# Runtime - Verification

Harsh Kanani - 112001015

13 November 2022

# Chapter 1

# Monitorability

## 1.1 Introduction

By definition, Monitorability stands for *The condition of being monitorable; the ability to be monitored.* In Runtime Verification We'll talk about the properties' Monitorability or in other words, we'll discuss if a property $\phi$ can be Monitored or not.

In this chapter, We'll first start looking at the properties of Propositional Logic, and then we'll extend our idea to Linear Temporal Logic (LTL). Monitorability is analogous to Decidability.

First let's formally define a Monitor:

- Observed System - $\mathbb{S}$
- Trace $\varepsilon = e_1.e_2.e_3....e_n$
  where $e_i \rightarrow$ observed events, snapshot of system-state.
- Monitor $\mathbb{M} : \mathbb{E}^* \rightarrow D(verdict)$
  $e_1, e_2.. \in \mathbb{E}, \quad d \in D$

## 1.2 Can we monitor all the properties?

$$\mathbb{M}(\varepsilon) \rightarrow verdict$$

- Specification over an infinite trace.
- Observation of a finite prefix.

For any given Trace $\varepsilon$ is it possible to give a verdict only by the observation of its finite prefix? Let's take an example of an atomic property $\phi$. Since $\phi$ is an atomic property at every time step it'll have a value of either true or false. We'll be able to give a verdict on $\phi$ since we only need to look at the current time step and return the value of $\phi$.

This ability to give verdict is called *Monitorability*. We concluded that $\phi$ is Monitorable. Now think about the property $\Box\phi$. Can we give a verdict if $\Box\phi$ is holding on to the infinite Trace $\varepsilon$ just by looking at its finite prefix?

Now there are two cases:

- If the finite prefix is of length n
  - $\phi$ always hold
  - There is at least one-time step where $\phi$ does not hold.
- In the first case, in the finite prefix $\phi$ is always holding. We can neither guarantee if $\phi$ will keep on holding in the future nor if in future $\phi$ will not hold.
- In the second case, we can surely say the Trace $\varepsilon$ is not holding the property $\phi$.

Hence for the Property like $\Box\phi$ there's a possibility for "refutation" but no possibility for "Satisfaction". We can take another example of a property $\Box\Diamond\phi$ where there's no possibility of either "Refutation" or "Satisfaction".

From this, we can conclude that for some properties we can directly give verdicts just by observing their finite trace whereas for some properties we won't be able to give verdicts. So it makes sense to distribute properties into different classes.

## 1.3 Monitorability Classes

Before defining different Monitorability Classes let us first look at the different verdicts which can be given by observing a finite trace. We'll give 3 different verdicts i.e.: (1) Satisfaction (2) Refutation (3) Undecidable.

### 1.3.1 Verdicts

1. **Satisfaction (verdict- "Satisfied"):** All possible extensions of current prefix satisfies $\phi$.
2. **Refutation (verdict- "Failed"):** No extension of current prefix satisfies $\phi$.
3. **Undecidable (verdict- "Undecided"):** Current prefix can be extended to satisfy $\phi$ but can also be extended to refute $\phi$.

**Let's take some examples to understand the concept more clearly.**

1. $\Box\phi$
   - Finite time $\rightarrow$ "Failed" $\rightarrow$ Once I see $(\neg\ \phi)$, all extension fail.
   - Cannot give "Satisfied".
2. $\Diamond\phi$
   - Cannot give "Failed".
   - Give $\rightarrow$ "Satisfied" $\rightarrow$ Once I see $\phi$, then all extension are Satisfied.
3. $\bigcirc\phi$
   - Can give "Satisfied".
   - Can give "Failed".
4. $\Box\Diamond\phi$
   - Always "Undecided"
   - Also known as *Liveness Property*

### 1.3.2 Classes of Properties

- Always Finitely Refutable (AFR):
  - A property $\phi$ is an AFR iff an execution/infinite trace that fails (or does not satisfy $\phi$) can be recognized with a finite prefix.
  - A finite prefix of $\phi$ cannot be extended (in any way) to satisfy $\phi$ (if $\phi$ is not satisfied by the infinite trace).
  - This kind of property is known as *Safety Property.*
  - For Example: $\Box\phi$, $\Box \bigcirc \phi$

