Department of Mathematics and Computer Science
Formal System Analysis

# Fairness Assumptions in the Modal $\mu$-Calculus

*Master Thesis*

Myrthe Spronck

*Supervisor:*
Tim Willemse

September 2023

## Abstract

The modal $\mu$-calculus is a highly expressive logic, but its formulae are often hard to understand. We have tools for testing if a model satisfies a model $\mu$-calculus formula, but if we are unsure of what the formula expresses we cannot draw definite conclusions from the results. To mitigate the difficulties in designing $\mu$-calculus formulae, property specification patterns have been designed to help researchers express common properties in the $\mu$-calculus. However, existing translations of these patterns to the modal $\mu$-calculus only allow for the inclusion fairness assumptions to a very limited degree, even though fairness assumptions are very useful when model checking. Fairness assumptions allow the researcher to specify that certain types of property violations that may exist in the model are unrealistic and should not be considered when determining if the property is satisfied. This is often required because most of the time, when researchers model systems they abstract away from certain details such as the way scheduling is done, which then results in scenarios being represented in the model that would never occur in reality. There exists, therefore, a need for a standard and proven way of including a variety of fairness assumptions in modal $\mu$-calculus formulae for properties. We extend the existing translations from the property specification patterns to the modal $\mu$-calculus with ways to incorporate three common fairness assumptions: weak fairness, strong fairness and fair reachability (also known as $\infty$-fairness or hyperfairness) of the actions in a model. We also discuss other fairness assumptions to a lesser extend, including unconditional fairness of actions and weak fairness, strong fairness and fair reachability of parts of a model other than actions. When it comes to the patterns, we start with a detailed discussion on the global response pattern, which is one of the most commonly occurring ones. We then generalise our approach to cover other patterns as well. Correctness proofs are included for both the global response formulae and the generalised formulae. We conclude we a brief discussion of how the formulae we have presented in this thesis can be used in the model checking toolset mCRL2.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

Model checking is a powerful technique for verifying the correctness of protocols, software, and systems. To do model checking, a model must be created using some modelling language. Additionally, the properties that should be satisfied by the model must be expressed as formulae in some logic. Automatic tools can then check if the formulae are satisfied by the model. In this thesis, the part of this process we are interested in is the expressing of properties. One logic that is used for this purpose is the modal $\mu$-calculus [29] (also referred to as just the $\mu$-calculus). The $\mu$-calculus is used in the model checking toolsets mCRL2 [11] and CADP [21], among others. In particular due to the inclusion of fixpoint operators, the modal $\mu$-calculus is highly expressive, subsuming many other frequently used logics such as Linear Temporal Logic (LTL) and Computation Tree Logic (CTL). The price paid for the expressiveness of the $\mu$-calculus is that its formulae can become difficult to understand and design. To quote Bradfield and Stirling [8]:

> "Fixpoint logics [such as the modal $\mu$-calculus] are notorious for being incomprehensible. Indeed, it has been known for several reasonably expert people to spend an hour or two trying to work out whether a one line formula means what it was intended to mean."

A model checking tool might report that a model satisfies a given formula, but if the researcher cannot be certain that the formula truly expresses the correct property then no convincing conclusions can be drawn. In [41] it is shown that even in published studies formulae crop up that do not express what the author claims they express. Even if the researcher is confident the formula is correct, they will also need to convince their peers. This gives rise to a desire to make it easier for researchers to design correct $\mu$-calculus formulae. A formula designed using a reliable technique will be more convincing all parties.

This goal is very general, and no single solution will be able to fully address it. We must therefore identify a smaller sub-problem, a subset of formulae that researchers want to use but which are hard to design and for which it is difficult to be sure they truly express what they should. Inspired by hurdles encountered in our own research while working on [44], as well as discussions with others in the field of model checking, we settle on the following sub-problem: the inclusion of fairness assumptions in formulae representing common properties. Fairness assumptions are assumptions on what executions of a model represent real executions of the modelled system. Specifically, they limit which *infinite* executions of the model are considered valid. Under a fairness assumption, a property need only be satisfied in those valid executions. Violations in invalid executions are disregarded. It is rather common for models to contain unrealistic infinite executions because when modelling it is often required to abstract away from many of the details of the true system. For example, a button that may be pressed at most one thousand times before the system accepts no further input may simply be modelled as a button that can be pressed infinitely often. Or a system that includes a scheduler that enforces that each internal process is allowed to progress equally often may be modelled without that scheduler, so that one process may act repeatedly without the others getting a chance to progress. Such choices simplify the model and make verification easier, but they can introduce executions that are not present in the real-world system. Fairness assumptions are a solution to this problem.

Different fairness assumptions are distinguished by which executions they deem to be invalid. It is important to make the right fairness assumptions when doing verification. If the assumption labels too many executions as invalid, then there is a risk that realistic violations of the property are ignored. But if the assumption invalidates too few executions, then it might be decided a property is unsatisfied, even though the violations occur on paths that are unrealistic. Hence, there is not one perfect fairness assumption, and work focused on the ways to use fairness assumptions in model checking should ideally cover multiple assumptions.

Fairness assumptions may be incorporated into a model checking procedure in a number of different ways. For example, the tool itself could have options for restricting the infinite paths that are considered. Consider the NuSMV model checker [12], which allows users to specify fairness constraints as part of the model and then accounts for those when doing LTL and CTL model checking. One can also use existing features in the model checker to ignore some executions. In [3], clever hiding of actions is used to ensure unrealistic executions can be ignored using other tools that are already present in the CADP toolset. A more generic solution is to incorporate the fairness assumption directly into the property that is being checked. This approach allows for great flexibility: the tool itself does not need to be changed and the designer of the formula can incorporate any fairness assumption they wish, independent of what the tool's designers have considered.

