 1, \dots, n-1\}$, an initial fragment of the natural numbers. If X is infinite, then the index set $J = \mathbb{N}$, the set of all natural numbers. We write X^ω to denote the set of all countably right-infinite words/sequences with entries drawn from X; that is, every member of X^ω is of the form;

$$x_{j_0}\ x_{j_1}\ x_{j_2}\ x_{j_3}\ \cdots\ x_{j_k}\ \cdots$$

where k ranges over all the natural numbers and $\{j_0, j_1, j_2, \dots, j_k, \dots\} \subseteq J$. Be careful to note that, even when X is a finite set (as it will often be), every word in X^ω is infinite.

A 'linear-time (LT) property' is a requirement on the traces of a transition system, *i.e.*, on the set *Traces*(TS). The next two definitions are taken from [PMC, Definitions 3.10 and 3.11, page 100].

Definition 8 (**LT Properties**). A *linear-time property* (or *LT property*) P over the set AP of atomic propositions is a subset of $(2^{AP})^\omega$.

Informally, we think of P as a set of 'desirable' or 'good' traces, so that traces in the complement $\big((2^{AP})^\omega - P\big)$ are considered 'undesirable', because observation of the latter means that something 'bad' has occurred in the running of the transition system. $\square$

[2] As a qualifier of how computations or processes may execute, *nondeterministic* is on a par with such fundamental notions as *random* and *probabilistic*, which are used in different contexts and mean different things.

Definition 9 (**Satisfaction of LT Properties**)**.** Let $\mathsf{TS} \triangleq (S, Act, \to, I, AP, L)$ be a transition system without terminal states, so that all the words in *Traces*(TS) are infinite. Let P be a LT property as in Definition 8.

- We say that TS *satisfies* P, denoted $\mathsf{TS} \models P$, iff $\mathit{Traces}(\mathsf{TS}) \subseteq P$.
- Let $s \in S$. We say that state s *satisfies* P, denoted $s \models P$, iff $\mathit{Traces}(s) \subseteq P$.

Informally, if TS satisfies P, then every trace observed during one of TS's executions will be a 'good' trace, indicating that TS's good operation (related to safety, or to fairness, or to any other desirable property) has not been compromised. Note, however, that we do not require that every trace in P is exhibited by some execution of TS; the latter requirement would be written as the equality $\mathit{Traces}(\mathsf{TS}) = P$. □

Let σ be a finite word in $(2^{AP})^*$, *i.e.*, a finite sequence of elements in 2^{AP}, and let σ' be one of the infinite words in $(2^{AP})^\omega$. The concatenation of σ and σ', *i.e.*, the sequence $\sigma\,\sigma'$ is again an infinite word in $(2^{AP})^\omega$. We call σ a *prefix* of $\sigma\,\sigma'$ and σ' a *suffix* of $\sigma\,\sigma'$.

The next definition is from [PMC, Definition 3.20, page 107]. It defines a particular kind of LT property, an 'invariance property'. To understand the notation "$A_j \models \Phi$" below, you should review the first three pages in [PMC, Appendix A.3, pages 915-917] on propositional logic.

Definition 10 (**Invariant Properties**)**.** An LT property P over AP is an *invariant* if there is a propositional-logic formula Φ over AP such that

$$P = \Big\{\ A_0 A_1 A_2 \cdots \in (2^{AP})^\omega \ \Big|\ \text{for every } j \geqslant 0 \text{ it holds that } A_j \models \Phi\ \Big\}.$$

The formula Φ is called the *invariant condition* (or *state condition*) of P. □

The next definition is equivalent to [PMC, Definition 3.22, page 112], though it is not written in the same way. Ours is simpler. A 'safety property' is a particular kind of LT property, which is meant to formalize the intuitive idea that '*bad things never happen*'.

Definition 11 (**Safety Properties**)**.** A LT property P over AP is a called a *safety property* iff there is a set Σ of non-empty finite trace fragments, *i.e.*, $\Sigma \subseteq (2^{AP})^+$, called *bad prefixes*, satisfying two conditions:

1. For every bad prefix $\sigma \in \Sigma$ and every $\sigma' \in (2^{AP})^\omega$, it holds that $\sigma\,\sigma' \in \big((2^{AP})^\omega - P\big)$.
2. For every $\sigma'' \in \big((2^{AP})^\omega - P\big)$, there is a bad prefix $\sigma \in \Sigma$ which is a prefix of σ''.

