Progress, Justness and Fairness in Modal μ-Calculus Formulae

Myrthe S. C. Spronck
Eindhoven University of Technology, The Netherlands

Bas Luttik
Eindhoven University of Technology, The Netherlands

Tim A. C. Willemse
Eindhoven University of Technology, The Netherlands

Abstract

When verifying liveness properties on a transition system, it is often necessary to discard spurious violating paths by making assumptions on which paths represent realistic executions. Capturing that some property holds under such an assumption in a logical formula is challenging and error-prone, particularly in the modal μ-calculus. In this paper, we present template formulae in the modal μ-calculus that can be instantiated to a broad range of liveness properties. We consider the following assumptions: progress, justness, weak fairness, strong fairness, and hyperfairness, each with respect to actions. The correctness of these formulae has been proven.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases Modal μ-calculus, Property specification, Completeness criteria, Progress, Justness, Fairness, Liveness properties

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.38

Related Version *Full Version*: `https://arxiv.org/abs/2407.08060`

Acknowledgements We thank an anonymous reviewer for the observation that weak and strong hyperfairness must be distinguished.

1 Introduction

Formal verification through model checking requires a formalisation of the properties of the modelled system as formulae in some logic, such as LTL [32], CTL [17] or the modal μ-calculus [29]. In this paper, we focus on the modal μ-calculus, a highly expressive logic used in established model checkers such as mCLR2 [10] and CADP [19].

A frequently encountered problem when checking liveness properties is that spurious violations are found, such as paths on which some components never make progress. Often, such paths do not represent realistic executions of the system. It is then a challenge to restrict verification to those paths that do represent realistic system executions. For this, we use completeness criteria [21, 22]: predicates on paths that say which paths are to be regarded as realistic runs of the system. These runs are called complete runs. Examples of completeness criteria are progress, justness and fairness.

It turns out that writing a modal μ-calculus formula for a property being satisfied under a completeness criterion is non-trivial. Since the μ-calculus is a branching-time logic, we cannot separately formalise when a path is complete and when it satisfies the property, and then combine the two formalisations with an implication. Instead, a more intricate integration of both aspects of a path is needed. Our aim is to achieve such an integration for a broad spectrum of liveness properties and establish the correctness of the resulting formulae. To this end, we shall consider a template property that can be instantiated to a plethora of liveness properties and, in particular, covers all liveness property patterns of [16]. Then, we present modal μ-calculus formulae integrating the completeness criteria of progress, justness, weak fairness, strong fairness, and hyperfairness with this template property.

As discussed in [23], for the formulation of realistic completeness criteria it is sometimes necessary to give special treatment to a set of blocking actions, i.e., actions that require cooperation of the environment in which the modelled system operates. Our template formulae are therefore parameterised with a set of blocking actions. We shall see that, given a set of blocking actions, there are two different interpretations of hyperfairness; we call these weak and strong hyperfairness.

Regarding our presented formulae, the progress formula is similar to those commonly used for liveness properties even when completeness is not explicitly considered. Our formulae for justness, weak fairness and weak hyperfairness only subtly differ from each other. We characterise the similarities these three share and give a generic formula that can be adapted to represent all completeness criteria that meet these conditions. Lastly, we observe that strong fairness and strong hyperfairness do not meet these conditions. We give alternative formulae that are significantly more complex. Whether more efficient formulae for these completeness criteria exist remains an open problem.

Modal μ-calculus formulae are often hard to interpret. Accordingly, it is not trivial to see that our formulae indeed express the integration of liveness properties with completeness criteria. We therefore include elaborate correctness proofs in the full version of this paper.

Our work is essentially a generalisation along two dimensions (viz., the completeness criterion and the liveness property) of the works of [34] and [6, 36]. In [34], the tool PASS is presented for automatically translating common property patterns into modal μ-calculus formulae. Some of those patterns integrate an assumption that excludes paths deemed unrealistic, but since the exact assumption is not stated separately, we cannot make a formal comparison with our approach. In [6], a formula for justness is presented, covering one of the properties we cover. This formula forms the basis for our justness, weak fairness and weak hyperfairness formulae. Our formulae for strong fairness and strong hyperfairness are in part inspired by the formula for termination under strong fairness presented in [36].

The organisation of this paper is as follows. In Section 2 we recap the relevant definitions on labelled transition systems, as well as the syntax and semantics of the modal μ-calculus. In Section 3, we motive our work with an example, and in Section 4 we give the completeness criteria we cover in this paper. In Section 5, we formally identify the class of liveness properties we study and relate it to a popular class of properties. Our template formulae are presented in Section 6, combining the completeness criteria from Section 4 with the property template from Section 5. We give a small application example in Section 7 and discuss the scope of our work in Section 8. Finally, we give our conclusions in Section 9.

2 Preliminaries

We represent models as labelled transition systems (LTSs). In this section, we briefly introduce the relevant definitions on LTSs, as well as the modal μ-calculus.

2.1 Labelled Transition Systems

▶ **Definition 1.** *An LTS is a tuple $M = (\mathcal{S}, s_{init}, Act, Trans)$ where*

- *$\mathcal{S}$ is a set of states,*
- *$s_{init} \in \mathcal{S}$ is the initial state,*
- *Act is a set of action labels, also referred to as the alphabet of the LTS, and*
- *$Trans \subseteq \mathcal{S} \times Act \times \mathcal{S}$ is a transition relation.*

In this paper, we only consider finite LTSs, such as the kind used in finite-state model checking. In particular, our formulae are proven correct under the assumption that *Act* is finite. We write $s \xrightarrow{a} s'$ as shorthand for $(s, a, s') \in \mathit{Trans}$, and for a given transition $t = (s, a, s')$ we write $\mathit{src}(t) = s$, $\mathit{act}(t) = a$ and $\mathit{trgt}(t) = s'$.

For the definitions below, we fix an LTS $M = (\mathcal{S}, s_{init}, \mathit{Act}, \mathit{Trans})$.

▶ **Definition 2.** *A* path *is an (alternating) sequence $\pi = s_0 t_1 s_1 t_2 \ldots$ of states $s_0, s_1, \ldots \in \mathcal{S}$ and transitions $t_1, t_2, \ldots \in \mathit{Trans}$. A path must start with a state, and must be either infinite, or end in a state. In the latter case, the end of the path is referred to as the* final state*. For all $i \geq 0$, t_{i+1} must satisfy $\mathit{src}(t_{i+1}) = s_i$ and $\mathit{trgt}(t_{i+1}) = s_{i+1}$.*

We sometimes refer to transitions on a path as steps. We say an action occurs on a path if a transition labelled with that action is on the path. We call a path on which no action in some set α occurs an α*-free* path. One path can be appended to another: let $\pi' = s'_0 t'_1 s'_1 \ldots t'_n s'_n$ and $\pi'' = s''_0 t''_1 s''_1 \ldots$, where π' must be finite and π'' may be finite or infinite. Then the path π defined as π'' appended to π' is written as $\pi = \pi' \cdot \pi'' = s'_0 t'_1 s'_1 \ldots t'_n s'_n t''_1 s''_1 \ldots$. This is only allowed when $s'_n = s''_0$.

▶ **Definition 3.** *We say that:*

- *A transition $t \in \mathit{Trans}$ is* enabled *in a state $s \in \mathcal{S}$ if, and only if, $\mathit{src}(t) = s$.*
- *An action $a \in \mathit{Act}$ is* enabled *in a state $s \in \mathcal{S}$ if, and only if, there exists a transition $t \in \mathit{Trans}$ with $\mathit{act}(t) = a$ that is enabled in s.*
- *An action $a \in \mathit{Act}$ is* perpetually enabled *on a path π if a is enabled in every state of π.*
- *An action $a \in \mathit{Act}$ is* relentlessly enabled *on a path π if every suffix of π contains a state in which a is enabled.*
- *A state without enabled actions is called a* deadlock state.

Every action that is perpetually enabled on a path is also relentlessly enabled on that path.

2.2 Modal μ-Calculus

The modal μ-calculus is given in [29]. Our presentation of the logic is based on [7, 8, 9, 26].

The syntax of the modal μ-calculus is described by the following grammar, in which a ranges over the set of actions *Act*, and X ranges over a set of formal variables *Var*.

$$\phi, \psi ::= \mathit{ff} \mid X \mid \neg\phi \mid \phi \vee \psi \mid \langle a \rangle \phi \mid \mu X.\phi$$

Here ff is false; $\neg$ represents negation; $\vee$ is disjunction; $\langle\ \rangle$ is the diamond operator; and μ is the least fixpoint operator. We say that $\mu X.\phi$ binds X in ϕ. Variables that are unbound in a formula are free, and a formula without free variables is closed.

A modal μ-calculus formula ϕ must both adhere to this grammar and be *syntactically monotonic*, meaning that for every occurrence of $\mu X.\psi$ in ϕ, every free occurrence of X in ψ must always be preceded by an even number of negations.

We give the semantics of a modal μ-calculus formula ϕ with respect to an arbitrary LTS $M = (\mathcal{S}, s_{init}, \mathit{Act}, \mathit{Trans})$ and environment $e : \mathit{Var} \to 2^{\mathcal{S}}$.

$$\begin{aligned}
&[\![\mathit{ff}]\!]^M_e = \emptyset && [\![\phi \vee \psi]\!]^M_e = [\![\phi]\!]^M_e \cup [\![\psi]\!]^M_e \\
&[\![X]\!]^M_e = e(X) && [\![\langle a \rangle \phi]\!]^M_e = \left\{ s \in \mathcal{S} \mid \exists_{s' \in \mathcal{S}}.s \xrightarrow{a} s' \wedge s' \in [\![\phi]\!]^M_e \right\} \\
&[\![\neg\phi]\!]^M_e = \mathcal{S} \setminus [\![\phi]\!]^M_e && [\![\mu X.\phi]\!]^M_e = \bigcap \left\{ \mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq [\![\phi]\!]^M_{e[X:=\mathcal{S}']} \right\}
\end{aligned}$$

In contexts where the model is fixed, we drop the M from $[\![\phi]\!]^M_e$. Additionally, we drop e when the environment does not affect the semantics of the formula, e.g. with closed formulae.