We therefore make this our chosen sub-problem of the larger issue of making the design of modal $\mu$-calculus formulae more reliable and accessible: how to incorporate a variety of fairness assumptions into modal $\mu$-calculus formulae. This requires an exploration of which fairness assumptions exist in the literature, which fairness assumptions make sense to make in the context of an event-based logic like the $\mu$-calculus, and how these fairness assumptions can be integrated into a formula. Additionally the design of a formula depends on what property the formula is supposed to express. Hence, our work requires us to have some idea of what sorts of properties we need to design formulae for. For this, we look to the property specification patterns presented in [15], which we introduce in the related work section and explore in detail in later chapters.

In this thesis, we present formulae that cover many of the most commonly occurring properties in model checking, under some of the most commonly discussed fairness assumptions in the literature. We include correctness proofs for our formulae, so that those using them can be assured of precisely what they express.

## 1.2 Related Work

We are not the first to consider that formulae, in both the modal $\mu$-calculus and other logics, can be hard to design. In 1999, Dwyer, Avrunin and Corbett considered the complexity of designing the right formula for a given property in an arbitrary logic and aimed to solve this through presenting design patterns for properties [15]. These *property specification patterns* (PSP) each represent a particular behaviour one may wish to specify, and all allow a variety of scopes during which that behaviour must hold. The patterns were originally given with LTL, CTL and Quantified Regular Expressions, but have since been extended with other logics. Modal $\mu$-calculus formulae for these patters have been designed by Radu Mateescu for the CADP toolset. Daniela Remenska later extended Mateescu's formulae with variant interpretations of some of the scopes, as well as formulae under a form of fairness close to what we call fair reachability of actions in this thesis [41]. We include a detailed presentation of the PSP patterns in Chapter 5.

To our knowledge, Remenska's formulae are some of the only existing guidelines for how to incorporate fairness assumptions in modal $\mu$-calculus formulae. Her formulae are part of the Property ASSistant (PASS) tool, which allows users to specify their properties using a questionnaire and then gives them the resulting $\mu$-calculus formula. Her work is one of the main reasons we chose to use PSP as the property structures to design formulae for. The other reason for using PSP as a basis is that surveys of properties used in the literature have shown a vast majority of properties are covered by the property specification patterns [15, 41].

The property specification patterns of Dwyer et al. appear to be the most popular pattern system for specifying properties in the literature. That does not mean no other pattern systems exist, although most we encounter in the literature are inspired by or extend PSP. For example, in [5] an survey is given of property specification patterns for real-time systems, including extensions of the patterns from [15]. Another example is [26], in which PSP is extended to also cover probabilistic properties. Many of these extensions of PSP are not relevant for us, since we use labelled transition systems without time or probability as our models.

When researching different fairness assumptions, [22] has been a phenomenal resource. In this paper, Höfner and Van Glabbeek present an overview of many of the fairness assumptions that appear in the literature. They also discuss the much weaker progress assumptions and introduce a new assumption: *justness*. We recap many of their definitions in Chapter 3 and use their taxonomy when discussion fairness assumptions. Another great overview of different fairness assumptions is [19], a book going into much detail on, in particular, strong, weak and unconditional fairness. A survey of the concept of fairness and the many perspectives that exist on it is given in [30]. Further bibliographic notes on fairness assumptions that appear in the literature are presented in Chapter 3.

In [7], Bouwman, Luttik and Willemse present a modal $\mu$-calculus formula for a property that matches the global response pattern for events under the justness assumption from [22]. Their formula forms the basis for many of the formulae we present in this thesis. They also present a proof of correctness for their formula, which was an inspiration for many of the proofs presented in this thesis as well.

## 1.3 Research Questions

We break up our problem into several research questions. We here present these questions, and give a brief preview of how we answer them.

First, we consider which fairness assumptions exist and which are the most interesting for us to discuss.

> **RQ1**: *Which fairness assumptions exist in the literature, and of those which are the most interesting and relevant for us to cover?*

Based on our literature study, we determine weak fairness, strong fairness and unconditional fairness are the most fundamental and commonly discussed fairness assumptions, at least for event-based logics such as the modal $\mu$-calculus is. Unconditional fairness has the undesirable property that it is infeasible, which leads to us covering it in less detail than the other two. In addition to these, we decide to cover the fair reachability assumption, since this appears to be a fairness assumption for which some modal $\mu$-calculus formulae have already been designed.

When it comes to what part of the model is treated fairly, we primarily look at the actions that the system takes. However, we also discuss a method for generalising what parts of the model are treated fairly.

The next step of our research is to choose a specific pattern from the property specification patterns and provide formulae for this pattern in our chosen fairness assumptions. We choose to start with the global response pattern, which says that whenever some action $q$ occurs, it must eventually by followed by an action $r$. This is one of the most commonly occurring patterns.

> **RQ2**: *How can our chosen fairness assumptions be integrated into modal $\mu$-calculus formulae following the global response pattern?*

We present formulae for global response under weak fairness of actions, strong fairness of actions, fair reachability of actions and unconditional fairness of actions. For the first three, we also provide correctness proofs. While it is correct, the strong fairness formula is less than ideal due to its complexity being exponential in the number of actions in the model. In later chapters we extend the formulae for global response to fairness over other aspects of the model than actions. We also present some ways the computational cost of the strong fairness formula can be mitigated, although we do not manage to fully resolve it.