In words, Part 1 says that the extension of every bad prefix $\sigma \in \Sigma$ to an infinite $\sigma\,\sigma'$ is a 'bad' trace. Part 2 says that every 'bad' trace σ'' is the extension of some bad prefix $\sigma \in \Sigma$.

Intuitively, a LP property P is a safety property if every violation of P occurs after a finite execution of the transition system. □

The finite trace fragments in the set Σ in Definition 11 are called *bad prefixes* of P. However, the two conditions 1 and 2 do not uniquely specify the bad prefixes of P, because if $\sigma_1 \in \Sigma$ and $\sigma_2 \in (2^{AP})^+$ then the concatenation $\sigma_1\sigma_2$ is again a bad prefix of P; *i.e.*, extending a bad prefix σ_1 with any non-empty finite word σ_2 produces another bad prefix $\sigma_1\sigma_2$. To recover 'uniqueness' we can choose Σ to be the largest set (or the smallest set) satisfying those two conditions, as in the next definition.

Definition 12 (**Bad Prefixes**)**.** The *largest* set Σ satisfying conditions 1 and 2 in Definition 11 is the set of all *bad prefixes* of P and denoted $\mathsf{BadPref}(P)$.

The *smallest* set Σ satisfying conditions 1 and 2 in Definition 11 is called the set of *minimal bad prefixes* of P and denoted $\mathsf{MinBadPref}(P)$. $\square$

It is easy to show from the definitions that:

- $\mathsf{MinBadPref}(P)$ and $\mathsf{BadPref}(P)$ are uniquely defined, and
- $\mathsf{MinBadPref}(P) \subsetneq \mathsf{BadPref}(P)$.

A pleasant fact about $\mathsf{MinBadPref}(P)$ is the following:

- If $\sigma_1, \sigma_2 \in \mathsf{MinBadPref}(P)$, then neither σ_1 nor σ_2 is a prefix of the other.

For any finite word $\sigma \in (2^{AP})^*$ or infinite word $\sigma \in (2^{AP})^\omega$, we denote the set of prefixes of σ by $\mathsf{pref}(\sigma)$, *i.e.*:

$$\mathsf{pref}(\sigma) \triangleq \left\{ \sigma' \in (2^{AP})^* \;\middle|\; \sigma' \text{ is a finite prefix of } \sigma \right\}$$

The next definition is taken from [PMC, Definition 3.33, page 121]. A 'liveness property' is another particular kind of LT property, which is meant to formalize the intuitive idea that '*good things eventually happen*'.

Definition 13 (**Liveness Properties**)**.** A LT property P over AP is a called a *liveness property* whenever $\mathsf{pref}(P) = (2^{AP})^*$.

In words, a LP property P is a liveness property if every finite word in $(2^{AP})^*$ can be extended to an infinite word in P. $\square$

A liveness property P can be further refined. For example, given two distinct propositional formulas Φ and Ψ over AP, we may require that every finite word in $(2^{AP})^*$ can be extended to an infinite word $\sigma \in P$ such that:[3]

- there is at least one entry $A \in 2^{AP}$ along σ such that $A \models \Phi$, or
- there are infinitely many entries $A \in 2^{AP}$ along σ such that $A \models \Phi$, or
- every entry A along σ such that $A \models \Phi$ precedes an entry B along σ such that $B \models \Psi$.

The first of these refinements may be called an 'eventually' liveness, the second may be called a 'repeated eventually' liveness, and the third may be called a 'starvation-freedom' liveness. These three refinements of liveness properties are discussed in [PMC, Example 3.34, page 121]. The next result is a remarkable connection between *safety* and *liveness*, given in [PMC, Theorem 3.37, page 124]:

Theorem 14 (Decomposition Theorem)**.** *For every LT property P over AP there exist a safety property P_{safe} and a liveness property P_{live}, both over the same AP, such that $P = P_{safe} \cap P_{live}$.*

Connections between 'safety properties' and 'liveness properties' are discussed further in [PMC, Subsection 3.4.2, pp. 122-126], in particular, the important question of when a LT property P can be

[3]Again here, to understand the notation "$A \models \Phi$", read the first three pages in [PMC, Appendix A.3, pages 915-917] on propositional logic.

required to be simultaneously a 'safety property' and a 'liveness property' [PMC, Lemma 3.35, page 133]. More of the connections between safety and liveness are taken up in a later handout $\boxed{\text{here}}$.