We use conjunction, $\wedge$, and implication, $\Rightarrow$, as the usual abbreviations. We also add several abbreviations: $tt = \neg ff$ for true; $[a]\phi = \neg\langle a\rangle\neg\phi$ for the box operator; and $\nu X.\phi = \neg\mu X.(\neg\phi[X := \neg X])$ for the greatest fixpoint.

To express formulae more compactly, we extend our syntax to allow regular expressions over finite sets of actions to be used in the box and diamond operators. Since we limit this to finite sets of actions, the syntactical extension does not increase the expressivity of the logic, it merely simplifies the presentation. This is a common extension of the μ-calculus syntax, for instance shown in [26], based on the operators defined for PDL [18]. We overload the symbol for a single action to also represent the singleton set containing that action. We use union, intersection, set difference, and set complement to describe sets of actions as usual. Regular expressions over sets of actions, henceforth referred to as *regular formulae*, are defined by the following grammar:

$$R, Q ::= \varepsilon \mid \alpha \mid R \cdot Q \mid R + Q \mid R^\star$$

The empty sequence is represented by ε, and α ranges over sets of actions. The symbol $\cdot$ represents concatenation, $+$ the union of formulae, and * is closure under repetition.

We define the meaning of the diamond operator over the new regular formulae as abbreviations of standard modal μ-calculus formulae:

$$\langle\varepsilon\rangle\phi = \phi \qquad \langle\alpha\rangle\phi = \bigvee_{a\in\alpha}\langle a\rangle\phi \qquad \langle R\cdot Q\rangle\phi = \langle R\rangle\langle Q\rangle\phi$$

$$\langle R + Q\rangle\phi = \langle R\rangle\phi \vee \langle Q\rangle\phi \qquad \langle R^\star\rangle\phi = \mu X.(\langle R\rangle X \vee \phi)$$

The box operator is defined dually. We say a path π *matches* a regular formula R if the sequence of actions on π is in the language of R.

3 Motivation

When analysing algorithms and systems, there are many different properties which may need to be checked. For instance, when model checking mutual exclusion algorithms we want to check linear properties such as mutual exclusion and starvation freedom, but also branching properties such as invariant reachability of the critical section. The modal μ-calculus, which subsumes even CTL*, is able to express all these properties and more, and is therefore used in toolsets such as mCLR2 [10] and CADP [19].

An issue that is frequently encountered when checking liveness properties in particular, is that the model admits executions that violate the property but do not represent realistic executions of the real system. For example, models of algorithms that contain a busy waiting loop usually admit executions where processes do nothing except wait. Infinite loops can also be introduced by abstractions of reality, such as modelling a loop to represent an event that occurs an arbitrary, but finite, number of times. Counterexamples that are due to such modelling artefacts obscure whether the property is satisfied on all realistic executions. The problem we address in this paper is how to avoid such counterexamples and check properties only on realistic executions. We illustrate the problem with an example, which we also employ as a running example throughout this paper.

▶ **Example 4.** Consider the coffee machine modelled in Figure 1. When a user places an *order* for one or more cups of coffee, they are required to scan their payment *card*. If the user prefers using coinage, they switch the machine to its alternate mode (*to_cash*), and then pay in *cash*. In the alternate mode, the machine can be switched back using *to_card*. After

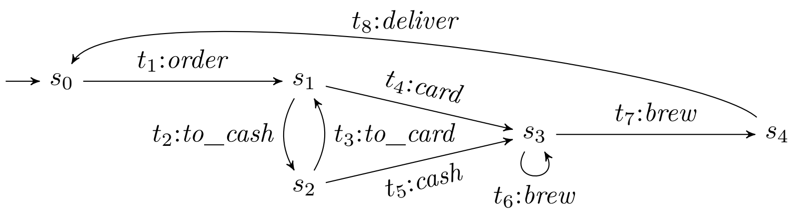

Figure 1 The LTS for the running example.

payment, the machine will *brew* the cup(s) of coffee. This is modelled as a non-deterministic choice between a looping and a final *brew* action, since at least one cup was ordered. Finally, the coffee is *deliver*ed and the machine awaits the next order.

We consider three example properties.

1. *Single order*: whenever an *order* is made, there may not be a second *order* until a *deliver* has taken place, $[Act^\star \cdot order \cdot \overline{deliver}^\star \cdot order]\mathit{ff}$.
2. *Inevitable delivery*: whenever an *order* is made, there will inevitably be an occurrence of *deliver*, $[Act^\star \cdot order]\mu X.(\langle Act\rangle tt \wedge [\overline{deliver}]X)$.
3. *Possible delivery*: it is invariantly possible to eventually execute the *deliver* action, $[Act^\star]\langle Act^\star \cdot deliver\rangle tt$.

The described problem occurs with *inevitable delivery*: $s_0t_1s_1t_4(s_3t_6)^\omega$ is a violating path, on which infinitely many cups are part of the same order. Similarly, $s_0t_1(s_1t_2s_2t_3)^\omega$ violates the property because the user never decides on a payment method. The first counterexample represents an impossible scenario, and the second gives information on problematic user behaviour but tells us little about the machine itself.

The kind of spurious counterexamples discussed in the example above primarily occur when checking liveness properties. We therefore focus on liveness properties, such as *inevitable delivery*, in this paper. We will briefly discuss safety properties in Section 8.

There are ad-hoc solutions to exclude unrealistic counterexamples, e.g. altering the model to remove the unrealistic executions, or tailoring the formula to exclude specific problematic counterexamples [25]. Such ad-hoc solutions are undesirable because they clutter the model or the formula, and are therefore error-prone. We aim for a more generic solution, of which the correctness can be established once and for all. Such a generic solution requires, on the one hand, a general method to distinguish between realistic and unrealistic executions, and, on the other hand, a general class of liveness properties.

A general method to distinguish between realistic and unrealistic executions is provided by *completeness criteria* [21, 22], i.e., predicates on paths that label some as complete and all others as incomplete. If a property is satisfied on all complete paths, it is satisfied under the given completeness criterion. Completeness criteria give us a model-independent way to determine which paths are unrealistic, and therefore a generic solution to the stated problem. Depending on the property and the model, we may prefer a different completeness criterion. We therefore consider several criteria instead of fixing one specific criterion. These completeness criteria are discussed in Section 4.

To find a general class of liveness properties, we take the *property specification patterns* (PSP) of [16] as a starting point. Since the modal μ-calculus as presented in Section 2.2 supports references to action occurrences but not state information, we specifically interpret these patterns on action occurrences. Our first contribution, in Section 5, will be to characterise a class of liveness properties that subsumes all liveness properties expressible in

PSP. Our second and main contribution is then presented in Section 6, where we combine the identified completeness criteria with our class of liveness properties, yielding template formulae for each combination.

4 Completeness Criteria

It is often assumed, sometimes implicitly, that as long as a system is capable of executing actions, it will continue to do so [24]. One could consider this the "default" completeness criterion, also known as *progress* [21]; it says that only paths that are infinite or end in a deadlock state model complete runs and are hence complete paths. We first present a modified version of the progress assumption that allows some actions to be blocked by the environment. We then define the other completeness criteria considered in this paper. As already remarked in the previous section, the modal μ-calculus is most suited to reasoning about action occurrences. Hence, we focus on completeness criteria defined on action labels. For more general definitions on sets of transitions, see [24].

4.1 Progress with Blocking Actions

In [23], it is argued that it is useful to consider some actions of an LTS as blocking. A blocking action is an action that depends on participation by the environment of the modelled system. Consequently, even when such an action is enabled in a state because the system is willing to perform it, it may not be possible for the action to occur because the environment is uncooperative. In this paper, we refer to the set of blocking actions as $\mathcal{B} \subseteq \mathit{Act}$, and the set of non-blocking actions as $\overline{\mathcal{B}} = \mathit{Act} \setminus \mathcal{B}$. Which actions are in $\mathcal{B}$ is a modelling choice.

The default progress assumption can be adapted to account for blocking actions [20, 24].

▶ **Definition 5.** *A state $s \in \mathcal{S}$ is a $\mathcal{B}$-*locked state *if, and only if, all actions enabled in s are in $\mathcal{B}$. A path π is $\mathcal{B}$-*progressing *if, and only if, it is infinite or ends in a $\mathcal{B}$-locked state.*

We refer to the assumption that only $\mathcal{B}$-progressing paths represent complete executions as $\mathcal{B}$-progress. The "default" completeness criterion is equivalent to $\emptyset$-progress.

▶ **Example 6.** Consider Figure 1. Here, *order* is an environment action, since it involves the user. If we do not assume that there will always be a next user, we should add *order* to $\mathcal{B}$. In some cases, we may want to consider the possibility that the machine is broken and not capable of producing coffee. In those cases, we should add *brew* to $\mathcal{B}$. Our choice of $\mathcal{B}$ affects which paths are progressing: $s_0 t_1 s_1 t_4 s_3$ is not $\emptyset$-progressing, but it is $\{\mathit{brew}\}$-progressing.

All completeness criteria we discuss in this paper are parameterised with a set of blocking actions. The justness and fairness assumptions discussed in the remainder of this section label paths as incomplete if certain actions do not occur. Since it can never be assumed that the environment supports the occurrence of blocking actions, we do not want justness and fairness to label paths as incomplete due to the non-occurrence of blocking actions.