- Always Finitely Satisfies (AFS):
  - A property $\phi$ is an AFS iff an execution/infinite trace that satisfies can be recognized with a finite prefix.
  - A finite prefix of $\phi$ cannot be extended (in any way) to refute $\phi$ (if $\phi$ is satisfied by the infinite trace).
  - This kind of property is known as *Co-Safety Property.*
  - For Example: $\Diamond\phi$, $\Diamond \bigcirc \phi$

- Never Finitely Refutes (NFR):
  - A property $\phi$ is an NFR iff an execution/infinite trace that refutes cannot be recognized with a finite prefix.
  - A finite prefix of $\phi$ can be extended (in a way) to satisfy $\phi$ (if $\phi$ is refuted by the inifinite trace).
  - This kind of property is known as *Liveness Property.*
  - For Example: $\Box\Diamond\phi$, $\Diamond\phi$

- Never Finitely Satisfies (NFS):
  - A property $\phi$ is an NFS iff an execution/infinite trace that satisfies cannot be recognized with a finite prefix.
  - A finite prefix of $\phi$ can be extended (in a way) to satisfy $\phi$ (if $\phi$ is refuted by the inifinite trace).
  - This kind of property is known as *Morbidity Property.*
  - For Example: $\Box\Diamond\phi$, $\Box\phi$

- Sometimes Finitely Satisfies (SFS):
  - For some infinite executions that satisfy the property, satisfaction can be identified after a finite prefix; for other infinite executions satisfying the property, this is not the case.
  - In other words, A property $\phi$ is an SFS iff depending on the trace $\phi$ can sometimes behave as NFS and sometimes as AFS.
  - For Example: $(\Box\phi \vee \psi)$
    - infinite trace $\sigma_1$: .....q....

    - infinite trace $\sigma_1$: ppp.......

- Sometimes Finitely Refutable (SFR):
  - For some infinite executions that satisfy the property, refutation can be identified after a finite prefix; for other infinite executions violating the property, this is not the case.
  - In other words, A property $\phi$ is an SFR iff depending on the trace $\phi$ can sometimes behave as NFR and sometimes as AFR.
  - For Example: $(\Box\phi \wedge \psi)$
    - infinite trace $\sigma_1$: $\neg$p....... (Cannot refute)
    - infinite trace $\sigma_2$: ppp....... (Can refute)

## 1.4 Venn-Diagram Of Monitoribility Classes

Let Prop be the set of all characteristics that can be described using a particular temporal formalism, such as Buchi automata or LTL. The equation Prop = AFR $\cup$ SFR $\cup$ NFR is then obvious. The only property that holds for both AFR and NFR and is shared by both of these classes is *True*. Moreover, it is true that Prop = AFS $\cup$ SFS $\cup$ NFS. *False* is the only temporal attribute that applies to both AFS and NFS, which are two of these classes.

After then, each temporal characteristic must belong to the classes XFR and XFS, where X is A, S, or N. The classification used for this is FR/FS. The FR/FS categorization further categorises the characteristics of safety, guarantee, liveness, and morbidity into more detailed groups.

It highlights the areas where these classes cross over.

The nine XFR and XFS combinations that are seen in Figure 1 are illustrated below.

**Examples:**

- SFR $\cap$ NFS: $(\Diamond$ p $\wedge \Box$ q)
- AFR $\cap$ NFS: $\Box$p
- AFR $\cap$ SFS: (p $\vee \Box$q)
- AFR $\cap$ AFS: $\bigcirc$p
- SFR $\cap$ AFS: ( p $\wedge \Diamond$q)
- NFR $\cap$ AFS: $\Diamond$p
- NFR $\cap$ SFS: $(\Box$ p $\vee \Diamond$q)
- NFR $\cap$ NFS: $(\Box\Diamond$ p)
- SFR $\cap$ SFS: ((p $\vee \Box\Diamond$ p) $\wedge\bigcirc$q)