Once we have designed formulae for global response, we use the insights gained to represent the other patterns from [15] as well.

> **RQ3**: *How can our chosen fairness assumptions be integrated into modal $\mu$-calculus formulae following the property specification patterns?*

To answer this question, we first observe that for many of the patterns, fairness is actually irrelevant: whether a property matching that pattern is satisfied in a model is not affect by whether a fairness assumption is made. At least, this is the case when the fairness assumption in question is feasible. For the main patterns that are left, we present formulae under weak fairness, strong fairness and fair reachability. Unconditional fairness is dropped at this point due to being infeasible. Rather than presenting many separate formulae, we present pattern-agnostic "base" formulae that include several placeholder variables. We then cover the different patterns by showing how the placeholders should be filled in. The base formulae are proven correct.

Finally, we consider how the formulae we present may be used in practice. For this, we look at the model checking toolset mCRL2, with which we have experience and which has some useful properties that we expand on in a later chapter.

> **RQ4**: *What is required for our designed formulae to be used in the model checking tool mCRL2?*

We provide a detailed explanation of what features a model should have for our formulae to be usable, and how the formulae themselves can be represented within mCRL2. We also include a small case study, showing these steps in practice and illustrating the respective efficiency of our formulae.

## 1.4 Content

This thesis starts with a preliminaries chapter, Chapter 2. Here, we present labelled transition systems, which are what we use to represent models. We also present most of the definitions on states, transitions and paths that we will use for the remainder of this thesis. In Section 2.2, we present the syntax and semantics of the modal $\mu$-calculus. In the final part of the preliminaries chapter, we introduce the tools we use throughout this thesis: mCRL2 and MLSolver.

While the presentation of the fairness assumptions could be considered part of the preliminaries, we present them separately in Chapter 3. We do this because this section is quite extensive, and does feature some original work, particularly on fair reachability. We explain the general structure of a fairness assumption, and then formally define weak fairness, strong fairness, unconditional fairness and fair reachability in Section 3.1. We express what parts of the model are being treated fairly through tasks, following the definitions from [22]. How these tasks can be chosen is explored in Section 3.2, with examples. The word feasibility was already mentioned, without explanation, in Section 1.3. This concept is explained in Section 3.3 and we prove (or cite existing proofs) that weak fairness, strong fairness and fair reachability are feasible, whereas unconditional fairness is not. In Section 1.2 we mentioned bibliographic notes on a variety of fairness assumptions, some of these are given throughout the chapter but most are in Section 3.4, where we mention some of the fairness assumptions from the literature we chose not to cover. We conclude the chapter by arguing why we chose to cover the fairness assumptions that we have. At this point, we have answered most of RQ1.

We then move on to answering RQ2 in Chapter 4. We introduce the global response pattern here in detail, since we have not yet fully explored all the property specification patterns. We also discuss some different approaches for incorporating fairness into a formula we have observed in the literature. We subsequently provide formulae for global response under weak fairness of actions (Section 4.3), fair reachability of actions (Section 4.4), unconditional fairness of actions (Section 4.5) and strong fairness of actions (Section 4.6). For the unconditional fairness formula an informal correctness argument is provided, the other three are properly proven.

We next aim to answer RQ3, but first provide an overview of the property specification patterns from [15], with Remenska's extensions [41] and a few extensions of our own, in Chapter 5. With this background, we then extend the formulae for global fairness to cover more patterns in Chapter 6. This chapter opens with a discussion of why fairness is irrelevant for some of the patterns. This can be seen as the final part of the answer to RQ1, since this explains why we disregard unconditional fairness for much of this thesis. After this, the pattern-agnostic base versions of the formulae are presented in Section 6.2. The table containing the ways the placeholder variables should be filled in is also included in this section. The correctness claims for these formulae are given in Section 6.3, although the proofs of these claims are postponed to Appendix B due to their length. Finally, we justify the contents of the table by discussing the ways the formulae are filled in for different patterns in more detail in Section 6.4.

Up until this point, we have only covered a form of fairness where all actions in the model must be treated fairly. In Chapter 7, we provide two generalisations of the fairness assumptions: we present formulae stating that only some actions are treated fairly; and we present formulae for different choices of tasks. These are Section 7.1 and Section 7.2 respectively. With these added types of fairness, we have properly answered RQ2 and RQ3.

Finally, we answer RQ4 in Chapter 8. We explain how an mCRL2 model must be modified to work with our formulae, and how the formulae themselves can be represented in the mCRL2 toolset. These explanations are given as a guide to others that may wish to use the formulae we present in their own research projects. Finally, we give a case study of analysing starvation freedom under fairness of actions and fairness of components in Dekker's mutual exclusion algorithm. The main observation of interest here is that strong fairness of actions is extremely slow, but strong fairness of components takes hardly any time to compute at all. This demonstrates one way to mitigate the complexity of the strong fairness formula. All mCRL2 files given as examples in this chapter, as well as the formulae and batch script from Appendix H, can be found on Github at `https://github.com/MSpronck/FairnessInMucalc`[1].

We end this thesis with our conclusions in Chapter 9, including ideas for future work in Section 9.2.

[1]This page has also been archived at `http://web.archive.org/web/20230831220858/https://github.com/MSpronck/FairnessInMucalc`. Throughout this thesis, we frequently include links to archived pages to reduce the risk of link rot, since we encountered many broken links while working on this project. Archived Github pages seem to be incorrect in some cases, not properly including all information. For the repository belonging to this thesis, the download link is archived at `http://web.archive.org/web/20230831224931/https://codeload.github.com/MSpronck/FairnessInMucalc/zip/refs/heads/main`. For all other Github pages, we link to archived versions of the raw files.