For readability the prefix $\mathcal{B}$- will sometimes be dropped from the names of the completeness criteria and their acronyms. From this point, we will always discuss completeness criteria with respect to a set of blocking actions.

4.2 Justness

Justness [20, 24] is a natural extension of progress to exclude infinite paths instead of finite paths. The idea is that in addition to the system as a whole progressing, individual components in that system should also be able to make progress unless they are prevented from doing so by other components. It is a weaker, and hence frequently more justifiable, assumption than the fairness assumptions we cover in the next section. In its original presentation, justness is defined with respect to sets of transitions. Which components contribute to a transition and how they contribute to them determines which transitions interfere with each other. We here consider justness defined with respect to actions instead, based on [6]. We do not go into how it is determined which actions interfere with each other here. For discussions on this topic and when the two definitions coincide, see [5, 6, 20].

Intuitively, justness of actions says that if an action a is enabled at some point of a path, then eventually some action that can interfere with the occurrence of a must occur in that path. That action may be a itself. In order to formalise the concept of interference, we require the concept of a *concurrency relation on actions*, $\mathrel{\smile\!\!\bullet}$.

▶ **Definition 7.** *Relation* $\mathrel{\smile\!\!\bullet} \subseteq \mathit{Act} \times \mathit{Act}$ *is a* concurrency relation on actions *if, and only if:*

1. $\mathrel{\smile\!\!\bullet}$ *is irreflexive, and*
2. *for all* $a \in \mathit{Act}$*, if* π *is a path from a state* s *in which* a *is enabled to a state* $s' \in \mathcal{S}$ *such that* $a \mathrel{\smile\!\!\bullet} b$ *for all actions* b *occurring in* π*, then* a *is enabled in* s'*.*

We write $\mathrel{\not\smile\!\!\bullet}$ for the complement of $\mathrel{\smile\!\!\bullet}$. Note that $\mathrel{\smile\!\!\bullet}$ may be asymmetric.

Read $a \mathrel{\smile\!\!\bullet} b$ as "a is concurrent with b", and $a \mathrel{\not\smile\!\!\bullet} b$ as "b interferes with a" or "b eliminates a". A labelled transition system can be extended with a concurrency relation on actions, which produces a *labelled transition system with concurrency* (LTSC).

We here present the definition for justness of actions with blocking actions.

▶ **Definition 8.** *A path* π *satisfies* $\mathcal{B}$-justness of actions ($\mathcal{B}$-JA) *if, and only if, for each action* $a \in \overline{\mathcal{B}}$ *that is enabled in some state* s *in* π*, an action* $a' \in \mathit{Act}$ *occurs in the suffix* π' *of* π *starting in* s *such that* $a \mathrel{\not\smile\!\!\bullet} a'$*.*

▶ **Example 9.** Consider Figure 1, specifically the path $s_0 t_1 (s_1 t_2 s_2 t_3)^\omega$. On this path the user keeps switching the mode of the machine, without paying. To see if this path satisfies $\emptyset$-JA, we need a concrete $\mathrel{\smile\!\!\bullet}$. Consider a $\mathrel{\smile\!\!\bullet}$ such that $\mathit{card} \mathrel{\not\smile\!\!\bullet} \mathit{to\_cash}$, $\mathit{cash} \mathrel{\not\smile\!\!\bullet} \mathit{to\_card}$, and $a \mathrel{\not\smile\!\!\bullet} a$ for all action labels a. These are all required for $\mathrel{\smile\!\!\bullet}$ to be a valid concurrency relation. This is because by Definition 7, $\mathrel{\smile\!\!\bullet}$ must be irreflexive, and when an action is enabled it must remain enabled on any path on which no interfering action occurs. Since *card* is enabled in s_1 but not s_2, it must be the case that $\mathit{card} \mathrel{\not\smile\!\!\bullet} \mathit{to\_cash}$. Similarly, we must have $\mathit{cash} \mathrel{\not\smile\!\!\bullet} \mathit{to\_card}$. With such a concurrency relation, the path satisfies $\emptyset$-JA since every action that is enabled is subsequently eliminated. In this LTS, there is no valid choice of $\mathrel{\smile\!\!\bullet}$ that makes this path violate $\emptyset$-JA. However, if we modify Figure 1 by replacing both *card* and *cash* with the action *pay*, then Definition 7 does not enforce that *to_cash* and *to_card* interfere with the actions on t_4 and t_5, since *pay* is enabled in both s_1 and s_2. We can choose whether $\mathit{pay} \mathrel{\smile\!\!\bullet} \mathit{to\_cash}$ and $\mathit{pay} \mathrel{\smile\!\!\bullet} \mathit{to\_card}$. If *pay* is concurrent with both, then the path $s_0 t_1 (s_1 t_2 s_2 t_3)^\omega$ violates $\emptyset$-JA. If either interferes with *pay*, then the path satisfies $\emptyset$-JA.

4.3 Fairness

There are situations where we want to exclude a larger set of infinite paths than those excluded by justness, or where we do not have a concurrency relation. For this, we can use what are called *fairness assumptions* in the literature. These are a class of predicates on paths that distinguish between *fair* and *unfair* infinite paths. It is assumed that only the fair paths are complete. For an overview of many common fairness assumptions, see [24]. In this paper, we consider weak fairness of actions, strong fairness of actions, and (weak and strong) hyperfairness of actions. Each of the assumptions we discuss has the general shape, adapted from [2], "if it is sufficiently often possible for an action to occur, it will occur sufficiently often". What it means for an action to be "sufficiently often possible" and "occur sufficiently often" depends on the exact assumption.

We first discuss *weak fairness of actions*, which says that actions that are always enabled must eventually occur. It is one of the most commonly discussed fairness assumptions. We define weak fairness of actions formally, with respect to a set of blocking actions $\mathcal{B}$.

▶ **Definition 10.** *A path π satisfies* $\mathcal{B}$-weak fairness of actions ($\mathcal{B}$-WFA) *if, and only if, for every suffix π' of π, every action $a \in \overline{\mathcal{B}}$ that is perpetually enabled in π' occurs in π'.*

▶ **Example 11.** Consider again Figure 1, with *card* and *cash* both replaced by *pay*. Then the path $s_0t_1(s_1t_2s_2t_3)^\omega$ violates $\emptyset$-WFA, since *pay* is perpetually enabled in a suffix of this path without occurring. If there are two separate actions for paying with cash or card, the path satisfies $\emptyset$-WFA because no actions are perpetually enabled in any suffix.

Next, *strong fairness of actions* says that on a path, all actions that are enabled infinitely often, must occur infinitely often. Formally, we define strong fairness of actions as:

▶ **Definition 12.** *A path π satisfies* $\mathcal{B}$-strong fairness of actions ($\mathcal{B}$-SFA) *if, and only if, for every suffix π' of π, every action $a \in \overline{\mathcal{B}}$ that is relentlessly enabled in π' occurs in π'.*

Strong fairness is a stronger assumption than weak fairness, since it classifies more paths as incomplete. This follows from perpetual enabledness implying relentless enabledness.

▶ **Example 13.** The path $s_0t_1(s_1t_2s_2t_3)^\omega$ in Figure 1 satisfies $\emptyset$-WFA since there are no perpetually enabled actions in any suffix of the path. However, *cash* is relentlessly enabled in suffixes of this path, and yet does not occur. Hence, this path violates $\emptyset$-SFA.

Finally, we discuss *hyperfairness of actions.* Informally, it says that on all fair paths, every action that can always become enabled must occur infinitely often. The idea is that if there is always a reachable future where the action occurs, then it is merely unlucky if the action does not occur infinitely often. The concept of hyperfairness is introduced and named in [3]. For our presentation of hyperfairness, we use the generalisation from [30]. We first formalise what it means that an action "can become" enabled, by defining *reachability.*

▶ **Definition 14.** *We say that:*

- *A state $s \in \mathcal{S}$ is* $\mathcal{B}$-reachable *from some state $s' \in \mathcal{S}$ if, and only if, there exists a $\mathcal{B}$-free path starting in s' that ends in s.*
- *An action $a \in Act$ is* $\mathcal{B}$-reachable *from some state $s \in \mathcal{S}$ if, and only if, there exists a state $s' \in \mathcal{S}$ that is $\mathcal{B}$-reachable from s and in which a is enabled.*
- *A state $s \in \mathcal{S}$ or action $a \in Act$ is* perpetually $\mathcal{B}$-reachable *on a path π if, and only if, it is $\mathcal{B}$-reachable from every state of π.*
- *A state $s \in \mathcal{S}$ or action $a \in Act$ is* relentlessly $\mathcal{B}$-reachable *on a path π if, and only if, every suffix of π contains a state from which it is $\mathcal{B}$-reachable.*

From the intuitive description of hyperfairness, it is clear it is a variant of weak or strong fairness with reachability instead of enabledness, giving us two possible definitions of hyperfairness. We name the two interpretations weak hyperfairness and strong hyperfairness respectively. Both interpretations of hyperfairness are reasonable, and in fact when not considering blocking actions, they coincide [30]. However, this is not the case when blocking actions are included in the definitions. We therefore consider both variants.

▶ **Definition 15.** *A path π satisfies* weak $\mathcal{B}$-hyperfairness of actions ($\mathcal{B}$-WHFA) *if, and only if, for every suffix π' of π, every action $a \in \overline{\mathcal{B}}$ that is perpetually $\mathcal{B}$-reachable in π' occurs in π'.*

▶ **Definition 16.** *A path π satisfies* strong $\mathcal{B}$-hyperfairness of actions ($\mathcal{B}$-SHFA) *if, and only if, for every suffix π' of π, every action $a \in \overline{\mathcal{B}}$ that is relentlessly $\mathcal{B}$-reachable in π' occurs in π'.*

Since enabledness implies reachability, WHFA is stronger than WFA, and SHFA is stronger than SFA. Perpetually reachability implies relentless reachability, so SHFA is also stronger than WHFA. However, as the next examples will show, SFA and WHFA are incomparable.