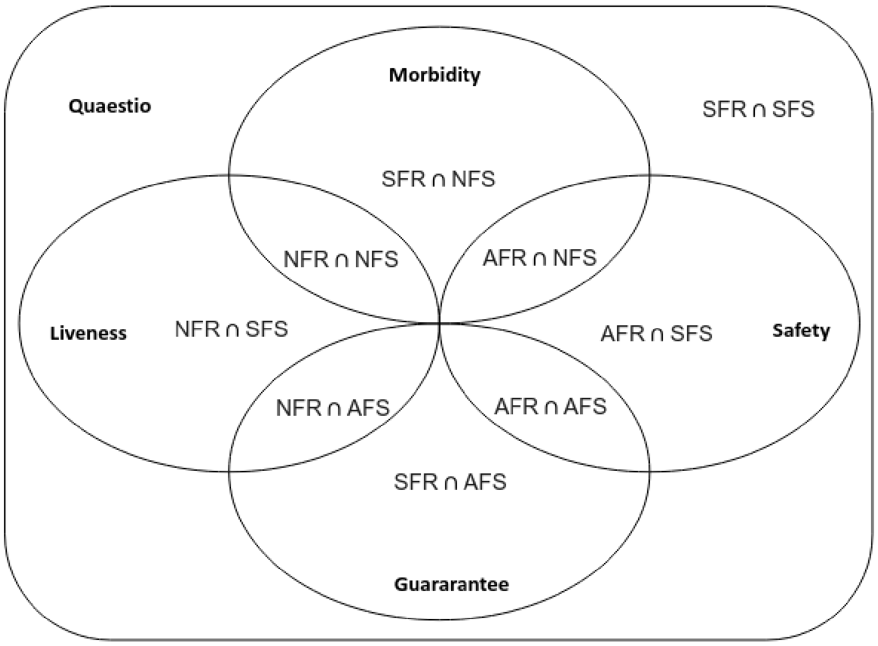


Fig. 1: Classification of properties: safety, guarantee, liveness, morbidity and quaestio.

The entire collection of properties Safety, guarantee, liveness, and morbidity do not apply to props. The properties that are lacking are in SFR ∪ SFS. The class of such features is known as Quaestio (Latin for question).

Be aware that we provided a property with simply the next-time operator as an example for AFR ∩ AFS. We demonstrate that any property $\phi$ in AFR ∩ AFS can be written for LTL using only the next time operator $\bigcirc$ and Boolean operators. Consider a tree whose edges are labeled with elements from $2^P$; every finite path down from the root is labeled with a prefix of a minimal good sequence for $\phi$. This will help you understand it. In other words, if a prefix is valid, the path will end in a leaf node. Given that each node can have a maximum of $2^{|P|}$ successors, this tree has finite branching. Assume this tree has an infinite path. This path must fulfilled since it contains a poor prefix that cannot be extended to satisfy because it is a safety property. So let's presume that this path fulfills. But because it is also guaranteed quality, it needs to start with a finite good. In contrast to the notion that the tree has an infinite path, the construction states that a good prefix leads to a leaf node and is not extended in the tree. One can clearly express in LTL based on the finitely many good pathways in the tree using the $\bigcirc$ operator and the Boolean operators because the tree is finite. The converse is also true. "any property that is expressible in this way corresponds to such a finite tree, and thus is in the intersection of a safety and liveness." [1]

## 1.5 Types Of Prefixes

1. **Good Prefix:** A trace is said to have a good prefix if all extensions of its finite prefix satisfy the property $\phi$.
2. **Bad Prefix:** A trace is said to have a bad prefix if all extensions of its finite prefix refute the property $\phi$.
3. **Ugly Prefix:** A trace is said to have an ugly prefix if some extensions of its finite prefix satisfy the property $\phi$ while some extensions could refute the property $\phi$. One cannot extend an ugly prefix into a good/bad prefix.