# Chapter 2

# Preliminaries

In this chapter, we present many of the basic definitions and concepts that are used throughout the rest of this thesis. We first introduce labelled transition systems in Section 2.1, then the modal $\mu$-calculus in Section 2.2. Finally, in Section 2.3, we cover the tools we use throughout this thesis. The background information on fairness assumptions is postponed to Chapter 3, since we present the preliminaries on fairness together with some new observations. For similar reasons, the background information on the property specification patterns is reserved for Chapter 5.

## 2.1 Labelled Transition Systems

We will first define the notion of a labelled transition system (LTS) which we use to define system models and the semantics of the modal $\mu$-calculus over those models. Labelled transition systems appear in many forms in the literature, we present one definition here.

**Definition 2.1** (Labelled transition systems)**.** Let $M = (\mathcal{S}, s_{init}, Act, Trans)$ be an LTS, where

- $\mathcal{S}$ is a set of states.
- $s_{init} \in \mathcal{S}$ is an initial state.
- $Act$ is a set of action labels. This set is also referred to as the alphabet of the LTS.
- $Trans \subseteq \mathcal{S} \times Act \times \mathcal{S}$ is a transition relation. We write $s \xrightarrow{a} t$ as shorthand for $(s, a, t) \in Trans$. For a transition $tr = (s, a, t)$ we have that $source(tr) = s$, $target(tr) = t$ and $action(tr) = a$.

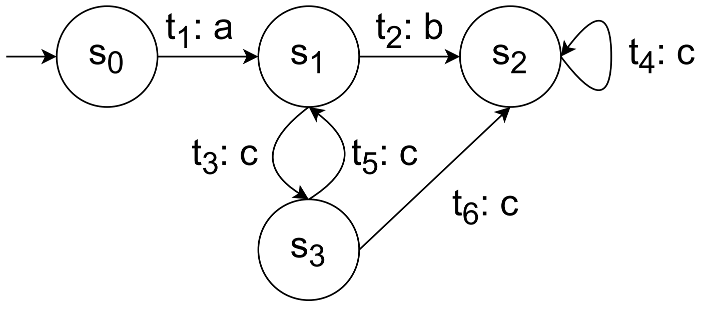


Figure 2.1: A graphical representation of the LTS described in Example 2.2.

In this thesis, we consider only finite labelled transition systems. This means systems with a finite number of states and a finite number of transitions. A consequence of only considering finite sets of transitions, we are also restricted to finite sets of actions. All correctness proofs in this thesis assume a finite system. When a part of a proof relies on this assumption, it is noted. Model checking is generally done on finite systems, hence this is not a strange assumption to make given the topic of this thesis.

**Example 2.2.** As an example of an LTS, we show a graphical representation of an LTS with $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$, $s_{init} = s_0$, $Act = \{a, b, c\}$, and $Trans = \{t_1, t_2, t_3, t_4, t_5, t_6\}$ where $t_1 = (s_0, a, s_1)$, $t_2 = (s_1, b, s_2)$, $t_3 = (s_1, c, s_3)$, $t_4 = (s_2, c, s_2)$, $t_5 = (s_3, c, s_1)$ and $t_6 = (s_3, c, s_2)$. The graphical representation is shown in Figure 2.1.

We give a number of definitions on LTSs, which will be used in subsequent chapters. Many of these come from [22]. These definitions are given on an LTS $M$ with $\mathcal{S} = \{s_0, s_1, \ldots\}$, $s_{init} = s_0$, $Act = \{a_1, a_2, \ldots\}$ and $Trans = \{t_1, t_2, \ldots\}$.

**Definition 2.3** (Path)**.** A path is an alternating sequence $\pi = s_0 t_1 s_1 t_2 ..$ of states and transitions, starting with a state and either being infinite or ending with a state. For all $i \geq 0$, $s_i t_{i+1} s_{i+1}$ can only be included in a path when $t_{i+1}$ is a transition from state $s_i$ to state $s_{i+1}$ in $M$.

Unless stated otherwise, we only consider paths that start in the initial state of the model. When we refer to the length of a path, we mean the number of transitions on that path. Hence, a path consisting of only a single state is a path of length zero. This path is sometimes called the empty path, even though it does contain a single state.

When discussing paths, we often use the term "step" to refer to mean the execution of some transition $t_k$ to go from $s_{k-1}$ to $s_k$. We sometimes use "$a$-step" to refer to an arbitrary transition that is labelled with the action $a$. Additionally, we often refer to a specific state or transition on a path, particularly in proofs. While a state or transition from the model can occur multiple times on the same path, when we write about a state $s_k$ or a transition $t_k$ for some $k$, we mean a specific *occurrence* of that state or transition on the path. A final note on our notation regarding paths in proofs: we often construct paths by appending a path $\pi'$ that starts in a state $s$ to a path $\pi$ that ends in that same state $s$. We then write $\pi\pi'$. This is a slight abuse of notation, since the state $s$ appears both at the end of the sequence $\pi$ and at the start of the sequence $\pi'$, but in the constructed path it cannot appear twice in a row. It should be understood that when we write $\pi\pi'$, the $s$ at the end of $\pi$ overlaps with the $s$ at the start of $\pi'$, so that there is only a single $s$ at that spot in the constructed sequence of states and transitions.

We proceed with some definitions of many of the properties a state, transition or path can have.