▶ **Example 17.** The impact of hyperfairness can clearly be seen when non-determinism is used. Consider the path $s_0t_1s_1t_4(s_3t_6)^\omega$ in Figure 1. This path satisfies $\emptyset$-SFA, since the only action that is relentlessly enabled on this path, *brew*, also occurs infinitely often. However, as long as *deliver* $\notin \mathcal{B}$ and *brew* $\notin \mathcal{B}$, this path does not satisfy $\mathcal{B}$-WHFA or $\mathcal{B}$-SHFA: *deliver* is $\mathcal{B}$-reachable from s_3, and therefore is perpetually and relentlessly $\mathcal{B}$-reachable in a suffix of this path, but does not occur. We here see $\mathcal{B}$-SFA does not imply $\mathcal{B}$-WHFA.

▶ **Example 18.** In Figure 1, consider $s_0t_1(s_1t_2s_2t_3)^\omega$ with $\mathcal{B} = \{order, to\_cash, to\_card\}$. This path satisfies $\mathcal{B}$-WHFA because *card* and *cash* are only $\mathcal{B}$-reachable from s_1 and s_2 respectively. They are not perpetually $\mathcal{B}$-reachable in any suffix of this path, therefore $\mathcal{B}$-WHFA is satisfied. However, they are relentlessly $\mathcal{B}$-reachable, so $\mathcal{B}$-SHFA is violated. This demonstrates that $\mathcal{B}$-WHFA and $\mathcal{B}$-SFHA do not coincide when blocking actions are considered. The actions *card* and *cash* are also relentlessly $\mathcal{B}$-enabled, so $\mathcal{B}$-SFA is also violated. Hence, $\mathcal{B}$-WHFA does not imply $\mathcal{B}$-SFA.

5 A Generalisation of the Property Specification Liveness Patterns

Dwyer, Avrunin and Corbett observed that a significant majority of properties that are used in practice can be fit into a set of property specification patterns [16]. These patterns consist of a *behaviour* that must be satisfied and a *scope* within a path that delimits where the behaviour must be satisfied. We focus on expressing properties that are captured by PSP.

Of all behaviours considered in [16], only *existence*, *existence at least*, *response* and *chain response* represent pure liveness properties. The *global* and *after* scopes, when combined with any of these four behaviours, give liveness properties.[1] All other scopes result in safety properties or properties that combine safety and liveness. Of those, we cover the *until* and *after-until* scopes, since we can incorporate those into our formulae with little difficulty.

For behaviours, *existence at least* says some action in a set S_r must occur at least k times in the scope; when $k = 1$ we call this *existence*. The *response* behaviour requires that whenever an action in a set S_q occurs, it must be followed by the occurrence of an action in S_r. When chains of action occurrences are used instead of individual action occurrences, this is called *chain response*. For the scopes, *global* refers to the full path and *after* to the path after the first occurrence of an action in a set S_a. The *until* scope refers to the path before the first occurrence of an action in a set S_b, or the full path if no such action occurs. Finally, *after-until* combines *after* and *until*, referring to every subpath of the path that starts after any occurrence of an action in S_a and ends before the following occurrence of an action in S_b. If no action in S_b occurs, the behaviour must still be satisfied after S_a.

[1] In the full version, we recap PSP and argue why only these patterns represent pure liveness properties.

▶ **Example 19.** Consider again the properties we presented in Example 4. *Single order* is *absence after-until*, with $S_a = \{order\}$, $S_b = \{deliver\}$ and $S_r = \{order\}$. *Inevitable delivery* is *global response* with $S_q = \{order\}$ and $S_r = \{deliver\}$. *Possible delivery* does not fit into the patterns on occurrences of actions, since it contains a requirement on states, specifically that the state admits a path on which *delivery* occurs.

We want to create formulae for all 16 combinations of the selected behaviours and scopes. To make our results more compact and generic, we first generalise these 16 patterns into a single template property. This template works by describing the shape of a violating path for a property that fits one of these patterns. Intuitively, this shape is: "after the occurrence of ρ, there are no occurrences of α_f up until the (optional) occurrence of α_e". For our template formulae to be syntactically correct, it is important that ρ is a regular formula, describing the prefix that a violating path must have, whereas α_f and α_e are sets of actions. The actions in α_f are those that are forbidden from occurring after ρ on a violating path, whereas the actions in α_e indicate the end of the scope in which α_f may not occur.

We formalise this template as follows:

▶ **Definition 20.** *A path π is $(\rho, \alpha_f, \alpha_e)$-*violating *if, and only if, there exist π_{pre} and π_{suf} such that:*

1. *$\pi = \pi_{pre} \cdot \pi_{suf}$, and*
2. *π_{pre} matches ρ, and*
3. *π_{suf} satisfies at least one of the following conditions:*
   a. *π_{suf} is α_f-free, or*
   b. *π_{suf} contains an occurrence of an action in α_e, and the prefix of π_{suf} before the first occurrence of an action in α_e is α_f-free.*

For readability, we frequently refer to $(\rho, \alpha_f, \alpha_e)$-violating paths as violating paths. We sometimes summarise condition 3 as "π_{suf} is α_f-free up until the first occurrence of α_e". See Figure 2 for an illustration of what types of paths are considered violating.

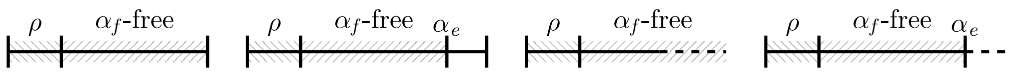

Figure 2 The four types of $(\rho, \alpha_f, \alpha_e)$-violating paths: finite or infinite, and without or with α_e. Always, it has a prefix matching ρ and is α_f-free up until the first occurrence of an action in α_e.

All 16 patterns can indeed be represented by the non-existence of $(\rho, \alpha_f, \alpha_e)$-violating paths, albeit some more directly than others. It turns out that ρ, α_f and α_e can mostly be determined separately for behaviour and scope. For these patterns, α_f is only affected by behaviour and α_e only by scope. However, we must split up the regular formula ρ into a behaviour component, ρ_b, and scope component, ρ_s, such that $\rho = \rho_s \cdot \rho_b$. See Table 1a and Table 1b for how the variables should be instantiated for the four scopes and three of the four behaviours. For a compact representation, we use $\sum$ to generalise the union operator on regular formulae ($+$). We also use x^i to represent i concatenations of x, where $x^0 = \varepsilon$.

We do not include *chain response* in Table 1b, since it does not fit into a single formula. However, it is possible to represent *chain response* as several *response* formulae placed in conjunction with each other.[2]

[2] We give an example of this in the full version of this paper.

Table 1 Variable instantiation for templates.

(a) For scopes.

Scope	ρ_s	α_e
Global	ε	$\emptyset$
Until	ε	S_b
After	$\overline{S_a}^\star \cdot S_a$	$\emptyset$
After-until	$Act^\star \cdot S_a$	S_b

(b) For behaviours.

Behaviour	ρ_b	α_f
Existence	ε	S_r
Existence at least k	$\sum_{0 \le i < k} (\overline{\alpha_e \cup S_r}^\star \cdot S_r)^i$	S_r
Response	$\overline{\alpha_e}^\star \cdot S_q$	S_r
Chain response	-	-

6 Template Formulae

In this section, we present the modal μ-calculus formulae representing the non-existence of a violating path, as defined in Section 5, that satisfies one of the completeness criteria from Section 4. We express the non-existence of such a path, rather than expressing the equivalent notion that all complete paths satisfy the property, because we find the resulting formulae to be more intuitive. We first present a formula for $\mathcal{B}$-progress only. Subsequently, we give the formulae for weak fairness, weak hyperfairness and justness using a common structure all three share. Finally, we present the formulae for strong fairness and strong hyperfairness. In the justness and fairness formulae, $\mathcal{B}$-progress is also included: these assumptions eliminate unrealistic infinite paths, but we still need progress to discard unrealistic finite paths.

Proofs of the theorems in this section are included in the full version of this paper. A sketch of the proof of Theorem 24 is included in Appendix A to illustrate our approach.

6.1 Progress

A formula for the non-existence of a violating path without progress is uninteresting. If progress is not assumed then all finite paths are complete, and therefore a path consisting of just ρ is a violating path whenever $\alpha_f \neq \emptyset$. The non-existence of a violating path would then be captured by $\neg\langle\rho\rangle tt$. This is why we include progress in all our formulae.

To represent progress, we must capture that as long as non-blocking actions are enabled, some transitions must still be executed. The following formula captures the non-existence of violating paths under $\mathcal{B}$-progress:

$$\neg\langle\rho\rangle\nu X.(\langle\alpha_e\rangle tt \vee [\overline{\mathcal{B}}]ff \vee \langle\overline{\alpha_f}\rangle X) \tag{1}$$

Intuitively, this formula says that there is no path that starts with a prefix matching ρ, after which infinitely often a transition can be taken that is not labelled with an action in α_f, or such transitions can be taken finitely often before a state is reached that is $\mathcal{B}$-locked or in which α_e is enabled. In the former case there is a $\mathcal{B}$-progressing path on which no actions in α_f occur after ρ. If a state in which α_e is enabled is reached, then it is guaranteed a violating and $\mathcal{B}$-progressing path exists: by arbitrarily extending the path as long as non-blocking actions are still enabled, a $\mathcal{B}$-progressing and violating path can be constructed.