**Def: *Monitoribility* is the lack of ugly prefixes fir the property $\phi$.**

- A property has:
  (1) Good prefix if $\phi$ is not in NFS (morbidity).
  (2) Bad prefix if $\phi$ is not in NFR (liveness).
- Some important points
  - A good/bad/ugly prefix continues to be good/bad/ugly even if extended.
  - Stop tracing the execution if we find a good or bad prefix.
  - After an ugly prefix, satisfaction or refutation depends on the entire infinite execution.
  - Every Safety Property is Monitorable and since Guarantee properties are just negation of safety properties one can deduct that Guarantee properties are also Monitorable

## 1.6 Algorithms For Monitoribility

- **Algorithm 1: Monitoring sequences using automata**
  This algorithm was given by Kupferman and Vardi for detecting the good and bad prefix.
  - For Good Prefix, Construct a Buchi Automaton $A_{\neg\phi}$ using the translation in [2]. This automata $A_{\neg\phi}$ may or may not be deterministic. First we'll delete all the states from which a cycle containing an accept state can not be reached. The set of states that $A_{\neg\phi}$ would be after observing that input is kept for each observed prefix when looking for a positive verdict for. One begins with the automaton $A_{\neg\phi}$'s set of initial states. The following set of successors S' is set to the successors of the states in S in accordance with the transition relation $\triangle$ of $A_{\neg\phi}$ given the current set of successors S and an event e $\in 2^P$. i.e. $S' = \{s' | s \in S \wedge (s, e, s') \in \triangle\}$. The monitoring sequence is good when it reaches the empty set of states, and the property must hold because the present prefix cannot be completed into an infinite execution satisfying $\neg\phi$.
  - This is simply a subset construction for a deterministic automaton $B_\phi$, whose accepting state is the empty set, transition relation is as stated above, and initial state is the set of initial states of automaton $A_{\neg\phi}$. Since this automaton has an O($2^{2^{|P|}}$) size, the size of the checked LTL property experiences a double exponential expansion. However, by computing the automaton $B_\phi$'s current state as we go along and running runtime verification, we can escape the double exponential explosion and avoid having to build the automaton $B_\phi$'s whole in advance. The size of the checked LTL attribute is proportional to the incremental processing for each event. Sadly, only one exponential explosion can be prevented [2].

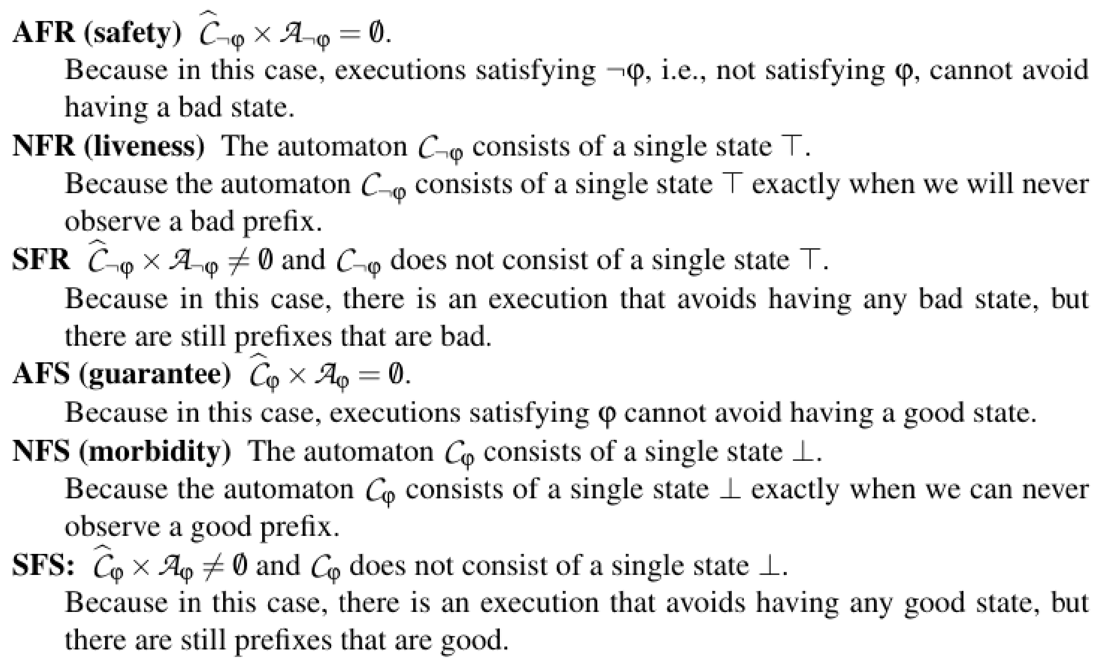
  - Using a symmetric construction and a subset construction as before, it is possible to check for a failed verdict for $\phi$ by first translating it into a Buchi automaton $A_\phi$ and then into the deterministic automaton $B_{\neg\phi}$. Please take note that both $B_{\neg\phi}$ and $B_\phi$ are constructed using $A_\phi$ and $A_{\neg\phi}$ respectively. Both automata are used for the monitored input during runtime verification of $\phi$, which results in a *failed* verdict if $B_{\neg\phi}$ reaches an accepting state, a *satisfied* verdict if $B_\phi$ reaches an accepting state, and an *undecided* verdict in all other cases. The method ensures that it will provide a conclusion of either positive or negative for any detected minimum good or bad prefix.