The next definition is taken from [PMC, Definition 4.61, page 199]. A 'persistence property' is a special kind of 'liveness property', which asserts that from a certain moment, a state condition Φ holds continuously.

Definition 15 (**Persistence Properties**). A *persistence property* P over AP is an LT property $P \subseteq (2^{AP})^\omega$ if there is a propositional-logic formula Φ such that:

$$P = \Big\{ A_0A_1A_2 \cdots \in (2^{AP})^\omega \;\Big|\; \text{there } i \geqslant 0 \text{ such that for every } j \geqslant i \text{ it holds that } A_j \models \Phi \Big\}.$$

The formula Φ is called the *persistence condition* (or *state condition*) of P. □

From the preceding definitions, we have the following implications (and in each case the reverse implication does not hold):

1. Every *invariant* property (Definition 10) is a *persistence* property (Definition 15).
2. Every *persistence* property (Definition 15) is a *liveness* property (Definition 13).

3 Regular Safety Properties and ω-Regular Properties

The next definition is taken from [PMC, Definition 4.11, page 159]. It is an additional requirement on 'safety properties'.

Definition 16 (**Regular Safety Properties**). A safety property P over AP as given in Definition 11 and Definition 12 is called a *regular safety property* if the set $\mathsf{BadPref}(P)$ of bad prefixes is a regular language over 2^{AP}. □

In [PMC, Lemma 4.12, page 161], it is shown that the definition of *regular safety property* can be equivalently given using the set of *minimal* bad prefixes:

- A safety property P is regular iff the set $\mathsf{MinBadPref}(P)$ is regular.

Not every safety property is a regular safety property. An example of a safety property which is not regular is [PMC, Example 4.15, page 163].

Further discussion of regular safety properties are in [PMC, Section 4.2, pages 159-170], including several examples and efficient verification using automata theory.

The next definition is [PMC, Definition 4.25, page 172].

Definition 17 (**ω-Regular Properties**). A LT property P over AP is called *ω-regular* if P is an ω-regular language over the alphabet 2^{AP}. □

LT properties that are ω-regular are particularly expressive, in that they subsume the expressive power of several other LT properties, as indicated by the following implications (and in each case the reverse implication does not hold):

1. Every *invariant* property over AP (Definition 10) is an ω-regular property over AP.

2. Every *regular safety* property over AP (Definition 11) is an ω-regular property over AP.
3. Many *liveness* properties over AP are ω-regular properties over AP.
4. Looking ahead: Every *LTL-formula* over AP expresses an ω-regular property over AP.

Fact 1 is already noted in [PMC, top of page 173]. Fact 2 and Fact 3 require each an argument and a proof. For Fact 4, see [PMC, Remark 5.43, page 286].

4 Fairness Properties

In the book [PMC], fairness properties are introduced and discussed, along with other LT properties, in the same chapter [PMC, Chapter 3, pages 126-141]. However, there are several subtle aspects about fairness that make it a little harder to understand than other LT properties. I decided to include them here in a separate section which, I suggest, should be read only after you have a reasonable grasp of the preceding sections in this handout. A good statement about why and how we can deal with fairness is in [PMC, pages 128-129]:

> In general, fairness assumptions are needed to prove liveness or other properties stating that the system makes some progress ("something good will eventually happen"). This is of vital importance if the transition system to be checked contains nondeterminism. Fairness is then concerned with resolving nondeterminism in such a way that it is not biased to consistently ignore a possible option.
>
> $[\cdots]$
>
> A fair execution (or trace) is characterized by the fact that certain fairness constraints are fulfilled. Fairness constraints are used to rule out computations that are considered to be unreasonable for the system under consideration. Fairness constraints come in different flavors:
>
> - *Unconditional fairness*: *e.g.*, "Every process gets its turn infinitely often."
> - *Strong fairness*: *e.g.*, "Every process that is enabled infinitely often gets its turn infinitely often."
> - *Weak fairness*: *e.g.*, "Every process that is continuously enabled, from a certain time instant on, gets its turn infinitely often."

Note carefully from the preceding paragraph: Fairness is about defining assumptions/restrictions on the scheduling of actions to enforce good things to happen in the presence of nondeterminism. Without such assumptions/restrictions, a scheduler can be unfair, as it can take advantage of nondeterminism to prevent good things from happening.