▶ **Theorem 21.** *A state in an LTS satisfies Formula 1 if, and only if, it does not admit $\mathcal{B}$-progressing paths that are $(\rho, \alpha_f, \alpha_e)$-violating.*

Since representing a liveness pattern without progress leads to uninteresting formulae, it is unsurprising that previous translations of PSP to the μ-calculus have also implicitly included progress. For instance, the translations from [31] for the liveness patterns of PSP are very similar to Formula 1, albeit in positive form and without blocking actions.

6.2 Weak Fairness, Weak Hyperfairness and Justness

For weak fairness, weak hyperfairness and justness, we employ a trick inspired by the formula for justness presented in [6] (which was in turn inspired by [11]): we translate a requirement on a full path into an invariant that can be evaluated within finitely many steps from every state of the path. We illustrate this using weak fairness.

On every suffix of a weakly fair path, every perpetually enabled non-blocking action occurs. To turn this into an invariant, we observe that we can evaluate a property on all suffixes of a path by evaluating it from every state of the path instead. Next we must determine, within finitely many steps, if an action is perpetually enabled on a possibly infinite path. We do this by observing that if an action is not perpetually enabled, it must become disabled within finitely many steps. An equivalent definition of WFA therefore is: a path π satisfies WFA if, and only if, for every state s in π, every action $a \in \overline{\mathcal{B}}$ that is enabled in s occurs or becomes disabled within finitely many steps on the suffix of π starting in s. This translation of WFA determines three things for every non-blocking action a. First, which actions may need to occur because of a; in the case of WFA this is a itself. Second, when those actions need to occur; for WFA this is when a is enabled. We refer to this as the action being "on". Finally, when those actions do not need to occur; for WFA this is when a becomes disabled. We refer to this as the action being "off". When an action that was previously on becomes off, or one of the required actions occurs, we say the action is "eliminated". By choosing different definitions for an action being on or off, and when an action is eliminated, we can also represent justness and weak hyperfairness in the same way.

We find that completeness criteria for which such a translation can be made can be represented using the same generalised formula. We will present this formula and how to instantiate it for WFA, WHFA and JA. However, we must first formalise what it means for a predicate on paths to be translatable to an invariant that can be evaluated within finitely many steps. We introduce the term *finitely realisable (path) predicates* for this purpose.

▶ **Definition 22.** *A path predicate P is* finitely realisable *if, and only if, there exist mappings ϕ_{on} and ϕ_{of} from non-blocking actions to closed modal μ-calculus formulae, and a mapping α_{el} from non-blocking actions to sets of actions, such that:*

1. *A path π satisfies predicate P if, and only if, all states s on π satisfy the following: for all $a \in \overline{\mathcal{B}}$, if s satisfies $\phi_{on}(a)$ then the suffix π' of π starting in s must contain an occurrence of some action in $\alpha_{el}(a)$ or a state that satisfies $\phi_{of}(a)$.*
2. *A state s is a $\mathcal{B}$-locked state if, and only if, $s \notin [\![\phi_{on}(a)]\!]$ for all $a \in \overline{\mathcal{B}}$.*
3. *For every state s and for all $a \in \overline{\mathcal{B}}$, $s \in [\![\phi_{on}(a)]\!]$ implies $s \notin [\![\phi_{of}(a)]\!]$.*
4. *For all states s and all $a \in \overline{\mathcal{B}}$ such that $s \in [\![\phi_{on}(a)]\!]$, if there exists a finite path π from s to a state s' such that there is no occurrence of an action in $\alpha_{el}(a)$ on π and there is no state on π that satisfies $\phi_{of}(a)$, then $s' \in [\![\phi_{on}(a)]\!]$.*

We refer to these four properties as the invariant property*, the* locking property*, the* exclusive property *and the* persistent property*, respectively.*

The general formula for finitely realisable predicates is as follows:

$$\neg\langle\rho\rangle\nu X.(\bigwedge_{a\in\overline{\mathcal{B}}}(\phi_{on}(a) \Rightarrow \langle\overline{\alpha_f}^\star\rangle(\langle\alpha_e\rangle tt \vee (\phi_{of}(a) \wedge X) \vee \langle\alpha_{el}(a) \setminus \alpha_f\rangle X))) \tag{2}$$

This formula has similarities to Formula 1, particularly how ρ and α_e are integrated. The important part is that after ρ, it must invariantly hold that all non-blocking actions for which $\phi_{on}(a)$ is satisfied are later eliminated. An action a is eliminated if, within finitely many steps, $\phi_{of}(a)$ is satisfied or an action in $\alpha_{el}(a)$ occurs. In both cases, the invariant must once again hold. After ρ, no actions in α_f may occur. The formula works correctly for finite paths as well as infinite ones: if it is possible to reach a $\mathcal{B}$-locked state after ρ without taking actions in α_f, then X is satisfied due to the locking property, and a violating path is found.

Formula 2 is a template formula in two ways: ρ, α_f and α_e determine what property is captured, and ϕ_{on}, ϕ_{of} and α_{el} determine the completeness criterion. In this paper, we only cover how to instantiate the formula for WFA, WHFA and JA, but it can also be used for other finitely realisable predicates. However, the correctness proof of the formula depends on the criterion being *feasible*. Feasibility on paths [2] is defined as follows.

▶ **Definition 23.** *A predicate on paths P is* feasible *if, and only if, for every LTS M, every finite path π in M can be extended to a path π' that satisfies P and is still a valid path in M.*

That WFA, WHFA and JA are indeed feasible for finite LTSs is proven in the full version.

▶ **Theorem 24.** *For all feasible and finitely realisable path predicates P, it holds that an LTSC satisfies Formula 2 if, and only if, its initial state does not admit $\mathcal{B}$-progressing paths that satisfy P and are $(\rho, \alpha_f, \alpha_e)$-violating.*

By instantiating the theorem for each completeness criterion, we derive the following:

▶ **Corollary 25.** *A state in an LTS satisfies Formula 2 with $\phi_{on}(a) = \langle a \rangle tt$, $\phi_{of}(a) = [a]\mathit{ff}$ and $\alpha_{el}(a) = \{a\}$ for all $a \in \overline{\mathcal{B}}$ if, and only if, it does not admit $\mathcal{B}$-progressing paths that satisfy $\mathcal{B}$-weak fairness of actions and are $(\rho, \alpha_f, \alpha_e)$-violating.*

▶ **Corollary 26.** *A state in an LTS satisfies Formula 2 with $\phi_{on}(a) = \langle \overline{\mathcal{B}}^{\star} \cdot a \rangle tt$, $\phi_{of}(a) = [\overline{\mathcal{B}}^{\star} \cdot a]\mathit{ff}$ and $\alpha_{el}(a) = \{a\}$ for all $a \in \overline{\mathcal{B}}$ if, and only if, it does not admit $\mathcal{B}$-progressing paths that satisfy weak $\mathcal{B}$-hyperfairness of actions and are $(\rho, \alpha_f, \alpha_e)$-violating.*

▶ **Corollary 27.** *A state in an LTSC satisfies Formula 2 with $\phi_{on}(a) = \langle a \rangle tt$, $\phi_{of}(a) = \mathit{ff}$ and $\alpha_{el}(a) = \{b \in Act \mid a \not\rightsquigarrow^{\bullet} b\}$ for all $a \in \overline{\mathcal{B}}$ if, and only if, it does not admit $\mathcal{B}$-progressing paths that satisfy $\mathcal{B}$-justness of actions and are $(\rho, \alpha_f, \alpha_e)$-violating.*

6.3 Strong Fairness and Strong Hyperfairness

SFA is not finitely realisable because we cannot observe within finitely many steps whether an action is relentlessly enabled: even if we observe several times that it is disabled, it may still be infinitely often enabled along the whole path. Hence, we cannot use Formula 2.

Instead we observe that, on a path, actions that are not relentlessly enabled must eventually become perpetually disabled. If the path is strongly fair, then all relentlessly enabled non-blocking actions occur infinitely often. We can therefore say that a path is strongly fair if we can divide all non-blocking actions into two disjoint sets: those that occur infinitely often and those that eventually become perpetually disabled. This observation is also made in [36], where a μ-calculus formula for termination under strong fairness is given.

Using this idea, we give the following template formula for SFA:

$$\neg\langle \rho \cdot \overline{\alpha_f}^{\star} \rangle (\langle \alpha_e \rangle tt \vee [\overline{\mathcal{B}}]\mathit{ff} \vee \bigvee_{\emptyset \neq F \subseteq \overline{\mathcal{B}}} \nu X.(\bigwedge_{a \in F} \mu W.((\bigwedge_{b \in \overline{\mathcal{B}} \setminus F} [b]\mathit{ff}) \wedge (\langle a \setminus \alpha_f \rangle X \vee \langle \overline{\alpha_f} \rangle W)))) \quad (3)$$

The use of negation, the exclusion of α_f, and ρ in the diamond operator at the start of this formula are the same as in Formula 1. We explain the start of the formula after addressing the part starting with $\bigvee_{\emptyset \neq F \subseteq \overline{\mathcal{B}}}$. Here, we use that on a strongly fair path, all non-blocking actions can be divided into those that occur infinitely often and those that become perpetually disabled. The disjunction over subsets considers all possible ways of selecting some non-empty subset F of $\overline{\mathcal{B}}$ that should occur infinitely often. The greatest fixpoint states that infinitely often, all those actions must indeed occur within finitely many steps. Additionally, at no point may a non-blocking action not in F be enabled. We exclude $F = \emptyset$ because the logic of the greatest fixed point formula we give relies on there being at least one a in F. The special case that F is empty and therefore a $\mathcal{B}$-locked state should be reached, is instead covered by explicitly considering $[\overline{\mathcal{B}}]\mathit{ff}$ earlier in the formula. Returning to the start of the formula, we allow a finite α_f-free path before the greatest fixpoint is satisfied. The reason is that it may take several steps before all the non-blocking actions that are only finitely often enabled become perpetually disabled. Since we include a finite prefix already, we also add the cases that an action in α_e becomes enabled or that a $\mathcal{B}$-locked state is reached here, rather than deeper into the formula like in Formula 2.