- **Algorithm 2: Checking availability of future verdicts**
  - In order to determine whether positive or negative results can still be produced after the current monitored prefix at runtime, we modify the previous runtime verification procedure. We look for states from which one cannot reach the accepting state by applying DFS to $B_\phi$. The automaton $C_\phi$ is then created by replacing these states with a single state $\perp$ and a self loop. Reaching $\perp$ occurs precisely when we will no longer have a good prefix $\sigma$, after watching a finite prefix with $C_\phi$. This implies the following, a satisfied verdict cannot be given for $\phi$.
  - In a similar manner, we use BFS on $B_{\neg\phi}$ to identify all the states in which the accepting state is not reachable, then swap out each of those states for a single state $\top$ using a self loop to produce $C_{\neg\phi}$. After monitoring a prefix, reaching $\top$ signifies that we can no longer have a bad prefix, hence a failed verdict cannot be given any longer. If we reach the pair of states $(\perp, \top)$, there is no sense in continuing the monitoring because the runtime verification will not provide any additional information positive or negative in the future. When the prefix being monitored is ugly, this occurs.
  - We are able to runtime verify while dynamically modifying the states of both $C_\phi$ and $C_{\neg\phi}$ automata in response to input events. However, we must be able to foresee when an accepting state will not be possible given the existing situation. The algorithm for that by Pnueli and Zaks can be done in space polynomial in the size of $C_\phi$ and $C_{\neg\phi}$, but it results in an incremental calculation where time complexity is twice exponential in the size of $\phi$. This is scarcely a fair level of complexity for an online algorithm's incremental calculation between subsequent monitored events. Therefore, it is preferable to calculate these two automata in advance before the monitoring begins, leaving the incremental complexity exponential in, as in Algorithm 1.

- **Algorithm 3: Checking monitorability**
  - A little modification to the $C_\phi$ and $C_{\neg\phi}$ design allows one to determine whether a property is monitorable. The process is straightforward: create the product $C_\phi \times C_{\neg\phi}$ and determine whether the state $(\perp, \top)$ is attainable. If this is the case, the property cannot be monitored because it is ugly and has a prefix that will shift the product automaton to this state. Separately verifying that $C_\phi$ can reach $\top$ and that $C\neg\phi$ can reach $\perp$ is insufficient. In the property $(\neg(p \wedge r) \wedge ((\neg p\mathcal{U}(r \wedge q)) \vee (\neg r\mathcal{U}(p \wedge 2q))))$, depending on whether the predicates r or p occur first, both $\perp$ and $\top$ can be achieved. However, in both scenarios, there is still a chance for a good or bad extension, thus it is a monitorable property.
  - There is absolutely no information that we can gather from observing the attribute if the automaton $C\phi \times C\neg\phi$ just has one state $(\perp, \top)$.
  - The above algorithm is reasonably easy to develop, but as the size of the LTL property increases, so does its complexity. Given that the algorithm is run offline and that LTL requirements are sometimes relatively brief, this might not be a problem.