Instead of referring to examples in the books [PMC] and [LCS], I depart from my plan for this handout and include a very simple example to illustrate the differences between the three flavors.

Example 18. I use the so-called *Guarded Command Language* (GCL). For more details on GCL, you can consult the Wikipedia article (click here). Below is a very short code written in GCL:

```
test1 := false ; test2 := false ;
do
```

```
    true   → test1 := true ;   ···           # process A
 |  test1  → test2 := true ;   ···           # process B
 |  test2  → test2 := false ;  ···           # process C
od
```

In GCL, the matching pair **do**-**od** enloses a repetitive construct, which is to be executed repeatedly as long as one of the guards is true. The guards here are: '**true**', 'test1', and 'test2'. We assume that the ellipsis '$\cdots$' in the code above do not change the value of the variables test1 and test2. The **do**-**od** command stops executing when none of its guards is **true**; otherwise one of the guards that has value **true** is chosen *nondeterministically* and the corresponding process is executed, after which the repetition is executed again.

An *unfair* scheduler may never execute process B and/or process C. An *unconditionally-fair* scheduler will eventually give every of the three processes a chance to execute without checking its eligibility. We take a process to be *eligible* for a run if its guard becomes **true**.

A *weakly-fair* scheduler will eventually execute process A and process B, but not process C, because only the first two processes have their guards become continuously **true** from a certain time instant on: the guard of process A is **true** and remains so from the moment the **do**-**od** is entered, and the guard of process B is **true** and remains so after process A is executed once.

Finally, a *strongly-fair* scheduler will eventually give a chance to all three processes to run infinitely many times: the guard of process A is always **true**, the guard of process B becomes **true** and remains so after process A is executed once, and the guard of process C alternates between **true** and **false** forever (it becomes **true** after process B is executed, it becomes **false** after process C is executed).

A possible representation (not the only one) by a transition system $\mathsf{TS} = (S, Act, \to, I, AP, L)$ of the GCL code is specified by:

- $S = \{s_0, s_1, s_2, s_3, s_4\}$,
- $Act = \varnothing$,
- $\to \subseteq S \times (Act \cup \{\tau\}) \times S$ is the transition relation depicted in Figure 1 where τ is a special constant outside Act, [4]
- $I = \{s_0\}$,
- $AP = \{\ \texttt{test1}, \texttt{test2}\ \} \cup \{\ \texttt{A, B, C}\}$ which we abbreviate as $\{\ \texttt{t1}, \texttt{t2}\ \} \cup \{\ \texttt{A, B, C}\}$,
- $L : S \to 2^{AP}$ such that:

  $$s_0 \mapsto \varnothing, \quad s_1 \mapsto \{\texttt{t1,A}\}, \quad s_2 \mapsto \{\texttt{t1,t2,B}\}, \quad s_3 \mapsto \{\texttt{t1,t2,A}\}, \quad s_4 \mapsto \{\texttt{t1,C}\}.$$

  We write "$L(s_1) = \{\texttt{t1,A}\}$", say, to mean that "when the transition system is in state s_1 it holds that $\texttt{t1} = \textbf{true}$ and process A has been executed once". And we write "$L(s_2) = \{\texttt{t1,t2,B}\}$" to mean that "when the transition system is in state s_2 it holds that $\texttt{t1} = \texttt{t2} = \textbf{true}$ and process B has been executed once". And similarly for the intended meanings of $L(s_0)$, $L(s_3)$, and $L(s_4)$.

A graphical representation of TS is shown in Figure 1. □

[4]Following a common practice, I write τ to denote a special constant, assumed outside the set Act of actions. The constant τ is used to represent so-called *silent steps* in the transition system, modeling events that are not observable to any witness of the system. In this example we assume that the scheduler's actions cannot be observed, hence $Act = \varnothing$.

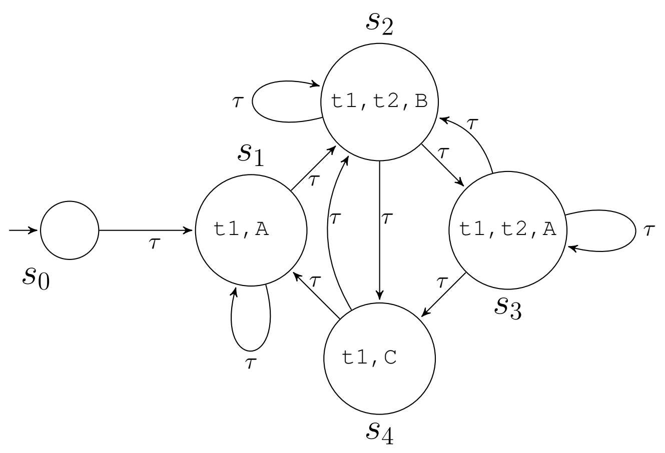

Figure 1: Transition system TS for Example 18. All the transition steps here are *silent*, denoted by τ.