▶ **Theorem 28.** *An LTS satisfies Formula 3 if, and only if, its initial state does not admit $\mathcal{B}$-progressing paths that satisfy $\mathcal{B}$-strong fairness of actions and are $(\rho, \alpha_f, \alpha_e)$-violating.*

Due to the quantification over subsets, the formula is exponential in the number of actions in $\overline{\mathcal{B}}$. Beyond small models, it is therefore not practical. However, it can serve as a basis for future work. For instance, if fairness is applied to sets of actions rather than individual actions, the formula is exponential in the number of sets instead, which may be smaller depending on how the sets are formed [35].

We can adapt the formula for strong fairness to a formula for strong hyperfairness, by replacing perpetual disabledness of non-blocking actions not in F with perpetual unreachability.

$$\neg\langle \rho \cdot \overline{\alpha_f}^\star\rangle(\langle\alpha_e\rangle tt \vee [\overline{\mathcal{B}}]\mathit{ff} \vee \bigvee_{\emptyset\neq F\subseteq\overline{\mathcal{B}}} \nu X.(\bigwedge_{a\in F} \mu W.((\bigwedge_{b\in\overline{\mathcal{B}}\setminus F} [\overline{\mathcal{B}}^\star \cdot b]\mathit{ff}) \wedge (\langle a\setminus\alpha_f\rangle X \vee \langle\overline{\alpha_f}\rangle W)))) \quad (4)$$

▶ **Theorem 29.** *An LTS satisfies Formula 4 if, and only if, its initial state does not admit a $\mathcal{B}$-progressing path that satisfies strong $\mathcal{B}$-hyperfairness of actions and is $(\rho, \alpha_f, \alpha_e)$-violating.*

Since we are not aware of other completeness criteria that fit the same structure, we do not provide a generalised formula here like we did with Formula 2.

7 Application Example

We here give an example of an application of the template formulae. In [25], several mutual exclusion algorithms are analysed using the mCRL2 toolset. Their analysis of Dekker's algorithm [14] presents the following modal μ-calculus formula for starvation freedom of processes with id's 0 and 1. For clarity, the notation has been adjusted to match the previous sections and action names have been simplified.

$$[Act^\star] \bigwedge_{i\in\{0,1\}} [\{\mathit{wish\_flag}(i,b) \mid b\in\mathbb{B}\}]\mu X.([\overline{\mathit{enter}(i)}]X \wedge \langle Act\rangle tt) \quad (5)$$

Starvation freedom is a *global response* property. In this case, the starvation freedom of a process i is represented as an instantiation of the pattern with $S_q = \{\mathit{wish\_flag}(i,b) \mid b \in \mathbb{B}\}$ and $S_r = \{\mathit{enter}(i)\}$. Indeed, the above formula is equivalent to:

$$\bigwedge_{i\in\{0,1\}} \neg\langle Act^\star \cdot \{\mathit{wish\_flag}(i,b) \mid b\in\mathbb{B}\}\rangle\nu X.(\langle\emptyset\rangle tt \vee [Act]\mathit{ff} \vee \langle\overline{\mathit{enter}(i)}\rangle X) \quad (6)$$

Observe that, when taking $\mathcal{B} = \emptyset$, the above matches a conjunction of two instances of Formula 1, taking ρ, α_e and α_f as suggested in Table 1 for *global response*. Thus, this formula captures starvation freedom under $\emptyset$-progress. In [25], it is reported that mCRL2 finds a violating path for this formula; a path which the authors note is unfair. The exact fairness assumption considered is not made concrete. As an ad-hoc solution, the modal μ-calculus formula is adjusted to specifically ignore that counterexample. Subsequently, mCRL2 finds another counterexample, which the authors again claim is unfair. Instead of creating yet another formula, they move on to Peterson's algorithm, which is deemed easier to analyse. Using our template formulae, we can easily produce a formula for starvation freedom under several different completeness criteria. We give the formula for $\emptyset$-WFA, as an example.

$$\bigwedge_{i\in\{0,1\}} \neg\langle Act^\star \cdot \{\mathit{wish\_flag}(i,b) \mid b \in \mathbb{B}\}\rangle \\ \nu X.(\bigwedge_{a\in Act} (\langle a\rangle tt \Rightarrow \langle\overline{\mathit{enter}(i)}^\star\rangle(\langle\emptyset\rangle tt \vee ([a]f\!f \wedge X) \vee \langle a \setminus \mathit{enter}(i)\rangle X))) \tag{7}$$

We check this formula on the model from [25] using mCRL2. Since mCRL2 only supports quantification over data parameters and not over actions, the conjunction over Act must be written out explicitly. The tool reports that the formula is violated. Examining the counterexample reveals this is because actions in the model do not show which process performs the action. Therefore, process i reading value v from a register r is labelled with the same action as process j reading v from r. We add the responsible process to each action label, and also define $\mathcal{B} = \{\mathit{wish\_flag}(i,i,b) \mid i \in \{0,1\}, b \in \mathbb{B}\}$, to capture that processes are allowed to remain in their non-critical section indefinitely. This was not considered in Formula 5, but it is part of the mutual exclusion problem [15, 22]. The tool reports that the modified formula is satisfied. We can therefore conclude that Dekker's algorithm satisfies starvation freedom when assuming weak fairness of actions, as long as it is taken into account for each action which process is responsible for it.

Our other formulae can be used in similar ways. An example of how to use the justness formula in mCRL2, including a method for encoding the concurrency relation, is given in [6].

8 Discussion

In this section, we briefly reflect on the coverage of the properties we consider, and our choice in focusing on the modal μ-calculus.

Firstly, we have exclusively addressed liveness properties in this paper thus far. As indicated previously, the problem we are considering primarily crops up for these properties. This is because, as pointed out in [22], when a completeness criterion is feasible, assuming the criterion holds true or not has no impact on whether a safety property is satisfied or not. The reason is that for safety properties on paths, any path that violates the property must contain a finite prefix such that any extension of that prefix also violates the property [1]. Therefore, if a completeness criterion is feasible, then whenever a model contains incomplete paths that violate a safety property it also contains complete paths that violate the property. All completeness criteria discussed in Section 4 are feasible with respect to finite LTSs, and hence we do not need to consider patterns that capture safety properties. For modal μ-calculus formulae for the safety properties of PSP, without integrated completeness criteria, we refer to [31] and [34]. For properties that are a combination of safety and liveness, the components can be turned into separate formulae and checked separately.

Readers may also wonder about alternative methods of representing properties under completeness criteria, such as using LTL. As indicated in Section 3, there are many contexts where we also want to consider non-linear properties, and hence the modal μ-calculus is preferred. Automatic translations from LTL to the modal μ-calculus exist, but can be exponential in complexity [13] and it is unclear at this time if this blow-up is avoided in this case. Anecdotal evidence [33] suggests this is not the case for existing translations. In [22] several completeness criteria are represented in LTL, but it is noted that this translation requires introducing new atomic propositions which hides the complexity of this translation. The representation of hyperfairness in particular may be expensive, since atomic propositions for all reachable actions are required. It is also unclear how to combine LTL-based translations effectively with symbolic model checking approaches. For these reasons, a direct representation in the modal μ-calculus is preferable.

9 Conclusion

In this paper, we have presented formulae for liveness properties under several completeness criteria. As part of this, we defined a property template that generalises the liveness properties of PSP, which has been estimated to cover a majority of properties found in the literature [16]. The completeness criteria covered are progress, justness, weak fairness, strong fairness, and hyperfairness, all defined with respect to actions and parameterised with a set of blocking actions. The formulae have all been manually proven to be correct.

For future work, one goal is to formalise our manual proofs using a proof assistant. Another avenue for future work is extending our formulae to cover a wider range of completeness criteria and properties. We suggest some potential extensions here.

One of our contributions is the identification of a shared common structure underlying justness, weak fairness and weak hyperfairness: they are finitely realisable path predicates. Our formula for such predicates can be adapted to arbitrary feasible finitely realisable path predicates. While we do not have such a generic formula for other completeness criteria, our characterisation of $(\rho, \alpha_f, \alpha_e)$-violating paths can be used as a basis to express the non-existence of complete paths violating many common properties for different notions of completeness as well, as we demonstrate with strong fairness and strong hyperfairness. We are especially interested in extending our formulae to allow fairness over sets of actions, rather than individual actions, similar to the task-based definitions from [24].

In terms of properties, we can look at proposed extensions of PSP, such as those suggested in [12]. There is also the *constrained chain* behaviour, which is a modification of precedence chain and response chain given in [16]. There are extensions of PSP to real-time [4, 28] and probabilistic [27] contexts as well. Finally, in [5] the formula from [6] that formed the basis of Formula 2 is extended to also include state information.

There are therefore many potentially useful extensions of the formulae presented in this paper. However, the presented template formulae already cover many completeness criteria and liveness properties, making them useful for model checking in practice.

References

1 Mack W. Alford, Leslie Lamport, and Geoff P. Mullery. Basic concepts. In Mack W. Alford, Jean-Pierre Ansart, Günter Hommel, Leslie Lamport, Barbara Liskov, Geoff P. Mullery, and Fred B. Schneider, editors, *Distributed Systems: Methods and Tools for Specification, An Advanced Course, April 3-12, 1984 and April 16-25, 1985, Munich, Germany*, volume 190 of *Lecture Notes in Computer Science*, pages 7–43. Springer, 1984. doi:10.1007/3-540-15216-4_12.