**Definition 2.4** (Enabledness of transitions)**.** A transition $t$ is *enabled* in a state $s$ if $source(t) = s$.

**Definition 2.5** (Perpetual enabledness)**.** A transition $t$ is *perpetually enabled* on a path $\pi$ if $t$ is enabled in every state of $\pi$.

**Definition 2.6** (Relentless enabledness)**.** A transition $t$ is *relentlessly enabled* on a path $\pi$ if each suffix of $\pi$ contains a state in which it is enabled.

**Definition 2.7** (Deadlock state)**.** A state $s$ is a *deadlock state* when there are no transitions enabled in $s$.

**Definition 2.8** (Complete and partial paths)**.** We say a path $\pi$ is *complete* if it is either infinite or ends in a deadlock state. If a path is not complete, it is *partial.*

If we speak of a path without specifying if it is complete or partial, it may be either. However, we primarily discuss complete paths in this thesis, because most of our arguments assume progress. We expand on this further in Section 2.1.1.

**Definition 2.9** (Reachable state)**.** A state $s$ is *reachable* from a state $s'$ if there exists a path starting in $s'$ which ends in $s$.

**Definition 2.10** (Reachable transition)**.** A transition $t$ is *reachable* from a state $s$ if there exists a path starting in $s$ which ends in a state $s'$ such that $source(t) = s'$.

The notions of relentless reachability and perpetual reachability of both states and transitions are defined analogously to relentless and perpetual enabledness of transitions. It is interesting to note that relentless and perpetual reachability of transitions are equivalent when considering complete paths, unlike relentless and perpetual enabledness of transitions. We prove both those claims here.

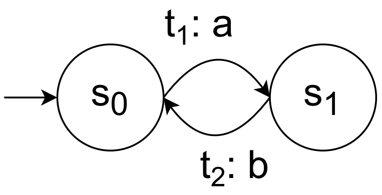


Figure 2.2: A counterexample showing that relentless enabledness of a transition does not imply perpetual enabledness.

**Lemma 2.11.** *On a complete path $\pi$, if a transition $t$ is perpetually enabled it is also relentlessly enabled.*

*Proof.* This follows directly from the definitions: if $t$ is perpetually enabled on $\pi$ then it is enabled in every state of $\pi$. Hence, every suffix of $\pi$ contains only states in which $t$ is enabled. From this we conclude that every suffix of $\pi$ contains at least one state in which $t$ is enabled, and $t$ is therefore relentlessly enabled on $\pi$. □

**Lemma 2.12.** *On a complete path $\pi$, a transition $t$ being relentlessly enabled does not imply that it is also perpetually enabled.*

*Proof.* This can be shown with a simple counterexample. See Figure 2.2. Consider the only complete path on this LTS: the infinite path $(s_0 t_1 s_1 t_2)^\omega$. On this path, both $t_1$ and $t_2$ are enabled relentlessly, but neither is enabled perpetually. □

**Lemma 2.13.** *On a complete path $\pi$, a transition $t$ is perpetually reachable if, and only if, it is relentlessly reachable.*

*Proof.* This proof has two directions. Firstly, that perpetual reachability implies relentless reachability. The argument here is very similar to that of Lemma 2.11: if $t$ is perpetually reachable on $\pi$ then it is reachable from every state of $\pi$. Hence every suffix of $\pi$ contains a state in which $t$ is reachable, and so $t$ is relentlessly reachable.

The other direction is that relentless reachability implies perpetual reachability. We assume that every suffix of $\pi$ contains a state from which $t$ is reachable. We will prove that $t$ is reachable from every state of $\pi$. Let $s$ be an arbitrary state on $\pi$. Let $\pi'$ be the suffix of $\pi$ starting at $s$. There must exist a state $s'$ on $\pi'$ from which $t$ is reachable, hence there exists a path $\pi_{s'}$ starting in $s'$ and ending in a state on which $t$ is enabled. Since $s'$ is on $\pi'$ and $s$ is the first state of $\pi'$, we know $s'$ comes later in $\pi$ than $s$ or $s = s'$. Let $\pi_s$ be the path starting in $s$ and ending in $s'$ (the empty path if $s = s'$). Then $\pi_s \pi_{s'}$ is a path starting in $s$, ending in a state on which $t$ is enabled. Hence, $t$ is reachable from $s$. Since $s$ is an arbitrary state on $\pi$, we have proven $t$ is reachable from every state of $\pi$ and hence $t$ is perpetually reachable on $\pi$. □

In much of this thesis, we abstract away from specific transitions and instead refer solely to actions. Each of the definitions on transitions we have presented can also be given for actions. In short: an action $a$ is enabled in a state $s$ if there is a transition $t$ enabled in $s$ that is labelled with $a$. An action $a$ is reachable from a state $s$ if a transition $t$ is reachable from $s$ that is labelled with $a$. Relentless and perpetual enabledness and reachability are also defined for actions, analogously to how they are defined for transitions.

We also give the following definition, which is specific to discussions on the occurrence of actions in paths:

**Definition 2.14** ($a$-free paths)**.** A path $\pi$ is $a$-free for an action $a$ if no transition labelled with $a$ occurs in $\pi$.

### 2.1.1 Assuming Progress

In the introduction, we stated that fairness assumptions limit which infinite executions (paths) of a model are considered valid. Progress assumptions are a strongly related concept: these are assumptions that limit which *finite* paths are considered valid. Progress assumptions are much less controversial than fairness assumptions, and are even frequently made implicitly [22].

A progress assumption boils down to assuming that the system underlying a model will not stop executing arbitrarily. Under what conditions we consider it reasonable for an execution to terminate depends on the system. The default progress assumption is the following: as long as it is possible for the system to perform an action, it will eventually do so. Applied to paths, this means a path under the assumption of progress cannot end in a state where transitions are still enabled.