In [PMC, Section 3.5], *action-based* notions of fairness are defined, *i.e.*, they are defined in terms of *executions* and the sequences of actions inducing these executions. Later in [PMC], a *state-based* view of fairness is considered which is defined in terms of *traces*, in [PMC, Subsection 5.1.6, pages 257-270], where connections between action-based and state-based fairness are also studied.

The following is taken from [PMC, Definition 3.43, page 130], written a little differently, using an action-based view of fairness.

Definition 19 (*Fairness: Unconditional, Strong, Weak*)**.** Let $\mathsf{TS} \triangleq (S, Act, \to, I, AP, L)$ be a transition system without terminal states, so that all the executions of TS are infinite. Let $A \subseteq Act$, a subset of actions, and consider an infinite execution ρ of TS:

$$\rho \triangleq \; s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} s_3 \xrightarrow{\alpha_3} \;\cdots$$

We say:

1. ρ is *unconditionally A-fair* iff $\;\overset{\infty}{\exists}\, j.\ (\alpha_j \in A)$,
2. ρ is *strongly A-fair* iff
$$\overset{\infty}{\exists}\, j. \Big(\Big\{ \alpha \in A \;\Big|\; \text{there is } s' \in S \text{ such that } s_j \xrightarrow{\alpha} s' \Big\} \neq \varnothing \Big) \;\text{ implies }\; \overset{\infty}{\exists}\, j.\ (\alpha_j \in A) \ ,$$
3. ρ is *weakly A-fair* iff
$$\overset{\infty}{\forall}\, j. \Big(\Big\{ \alpha \in A \;\Big|\; \text{there is } s' \in S \text{ such that } s_j \xrightarrow{\alpha} s' \Big\} \neq \varnothing \Big) \;\text{ implies }\; \overset{\infty}{\exists}\, j.\ (\alpha_j \in A) \ .$$

The symbols "$\overset{\infty}{\exists}$" and "$\overset{\infty}{\forall}$" here are not formal symbols, *i.e.*, not part of a formal (mathematically defined) logic with quantifiers (*e.g.*, first-order logic). They are shorthands for phrases in plain English:[5]

- "$\overset{\infty}{\exists}$" stands for '*there are infinitely many*'.
  Hence, "$\overset{\infty}{\exists}\, j.(\cdots)$" should be read as "there are infinitely many j such that $(\cdots)$".

[5]If you are familiar with the duality between "$\exists$" and "$\forall$" in first-order logic (*i.e.*, "$\neg\exists\neg$" is equivalent to "$\forall$" and "$\neg\forall\neg$" is equivalent to "$\exists$"), the same duality holds between "$\overset{\infty}{\exists}$" and "$\overset{\infty}{\forall}$" (*i.e.*, "$\neg\overset{\infty}{\exists}\neg$" is equivalent to "$\overset{\infty}{\forall}$" and "$\neg\overset{\infty}{\forall}\neg$" is equivalent to "$\overset{\infty}{\exists}$"). This requires a little proof, which is simple enough by using the duality between "$\vee$" and "$\wedge$".

- "$\overset{\infty}{\forall}$" stands for '*for almost all*' or '*for all except for finitely many*'.
  Hence, "$\overset{\infty}{\forall}\, j.(\cdots)$" should be read as "for almost all j it holds that $(\cdots)$". □

Does the preceding definition settles the issue of satisfying fairness? Not quite. Different levels of fairness are enforced by a spectrum of constraints. From [PMC, page 131]:

> An important question now is: given a verification problem, which fairness notion to use? Unfortunately, there is no clear answer to this question. Different forms of fairness do exist – the above is just a small, though important, fragment of all possible fairness notions – and there is no single favorite notion. For verification purposes, fairness constraints are crucial, though. Recall that the purpose of fairness constraints is to rule out certain "unreasonable" computations. If the fairness constraint is too strong, relevant computations may not be considered. In case a property is satisfied (for a transition system), it might well be the case that some reasonable computation that is not considered (as it is ruled out by the fairness constraint) refutes this property. On the other hand, if the fairness constraint is too weak, we may fail to prove a certain property as some unreasonable computations (that are not ruled out) refute it.