2 Krzysztof R. Apt, Nissim Francez, and Shmuel Katz. Appraising fairness in languages for distributed programming. *Distributed Comput.*, 2(4):226–241, 1988. doi:10.1007/BF01872848.

3 Paul C. Attie, Nissim Francez, and Orna Grumberg. Fairness and hyperfairness in multi-party interactions. In Frances E. Allen, editor, *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, January 1990*, pages 292–305. ACM Press, 1990. doi:10.1145/96709.96739.

4 Pierfrancesco Bellini, Paolo Nesi, and Davide Rogai. Expressing and organizing real-time specification patterns via temporal logics. *J. Syst. Softw.*, 82(2):183–196, 2009. doi:10.1016/j.jss.2008.06.041.

5 Mark S. Bouwman. *Supporting Railway Standardisation with Formal Verification*. Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science, Eindhoven University of Technology, 2023. URL: https://pure.tue.nl/ws/portalfiles/portal/307965423/20231023_Bouwman_hf.pdf.

6 Mark S. Bouwman, Bas Luttik, and Tim A. C. Willemse. Off-the-shelf automated analysis of liveness properties for just paths. *Acta Informatica*, 57(3-5):551–590, 2020. doi:10.1007/s00236-020-00371-w.

7 Julian C. Bradfield and Colin Stirling. Modal logics and mu-calculi: an introduction. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 293–330. Elsevier Science, 2001. doi:10.1016/b978-044482830-9/50022-9.

8 Julian C. Bradfield and Colin Stirling. Modal mu-calculi. In Patrick Blackburn, Johan Van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in logic and practical reasoning*, pages 721–756. Elsevier, 2007. doi:10.1016/s1570-2464(07)80015-2.

9 Julian C. Bradfield and Igor Walukiewicz. The mu-calculus and model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 871–919. Springer, 2018. doi:10.1007/978-3-319-10575-8_26.

10 Olav Bunte, Jan Friso Groote, Jeroen J. A. Keiren, Maurice Laveaux, Thomas Neele, Erik P. de Vink, Wieger Wesselink, Anton Wijs, and Tim A. C. Willemse. The mCRL2 toolset for analysing concurrent systems. In Tomáš Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II*, volume 11428 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2019. doi:10.1007/978-3-030-17465-1_2.

11 Edmund M. Clarke, Orna Grumberg, Kenneth L. McMillan, and Xudong Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In Bryan Preas, editor, *Proceedings of the 32st Conference on Design Automation, San Francisco, California, USA, Moscone Center, June 12-16, 1995*, pages 427–432. ACM Press, 1995. doi:10.1145/217474.217565.

12 Rachel L. Cobleigh, George S. Avrunin, and Lori A. Clarke. User guidance for creating precise and accessible property specifications. In Michal Young and Premkumar T. Devanbu, editors, *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2006, Portland, Oregon, USA, November 5-11, 2006*, pages 208–218. ACM, 2006. doi:10.1145/1181775.1181801.

13 Sjoerd Cranen, Jan Friso Groote, and Michel A. Reniers. A linear translation from CTL* to the first-order modal μ-calculus. *Theor. Comput. Sci.*, 412(28):3129–3139, 2011. doi:10.1016/j.tcs.2011.02.034.

14 Edsger W Dijkstra. Over de sequentialiteit van procesbeschrijvingen (EWD-35). EW dijkstra archive. *Center for American History, University of Texas at Austin*, 1962. URL: https://www.cs.utexas.edu/~EWD/ewd00xx/EWD35.PDF.

15 Edsger W. Dijkstra. Solution of a problem in concurrent programming control. *Commun. ACM*, 8(9):569, 1965. doi:10.1145/365559.365617.

16 Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In Barry W. Boehm, David Garlan, and Jeff Kramer, editors, *Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, May 16-22, 1999*, pages 411–420. ACM, 1999. doi:10.1145/302405.302672.

17 E. Allen Emerson and Edmund M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982. doi:10.1016/0167-6423(83)90017-5.

18 Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979. doi:10.1016/0022-0000(79)90046-1.

19 Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2011: a toolbox for the construction and analysis of distributed processes. *Int. J. Softw. Tools Technol. Transf.*, 15(2):89–107, 2013. doi:10.1007/s10009-012-0244-z.

20 Rob J. van Glabbeek. Justness - A completeness criterion for capturing liveness properties (extended abstract). In Mikołaj ojańczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11425 of *Lecture Notes in Computer Science*, pages 505–522. Springer, 2019. doi:10.1007/978-3-030-17127-8_29.

21 Rob J. van Glabbeek. Reactive temporal logic. In Ornela Dardha and Jurriaan Rot, editors, *Proceedings Combined 27th International Workshop on Expressiveness in Concurrency and 17th Workshop on Structural Operational Semantics, EXPRESS/SOS 2020, and 17th Workshop on Structural Operational SemanticsOnline, 31 August 2020*, volume 322 of *EPTCS*, pages 51–68. Open Publishing Association, 2020. doi:10.4204/EPTCS.322.6.

22 Rob J. van Glabbeek. Modelling mutual exclusion in a process algebra with time-outs. *Inf. Comput.*, 294:105079, 2023. doi:10.1016/j.ic.2023.105079.

23 Rob J. van Glabbeek and Peter Höfner. CCS: It's not fair! fair schedulers cannot be implemented in CCS-like languages even under progress and certain fairness assumptions. *Acta Informatica*, 52(2-3):175–205, 2015. doi:10.1007/s00236-015-0221-6.

24 Rob J. van Glabbeek and Peter Höfner. Progress, Justness, and Fairness. *ACM Comput. Surv.*, 52(4):69:1–69:38, 2019. doi:10.1145/3329125.

25 Jan Friso Groote and Jeroen J. A. Keiren. Tutorial: designing distributed software in mCRL2. In Kirstin Peters and Tim A. C. Willemse, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 41st IFIP WG 6.1 International Conference, FORTE 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings*, volume 12719 of *Lecture Notes in Computer Science*, pages 226–243. Springer, 2021. doi:10.1007/978-3-030-78089-0_15.

26 Jan Friso Groote and Mohammad Reza Mousavi. *Modeling and Analysis of Communicating Systems*. MIT Press, August 2014. URL: https://mitpress.mit.edu/books/modeling-and-analysis-communicating-systems.

27 Lars Grunske. Specification patterns for probabilistic quality properties. In Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn, editors, *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 31–40. ACM, 2008. doi:10.1145/1368088.1368094.

28 Sascha Konrad and Betty H. C. Cheng. Real-time specification patterns. In Gruia-Catalin Roman, William G. Griswold, and Bashar Nuseibeh, editors, *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*, pages 372–381. ACM, 2005. doi:10.1145/1062455.1062526.

29 Dexter Kozen. Results on the propositional μ-calculus. *Theor. Comput. Sci.*, 27(3):333–354, 1983. doi:10.1016/0304-3975(82)90125-6.

30 Leslie Lamport. Fairness and hyperfairness. *Distributed Comput.*, 13(4):239–245, 2000. doi:10.1007/PL00008921.

31 Radu Mateescu. Property Pattern Mappings for RAFMC, 2019. Available at: https://cadp.inria.fr/resources/evaluator/rafmc.html (Accessed: 26 January 2024).

32 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

33 Jaco van de Pol and Michael Weber. A multi-core solver for parity games. *Electronic Notes in Theoretical Computer Science*, 220(2):19–34, 2008. Proceedings of the 7th International Workshop on Parallel and Distributed Methods in verifiCation (PDMC 2008). doi:10.1016/j.entcs.2008.11.011.

34 Daniela Remenska. *Bringing Model Checking Closer To Practical Software Engineering*. PhD thesis, Vrije U., Amsterdam, 2016. PhD Thesis, available at: https://hdl.handle.net/1871/53958.

35 Myrthe S. C. Spronck. Fairness assumptions in the modal μ-calculus, 2023. Master's thesis, Eindhoven University of Technology, available at https://research.tue.nl/en/studentTheses/fairness-assumptions-in-the-modal-%C2%B5-calculus.

36 Frank A. Stomp, Willem-Paul de Roever, and Rob T. Gerth. The μ-calculus as an assertion-language for fairness arguments. *Inf. Comput.*, 82(3):278–322, 1989. doi:10.1016/0890-5401(89)90004-7.

A Proof Sketch

Full proofs are included in the appendices of the full version of this paper. Here, we provide an outline of the proof of Theorem 24 by presenting all the supporting lemmas and propositions proven. We include brief sketches of how we prove these claims, but not the full proofs. This is done to illustrate the approach we have taken. This appendix corresponds to Appendix D.3 in the full version. We begin with restating Theorem 24.

▶ **Theorem 24.** *For all feasible and finitely realisable path predicates P, it holds that an LTSC satisfies Formula 2 if, and only if, its initial state does not admit $\mathcal{B}$-progressing paths that satisfy P and are $(\rho, \alpha_f, \alpha_e)$-violating.*

The following propositions[3] give properties of finitely realisable paths that can be derived from the invariant, locking, exclusive and persistent properties.

▶ **Proposition 46.** *Every $\mathcal{B}$-progressing, finite path satisfies every finitely realisable path predicate, for all $\mathcal{B} \subseteq Act$.*

▶ **Proposition 47.** *Every path that satisfies a finitely realisable path predicate P is $\mathcal{B}$-progressing.*

▶ **Proposition 48.** *If a path π satisfies finitely realisable path predicate P, then every path of which π is a suffix also satisfies P.*

▶ **Proposition 49.** *If a path π satisfies a finitely realisable path predicate P, then every suffix of π also satisfies P.*

Proposition 46 follows from the invariant, locking, and persistent properties. For Proposition 47, we use the invariant, locking, and exclusive properties, and for Proposition 48, the invariant and persistent property are enough. Finally, Proposition 49 follows directly from the invariant property.