**Definition 2.15** (Progress)**.** A path $\pi$ is *progressing* if it is either infinite or ends in a deadlock state. The *progress* assumption states all paths are progressing. Hence, if we assume progress only progressing paths are considered valid.

Note that the definition of a progressing path coincides with the definition of a complete path from Definition 2.8. Hence, under this progress assumption only complete paths are considered valid. This is the progress assumption we use in this thesis. Unless stated otherwise, we make this assumption for all subsequent discussions and arguments.

Assuming Definition 2.15 is often justified. If we are modelling a system that does not require outside stimuli to perform actions there is little reason why it would stop performing actions while it is still capable of making progress. However, there are some contexts where this assumption is too strong because the system does rely on interactions with an environment. As an example, consider a model representing a coffee machine that has an action *order_made*, which represents a user requesting a coffee. This coffee machine interacts with its environment, the users, and the *order_made* action represents an interaction with that environment. We may want to avoid assuming that there will always be a next customer when we are doing verification. This gives rise to the concept of *blocking* actions: actions that require participation of the environment, and hence should not be assumed to always occur eventually. If we use the progress assumption from Definition 2.15, we would be assuming that a path is not valid if it ends in a state in which *order_made* is enabled. This would mean assuming there will always be a next customer. In this case, this progress assumption would be too strong. An alternative progress assumption, taking blocking actions into account, is presented in [22]. The starvation freedom under justness $\mu$-calculus formula presented in [7] takes blocking actions into account.

Designing modal $\mu$-calculus formulae that take different progress assumptions into account is an interesting topic. However, in this thesis we focus specifically on fairness and so we do not consider blocking actions and progress assumptions, other than Definition 2.15, any further here.

## 2.2 Modal $\mu$-Calculus

There are many introductions to the modal $\mu$-calculus available. For the overview given here we base ourselves largely on [8, 9, 10] and [24, Chapter 6].

### 2.2.1 Syntax

The syntax of the modal $\mu$-calculus is described by the following grammar, which is parameterised with the set of action labels *Act* as well as a set of formal variables *Var*:

$$\phi, \psi ::= \mathit{tt} \mid \mathit{ff} \mid X \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \Rightarrow \psi \mid [a]\phi \mid \langle a \rangle \phi \mid \mu X.\phi \mid \nu X.\phi$$

Where *tt* and *ff* are the Booleans true and false, $X$ is taken from the set *Var* and $a$ from the set of action labels *Act*. The $\langle\ \rangle$ is referred to as the diamond operator, $[\ ]$ as the box operator. The $\mu$ is called the least fixpoint operator, $\nu$ the greatest fixpoint operator[1].

The following order of precedence is used: $[\ ]$ and $\langle\ \rangle$ bind the strongest, followed by $\neg$, then $\wedge$ and $\vee$, then $\Rightarrow$, and finally $\mu$ and $\nu$. For example $\nu X.\neg[a]X \wedge$

[1]The terms "fixpoint" and "fixed point" are frequently used interchangeably. We follow the suggestion made in [8] of speaking of "fixpoint operators" which give us the "fixed points" of functions.

$\phi \Rightarrow \langle a \rangle tt$ is the same as $\nu X.((\neg([a]X) \wedge \phi) \Rightarrow \langle a \rangle tt)$. We will rarely rely on this order of precedence, instead we will use brackets whenever there may be any ambiguity.

There is an additional requirement on $\mu$-calculus formulae: they must have syntactic monotonicity. A formula $\phi$ is *syntactically monotonic* if, and only if, for every fixed point $\mu X.\psi$ or $\nu X.\psi$ in $\phi$, $X$ occurs positively in $\psi$. This means that every occurrence of the variable $X$ in $\psi$ must be preceded by an even number of negations. For the purpose of counting negations, $\phi \Rightarrow \psi$ should be read as $\neg\phi \vee \psi$. We impose the requirement of syntactic monotonicity to guarantee fixpoints are meaningful: syntactic monotonicity is sufficient for ensuring that the least and greatest fixed points of a formula $\phi$ evaluated on a model $M$ actually exist and can be computed when calculating the semantics of $(\mu/\nu)X.\phi$ on $M$. For the reasons why this is the case, we refer to the cited sources. The underlying theory is outside the scope of this thesis.

It should be noted that there are other variants of the modal $\mu$-calculus. In particular, some presentations include atomic propositions which allow for state information to be referenced. We do not account for this here, since the forms of fairness we consider are restricted to transitions being taken fairly rather than states being visited fairly.

### 2.2.2 Semantics

Before we give the formal semantics of the modal $\mu$-calculus, we present an intuitive explanation. Given a model $M = (\mathcal{S}, s_{init}, Act, Trans)$, the semantics of formula $\phi$ are the subset of $\mathcal{S}$ on which that formula evaluates to true. Boolean $tt$ is true in every state, $ff$ in no state. A formal variable describes a set of states, and so is true in every state that is in that set. The formal variables can be bound by $\mu$ or $\nu$, or they can be free in which case an environment, mapping each free variable to a set of states, must be given before the formula can be evaluated. The connectives from propositional logic behave as would be expected. The box and diamond operators, $\langle\ \rangle$ and $[\ ]$, represent the possibility and necessity modalities respectively. The formula $\langle a \rangle \phi$ can be read as "it is possible to do an $a$-action such that in the next state $\phi$ holds". Dually, $[a]\phi$ can be read as "after every $a$-action, in the next state $\phi$ necessarily holds". The fixpoint operators, $\mu$ and $\nu$, are more difficult to understand intuitively, this is part of the reason why $\mu$-calculus formulae can be hard to interpret.