Nonetheless, as a general rule, we have the following relationship, from [PMC, page 132]:

> *unconditional A-fairness* implies *strong A-fairness* and
> *strong A-fairness* implies *weak A-fairness*,
> where the reverse implications do not hold in general.

On a first reading of this handout, you should skip the rest of this section, and also the corresponding parts in [PMC, pages 133-141].

Sometimes it is necessary to impose different restrictions on different, possibly disjoint, sets of actions in order to achieve the desired fairness. Towards this end, [PMC] introduces the notion of a *fairness assumption* which involves different notions of fairness with respect to several sets of actions – in the definition to follow [PMC, Definition 3.46, page 133], I limit the notion to three subsets of actions. It should be read in conjunction with Definition 19.

Definition 20 (**Fairness Assumption**). A *fairness assumption* over Act is a triple $\mathcal{F}$ of subsets of Act, *i.e.*, $\mathcal{F} \triangleq (A_{\text{ucond}}, A_{\text{strong}}, A_{\text{weak}})$ where $A_{\text{ucond}}, A_{\text{strong}}, A_{\text{weak}} \subseteq Act$. We say an infinite execution ρ is *$\mathcal{F}$-fair* iff:

- ρ is unconditionally A_{ucond}-fair, and
- ρ is strongly A_{strong}-fair, and
- ρ is weakly A_{weak}-fair.

If the triple $\mathcal{F}$ is clear from the context, we say "ρ is *fair*" instead of "ρ is *$\mathcal{F}$-fair*".[6] □

Our discussion of fairness so far has been exclusively relative to *executions* (Definition 2). In the book [PMC, top of page 134], it is explained how to lift the definition of fairness to *paths* (Definition 3) and *traces* (Definition 4).

[6] Our definition here is somewhat less general than in [PMC, Definition 3.46, page 133]. In the latter, $\mathcal{F}$ is given as a triple of *sets of subsets*, *i.e.*, $\mathcal{F} = (\mathcal{F}_{\text{ucond}}, \mathcal{F}_{\text{strong}}, \mathcal{F}_{\text{weak}})$ where $\mathcal{F}_{\text{ucond}}, \mathcal{F}_{\text{strong}}, \mathcal{F}_{\text{weak}} \subseteq 2^{Act}$. Then ρ is said to be $\mathcal{F}$-fair iff ρ is unconditionally/strongly/weakly A-fair for every A in $\mathcal{F}_{\text{ucond}}/\mathcal{F}_{\text{strong}}/\mathcal{F}_{\text{weak}}$, respectively. For the examples and exercises we assign, our simpler definition will suffice.

The lifting of fairness to *traces* is relevant for what it means that a LP property is 'fairly satisfied', because the satisfaction of LP properties is relative to *traces* (Definition 9). The next definition is taken from [PMC, Definition 3.48, page 135], somewhat simplified, where we use:

$$\mathit{FairTraces}_{\mathcal{F}}(\mathsf{TS}) \triangleq \{\, \sigma \in \mathit{Traces}(\mathsf{TS}) \mid \sigma \text{ is } \mathcal{F}\text{-fair} \,\}$$

To understand what $\mathit{FairTraces}_{\mathcal{F}}(\mathsf{TS})$ means, you may wish to review Definition 4 and the text following it, down to Remark 5 and including footnote 1.

Definition 21 (**Fair Satisfaction for LT Properties**)**.** Let $\mathsf{TS} \triangleq (S, \mathit{Act}, \rightarrow, I, AP, L)$ be a transition system, $\mathcal{F}$ be a fair assumption over Act, and P be a LT property over AP. We say TS *fairly satisfies* P, in symbols $\mathsf{TS} \models_{\mathcal{F}} P$, iff $\mathit{FairTraces}_{\mathcal{F}}(\mathsf{TS}) \subseteq P$. □

There is more material in [PMC, Section 3.5.3, pages 139-141] in relation to the connections between *fairness* and *safety*, in particular [PMC, Definition 3.54, page 139] and [PMC, Theorem 3.55, page 140], which you can skip on a first reading.