For the proof of the main theorem, we fix an LTS M, as well as $\mathcal{B}$, ρ, α_f and α_e. We also fix a feasible, finitely realisable path predicate P. To characterise the semantics of the formula, we first split it into multiple smaller subformulae.

[3] We use the same numbering here as in the full version, hence the jump to 46.

$$\begin{aligned}
&\mathit{violate}_G && = \langle \rho \rangle \mathit{invariant}_G \\
&\mathit{invariant}_G && = \nu X.(\bigwedge_{a \in \overline{\mathcal{B}}} (\phi_{on}(a) \Rightarrow \mathit{eliminate}_G(a))) \\
&\mathit{eliminate}_G(a) && = \langle \overline{\alpha_f}^\star \rangle (\langle \alpha_e \rangle tt \vee (\phi_{of}(a) \wedge X) \vee \langle \alpha_{el}(a) \setminus \alpha_f \rangle X)
\end{aligned}$$

We have that Formula 2 = $\neg \mathit{violate}_G$.

The proof proceeds by characterising the semantics of every subformulae. We define the length of a path to be the number of transitions. A path of length 0 is called the empty path.

▶ **Lemma 50.** *For all environments e, states $s \in \mathcal{S}$, actions $a \in \overline{\mathcal{B}}$ and sets $\mathcal{F} \subseteq \mathcal{S}$, it holds that $s \in [\![\mathit{eliminate}_G(a)]\!]_{e[X:=\mathcal{F}]}$ if, and only if, s admits a finite path π with final state s_{final} that satisfies the following conditions:*

1. *π is α_f-free, and*
2. *one of the following three holds:*
   a. *at least one action in α_e is enabled in s_{final}, or*
   b. *$s_{\mathit{final}} \in \mathcal{F}$ and s_{final} satisfies $\phi_{of}(a)$, or*
   c. *$s_{\mathit{final}} \in \mathcal{F}$ and the last transition in π, t_{final}, is labelled with an action in $\alpha_{el}(a) \setminus \alpha_f$.*

In the full proof of Lemma 50, we use another supporting proposition that characterises the semantics of a simple least fixpoint formula that generalises $\mathit{eliminate}_G(a)$, and then show in detail how the lemma follows. For this overview, we only give an intuitive explanation: the $\langle \overline{\alpha_f}^\star \rangle$ part of $\mathit{eliminate}_G(a)$ gives us the finite, α_f-free path π that the lemma refers to. The conditions on the final state of this path follow from the rest of the formula: $\langle \alpha_e \rangle tt$ considers the possibility that an action in α_e is enabled; $\phi_{of}(a) \wedge X$ represents reaching a state in $\mathcal{F}$ where $\phi_{of}(a)$ is satisfied (recall that Lemma 50 refers to the environment where X is mapped to $\mathcal{F}$); and $\langle \alpha_{el}(a) \setminus \alpha_f \rangle X$ appends one extra transition to a state in $\mathcal{F}$, eliminating a. Since the total path has to be α_f-free, the eliminating action may not be in α_f.

The next step is $\mathit{invariant}_G$. This formula exactly describes those states that admit paths that are $\mathcal{B}$-progressing, $(\varepsilon, \alpha_f, \alpha_e)$-violating and satisfy P. Since ε is the empty sequence, we are ignoring the ρ-prefix for now. We define the set S_G to be exactly those states in M that admit a path π meeting the following conditions:

- π satisfies P, and
- π is $\mathcal{B}$-progressing, and
- π satisfies one of the following conditions:
  - π is α_f-free, or
  - π contains an occurrence of an action in α_e, and the prefix of π before the first occurrence of an action in α_e is α_f-free.

Our goal is then to prove that $[\![\mathit{invariant}_G]\!]_e = S_G$. This takes two steps: first we prove that S_G is a fixed point of the transformer characterising $\mathit{invariant}_G$, and then that it is the *greatest* fixed point.

▶ **Lemma 51.** *S_G is a fixed point of the transformer T_G defined by:*

$$T_G(\mathcal{F}) = \bigcap_{a \in \overline{\mathcal{B}}} \left\{ s \in \mathcal{S} \mid s \in [\![\phi_{on}(a)]\!]_{e[X:=\mathcal{F}]} \Rightarrow s \in [\![\mathit{eliminate}_G(a)]\!]_{e[X:=\mathcal{F}]} \right\}$$

Proving that S_G is a fixed point means proving $T_G(S_G) = S_G$, which we do by mutual set inclusion. We briefly explain how we reach the conclusion that if a state s is in $T_G(S_G)$, then it must also be in S_G. If there are no non-blocking actions a such that $s \in [\![\phi_{on}(a)]\!]$, then the empty path witnesses that s is in S_G; for this we use the locking property as well as Proposition 46. If there is an action $a \in \overline{\mathcal{B}}$ such that $s \in [\![\phi_{on}(a)]\!]$, then we know from $s \in T_G(S_G)$ that $s \in [\![\mathit{eliminate}_G]\!]_{e[X:=S_G]}$. Then Lemma 50 yields a finite path π that is α_f-free and on which a state in which α_e is enabled is reached, or a is eliminated and a state in S_G is reached. In the former case, we can use feasibility of P and Proposition 47 to find a path that satisfies P and is $\mathcal{B}$-progressing, and that is also α_f-free until the first occurrence of an action in α_e. Hence, we find a path that witnesses $s \in S_G$. If a is eliminated instead, then since π reaches a state that is in S_G, we can create a path $\pi'' = \pi \cdot \pi'$, where π' is a path from the final state of π that is $\mathcal{B}$-progressing, satisfies P, and is α_f-free up until the first occurrence of an action in α_e. Using Proposition 48, we can then show π'' witnesses $s \in S_G$. The other part of the proof, that an arbitrary state in S_G is also in $T_G(S_G)$, works very similarly, just in the other direction. We need Proposition 49 in that part of the proof in place of Proposition 48.

To prove that S_G is actually the greatest fixed point of T_G, we use the following supporting lemma:

▶ **Lemma 52.** *For all states s in a fixed point $\mathcal{F}$ of T_G as defined in Lemma 51, if there is no action in α_e that is reachable from s without doing an action in α_f and there exists at least one action $a \in \overline{\mathcal{B}}$ such that s satisfies $\phi_{on}(a)$, then there exists a finite path π from s to some state s' meeting all of the following conditions:*

1. *$s' \in \mathcal{F}$, and*
2. *π has length at least one, and*
3. *π is α_f-free, and*
4. *for all actions $a \in \overline{\mathcal{B}}$ such that s satisfies $\phi_{on}(a)$, there is a state on π that satisfies $\phi_{of}(a)$ or there is a transition on π labelled with an action in $\alpha_{el}(a)$.*

However, for an intuitive explanation of the proof that S_G is the greatest fixed point of T_G we do not need this supporting lemma. We therefore do not go into its proof here.

▶ **Lemma 53.** *S_G is the greatest fixed point of the transformer T_G as defined in Lemma 51.*

In the proof of Lemma 53, we take an arbitrary state s in an arbitrary fixed point $\mathcal{F}$ of T_G, and then prove that $s \in S_G$. This is done by constructing a path π from s that satisfies P, is $\mathcal{B}$-progressing, and $(\varepsilon, \alpha_f, \alpha_e)$-violating. The proof considers three cases. The first case is when s is $\mathcal{B}$-locked. In that case, the empty path is trivially $\mathcal{B}$-progressing and violating, and by Proposition 46 also satisfies P. The second case is that it is possible to reach a state in which an action in α_e is enabled without taking actions in α_f. If this is the case, then that path can be extended using feasibility of P to create a path that is violating, satisfies P, and, by Proposition 47, is $\mathcal{B}$-progressing. The most complicated case is the one in which neither of the previous two is true. The idea is that we construct a path from s by repeatedly adding α_f-free path segments to an initial path $\pi = s$, in a potentially endless construction. In every iteration, we consider whether the final state of the path constructed thus far is $\mathcal{B}$-locked. If so, then, similar to the first case, we have found a witness for $s \in S_G$. If not, then there is some non-blocking action a that is "on". And therefore, by the definition of T_G, there is a finite α_f-free path on which a is eliminated. We can disregard the possibility that we instead reach a state in which α_e is enabled, since we addressed that case separately. The segment we append to π is the finite α_f-free path on which a is eliminated. By the persistent property, non-blocking actions for which ϕ_{on} is satisfied but are not eliminated remain "on", and hence can be eliminated later in π. We can therefore simply keep track of all the actions for which ϕ_{on} is satisfied in states we encounter, and eliminate them all in turn. This produces an infinite, and therefore $\mathcal{B}$-progressing, path that satisfies P, and that is entirely α_f-free. So in this case too, we construct a path that witnesses $s \in S_G$.

Since the semantics of $\mathit{invariant}_G$ are characterised as the greatest fixed point of T_G, we can conclude the following from the definition of S_G.

▶ **Corollary 54.** *The set of states characterised by $\mathit{invariant}_G$ is exactly the set of states that admit $\mathcal{B}$-progressing, $(\varepsilon, \alpha_f, \alpha_e)$-violating paths that satisfy P.*

We then prepend the ρ part of the formula.

▶ **Lemma 55.** *For all environments e and states $s \in \mathcal{S}$, it holds that $s \in [\![\mathit{violate}_G]\!]_e$ if, and only if, s admits a path that is $\mathcal{B}$-progressing, satisfies P and is $(\rho, \alpha_f, \alpha_e)$-violating.*

This step is rather trivial, since it follows directly from Corollary 54 and the basic definition of the modal μ-calculus.

The final step of the proof is then to negate $\mathit{violate}_G$. Theorem 24 follows directly.