Fixpoints operators can be understood as a way to add recursion to a formula. This is useful because the $\mu$-calculus is a state-based logic, and many of the properties we want to express are on paths. Using recursion, we can reason about paths. The difference between the least and greatest fixpoint operators is how far the recursion may be unfolded. The least fixed point, indicated by $\mu$, represents finite unfolding. It is often used to indicate that something must eventually happen: if something eventually happens, then you can only take finitely many steps where it does not happen. Therefore, $\mu$ is often used in liveness properties ("something good eventually happens"). On the other hand, the greatest fixed point, $\nu$, allows for infinite unfolding. It is often used to express that something must hold invariantly: if something is invariantly true, then even if you take infinitely many steps it must remain true. The greatest fixpoint operator is often used for safety properties ("something bad never happens"). The slogan "$\nu$ means looping, $\mu$ means finite looping" [9] can give us some intuitive insight into formulae, but when fixed points are nested it is often not enough. When least and greatest fixpoint operators alternate, it becomes particularly hard to grasp the meaning of a formula.

The formal semantics of a modal $\mu$-calculus formula is the set of states of a given LTS in which it is satisfied. Hence, to calculate the semantics of a formula $\phi$, we also need to give an LTS $M = (\mathcal{S}, s_{init}, Act, Trans)$. Additionally, as previously suggested, if there are any free formal variables in $\phi$ we also need an environment $\epsilon$ which maps from $Var$ to subsets of $\mathcal{S}$. We give the semantics $[\![\phi]\!]^M_\epsilon$ as follows:

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

Where $\epsilon[X := \mathcal{S}']$ represents an environment where $X$ is interpreted as $\mathcal{S}'$, and all other variables are interpreted as they are in $\epsilon$.

We say that a state $s$ of $M$ satisfies formula $\phi$, written as $s \vDash \phi$, if, and only if, $s \in [\![\phi]\!]^M_{\epsilon_0}$ where $\epsilon_0$ gives the initial mapping for all free variables. An LTS satisfies $\phi$ if, and only if, its initial state satisfies $\phi$. If the model $M$ is fixed, we often leave it out and just write $[\![\phi]\!]_\epsilon$.

There are alternative ways to give the semantics of $\mu$-calculus formulae, for example the semantics can be given in terms of games as shown in [10]. We restrict ourselves to the semantics presented above.

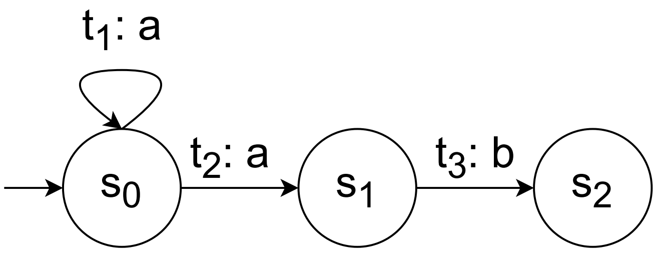


Figure 2.3: The LTS for Example 2.16.

We illustrate the given semantics with a few example formulae, specifically showing the different uses of least fixpoint, greatest fixpoint, box and diamond operators. For the following example, we take $M$ to be the LTS shown in Figure 2.3 and we take an arbitrary environment $\epsilon$, since there will be no free variables in our example formulae.

**Example 2.16.** We stated that least fixpoints are often used for liveness properties and greatest fixpoints for safety properties. We illustrate that here. Take for instance the liveness property "there exists a sequence of $a$-actions which leads to a state where $b$ is enabled". In the modal $\mu$-calculus this could be expressed as $\phi_1 = \mu Y.(\langle a\rangle Y \vee \langle b\rangle tt)$[2]. This formula holds in states where $b$ is enabled, or which can reach a state where $b$ is enabled in finitely many $a$-steps. Consider the LTS in Figure 2.3, this formula is satisfied in the initial state of that LTS because it is possible to reach $s_1$, in which $b$ is enabled, by doing the transition $t_2$ from $s_0$, which is labelled with an $a$. We can calculate the semantics to confirm this. We do not show every intermediate step of the calculation, to save space.

$$\begin{aligned}
[\![\phi_1]\!]^M_\epsilon &= \bigcap\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq [\![\langle a\rangle Y \vee \langle b\rangle tt]\!]^M_{\epsilon[Y:=\mathcal{S}']}\}\\
&= \bigcap\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq [\![\langle a\rangle Y]\!]^M_{\epsilon[Y:=\mathcal{S}']} \cup [\![\langle b\rangle tt]\!]^M_{\epsilon[Y:=\mathcal{S}']}\}\\
&= \bigcap\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq [\![\langle a\rangle Y]\!]^M_{\epsilon[Y:=\mathcal{S}']} \cup \{s \in \mathcal{S} \mid \exists_{s'\in\mathcal{S}}.s \xrightarrow{b} s' \wedge s' \in \mathcal{S}\}\}\\
&= \bigcap\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq \{s \in \mathcal{S} \mid \exists_{s'\in\mathcal{S}}.s \xrightarrow{a} s' \wedge s' \in \mathcal{S}'\} \cup \{s_1\}\}\\
&= \{s_0, s_1\} \cap \{s_0, s_1, s_2\}\\
&= \{s_0, s_1\}
\end{aligned}$$

For the second to last step, this state space is small enough that we can consider every possible $\mathcal{S}' \subseteq \mathcal{S}$ and manually check if they satisfy $\mathcal{S}' \supseteq \{s \in \mathcal{S} \mid \exists_{s'\in\mathcal{S}}.s \xrightarrow{a} s' \wedge s' \in \mathcal{S}'\} \cup \{s_1\}\}$. We find only $\{s_0, s_1\}$ and $\{s_0, s_1, s_2\}$ meet this requirement.