  - $\psi$ has a good prefix $\Leftrightarrow$ $\psi$ is not a morbidity property.
  - For Example: if $\phi = \neg\psi$ is not a liveness property. Now, $\psi$ is not a liveness property $\Leftrightarrow$ either $\phi$ is valid or $\Diamond\neg\phi$ is monitorable.[1]

- **Algorithm 4: Identifying the class of a property**
  - For each given temporal property, we may distinguish the classes of properties AFS (guarantee), SFS, NFS (morbidity), AFR (safety), SFR, and NFS (liveness). As a result, we can also determine whether a property lies at the intersection of two of these classes.
  - We reverse acceptance in $C_\phi$ for the classes AFS, SFS, and NFS, meaning that all states except the empty state are accepting, resulting in $C'_\phi$. We now grab product $C'_\phi \times A_\phi$ and check if they are empty. With the property and state space of $C'_\phi$, we may use a method that conducts model checking[3]. The exact executions that satisfy the criterion and lack a suitable prefix make up the language (approved sequences) of $C'_\phi \times A_\phi$. To determine whether a property $\phi$ is satisfied for such executions, it is never enough to simply note the presence of a finite prefix. We use a similar technique for AFR, SFR, and NFR, taking the product $C'_{\neg\phi} \times A_{\neg\phi}$ and eliminating the accepting state from $C_{\neg\phi}$ to yield $D_{\neg\phi}$.
  - Now we'll follow the conditions listed below for identifying the different classes.



**AFR (safety)** $\widehat{\mathcal{C}}_{\neg\varphi} \times \mathcal{A}_{\neg\varphi} = \emptyset$.
Because in this case, executions satisfying $\neg\varphi$, i.e., not satisfying $\varphi$, cannot avoid having a bad state.

**NFR (liveness)** The automaton $\mathcal{C}_{\neg\varphi}$ consists of a single state $\top$.
Because the automaton $\mathcal{C}_{\neg\varphi}$ consists of a single state $\top$ exactly when we will never observe a bad prefix.

**SFR** $\widehat{\mathcal{C}}_{\neg\varphi} \times \mathcal{A}_{\neg\varphi} \neq \emptyset$ and $\mathcal{C}_{\neg\varphi}$ does not consist of a single state $\top$.
Because in this case, there is an execution that avoids having any bad state, but there are still prefixes that are bad.

**AFS (guarantee)** $\widehat{\mathcal{C}}_{\varphi} \times \mathcal{A}_{\varphi} = \emptyset$.
Because in this case, executions satisfying $\varphi$ cannot avoid having a good state.

**NFS (morbidity)** The automaton $\mathcal{C}_{\varphi}$ consists of a single state $\bot$.
Because the automaton $\mathcal{C}_{\varphi}$ consists of a single state $\bot$ exactly when we can never observe a good prefix.

**SFS:** $\widehat{\mathcal{C}}_{\varphi} \times \mathcal{A}_{\varphi} \neq \emptyset$ and $\mathcal{C}_{\varphi}$ does not consist of a single state $\bot$.
Because in this case, there is an execution that avoids having any good state, but there are still prefixes that are good.

Figure 1.1: Conditions to identify the classes by [1]

## 1.7 Refining Monitoribility

We start by examining the connection between the above categorization of attributes and modulability. Any property that is in AFR (safety) or AFS (guarantee) is monitorable. A property that is NFR $\cap$ NFS cannot be observed. Actually, no decision is ever anticipated on a sequence that is checked against such a property. The three classes SFR $\cap$ SFS, SFR $\cap$ NFS, and NFR $\cap$ SFS are the only ones for which some attributes can be monitored and others cannot. The table below serves as an example of this.

| Class | monitorable example | non-monitorable example |
|---|---|---|
| SFR $\cap$ SFS | $((\Diamond r \vee \Box\Diamond p) \wedge \bigcirc q)$ | $((p \vee \Box\Diamond p) \wedge \bigcirc q)$ |
| SFR $\cap$ NFS | $(\Diamond p \wedge \Box q)$ | $(\Box\Diamond p \wedge \bigcirc q)$ |
| NFR $\cap$ SFS | $(\Box p \vee \Diamond q)$ | $(\Box\Diamond p \vee \bigcirc q)$ |

- If a property can't have and ugly prefix then it is *Monitorable*. Safety and gurantee property are always monitorable. and some examples of SFR $\cap$ SFS, SFR $\cap$ NFS, and NFR $\cap$ SFS are also monitorable.
- If there is no information that can be gathered by monitoring a property for any limited period of time, it has *zero monitoring information.* The properties with no monitoring data are those at the intersection of liveness and morbidity. Applying algorithm 3 (or algorithm 4 for verifying that the property is both in NFR and in NFS) will show whether a property has no monitoring data.
- If there are ugly prefixes, but not all finite prefixes, then the property is *weakly monitorable.* In this situation, there is still information that we can gather by keeping an eye on it, but occasionally we can notice an unsightly prefix from which we cannot draw any worthwhile conclusions in a limited amount of time. To make sure a property is not monitorable but also does not have zero monitoring data, apply algorithm 3. In this situation, one can use Algorithm 2 to perform the runtime verification in addition to utilising Algorithm 1 to accomplish the runtime verification, abandoning the runtime verification if this is not the case. Figure 2 shows the weakly monitorable features as dark grey patches.

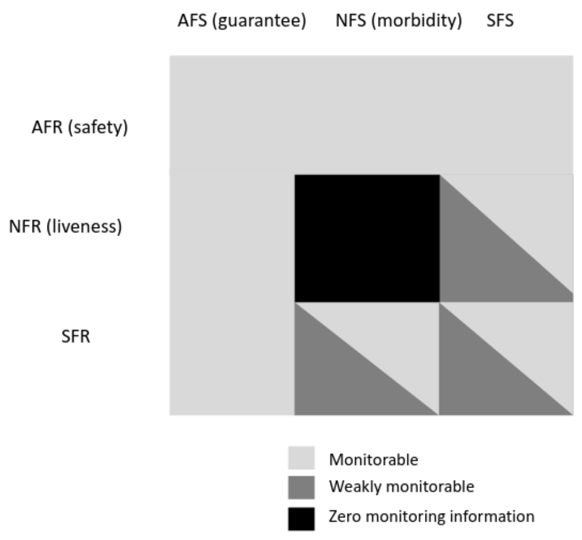


Fig. 2: Classification of properties according to monitorability.

## 1.8 Conclusion

The concept of monitorability describes the types of conclusions that can be drawn by seeing a limited number of execution prefixes. When provided a limited prefix of an execution, monitorability refers to the capacity to arrive at a verdict—positive or negative. Particularly, non-monitorability describes circumstances where it might not be worthwhile to wait for a decision any longer. However, we proposed that the criterion of monitorability needs to be clarified, allowing for the monitoring of qualities where certain useful verdicts may be observed a priori, even if these verdicts are no longer available after witnessing some prefix of the execution.

Additionally, Techniques for determining whether a property is monitorable or not, whether it falls into a specific monitorability class, and what sort of result (positive or negative) we may anticipate after monitoring a specific prefix against a particular attribute. This is helpful in determining if runtime verification should be used for a specific temporal attribute given expected verdicts and what kind of verdicts are still possible after a specific monitored prefix. It also enables us to determine when there is no more useful information that we can anticipate during runtime verification, at which point we can stop monitoring.

## 1.9 References

1 Peled, Doron and Havelund, Klaus. (2019). Refining the Safety–Liveness Classification of Temporal Properties According to Monitorability. $10.1007/978 - 3 - 030 - 22348 - 9_1 4$.

2 O. Kupferman, M. Y. Vardi, Model checking of safety properties. Formal Methods in System Design 19(3): 291-314, 2001

3 E. M. Clarke, O. Grumberg, D. Peled, Model checking, MIT Press, 2000