[2]Where possible, we prefer to use $X$ and $Z$ as the formal variables for greatest fixpoints and $Y$ and $W$ for least fixpoints. This is purely a stylistic preference.

Replacing the first diamond operator with a box results in $\phi_2 = \mu T.([a]Y \vee \langle b\rangle tt)$. This states all sequences of $a$-steps lead to a state where $b$ is enabled in finitely many transitions. This formula is not satisfied by $s_0$, since the transition $t_1$ can be taken infinitely often from $s_0$ without every reaching a state where $b$ is enabled. We once again compute the semantics:

$$\begin{aligned}
[\![\phi_2]\!]^M_\epsilon &= \bigcap\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq [\![[a]Y \vee \langle b\rangle tt]\!]^M_{\epsilon[Y:=\mathcal{S}']}\} \\
&= \bigcap\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq [\![[a]Y]\!]^M_{\epsilon[Y:=\mathcal{S}']} \cup \{s_1\}\} \\
&= \bigcap\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq \{s \in \mathcal{S} \mid \forall_{s' \in \mathcal{S}}. s \xrightarrow{a} s' \Rightarrow s' \in \mathcal{S}'\} \cup \{s_1\}\} \\
&= \{s_1, s_2\} \cap \{s_0, s_1, s_2\} \\
&= \{s_1, s_2\}
\end{aligned}$$

Note that $s_1$ and $s_2$ satisfy this formula because they do not have any $a$-transitions, which means that any condition on *all* sequences of $a$-steps is vacuously satisfied.

Greatest fixpoints are more suited for safety properties, such as "along all sequences of $a$-actions, $b$ is never enabled". This property can be expressed as $\phi_3 = \nu X.([a]X \wedge [b]f\!f)$. We use $[b]f\!f$ to represent states that do not admit $b$-transitions, since that is the only way for a state to satisfy the condition that all $b$-transitions lead to states in the empty set. Of course, this formula is not satisfied in $s_0$, since after taking $t_2$ we are in a state where $b$ is enabled.

$$\begin{aligned}
[\![\phi_3]\!]^M_\epsilon &= \bigcup\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \subseteq [\![[a]X \wedge [b]f\!f]\!]^M_{\epsilon[X:=\mathcal{S}']}\} \\
&= \bigcup\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \subseteq [\![[a]X]\!]^M_{\epsilon[X:=\mathcal{S}']} \cap [\![[b]f\!f]\!]^M_{\epsilon[X:=\mathcal{S}']}\} \\
&= \bigcup\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \subseteq [\![[a]X]\!]^M_{\epsilon[X:=\mathcal{S}']} \cap \{s \in \mathcal{S} \mid \forall_{s' \in \mathcal{S}}. s \xrightarrow{b} s' \Rightarrow s' \in \emptyset\}\} \\
&= \bigcup\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \subseteq \{s \in \mathcal{S} \mid \forall_{s' \in \mathcal{S}}. s \xrightarrow{a} s' \Rightarrow s' \in \mathcal{S}'\} \cap \{s_0, s_2\}\} \\
&= \emptyset \cup \{s_2\} \\
&= \{s_2\}
\end{aligned}$$

Swapping out the box for the diamond gives us $\phi_4 = \nu X.(\langle a\rangle X \wedge [b]f\!f)$, which is satisfied in $s_0$ because $t_1$ exists: no $b$-transitions are enabled in $s_0$ and from $s_0$ we can take an $a$-transition back to $s_0$, so we have a loop where $b$ is never enabled.

$$\begin{aligned}
[\![\phi_4]\!]^M_\epsilon &= \bigcup\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \subseteq [\![\langle a\rangle X \wedge [b]f\!f]\!]^M_{\epsilon[X:=\mathcal{S}']}\} \\
&= \bigcup\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \subseteq [\![\langle a\rangle X]\!]^M_{\epsilon[X:=\mathcal{S}']} \cap \{s_0, s_2\}\} \\
&= \bigcup\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \subseteq \{s \in \mathcal{S} \mid \exists_{s' \in \mathcal{S}}. s \xrightarrow{a} s' \wedge s' \in \mathcal{S}'\} \cap \{s_0, s_2\}\} \\
&= \emptyset \cup \{s_0\} \\
&= \{s_0\}
\end{aligned}$$

### 2.2.3 Positive Normal Form

If there are no negations in a formula besides *ff* (which can be seen as $\neg tt$), the formula is in *positive normal form* (PSN). Any $\mu$-calculus formula can be rewritten to a semantically equivalent formula in positive normal form using the following rules:

- $\neg tt \equiv \mathit{ff}$
- $\neg(\phi \wedge \psi) \equiv (\neg\phi) \vee (\neg\psi)$
- $\neg(\phi \Rightarrow \psi) \equiv \neg((\neg\phi) \vee \psi)$
- $\neg([a]\phi) \equiv \langle a\rangle(\neg\phi)$
- $\neg(\mu X.(\phi)) \equiv \nu X.(\neg\phi[X := \neg X])$

Where $\equiv$ means that two expressions are semantically equivalent: they give the same results when they are evaluated on the same model and environment, regardless of what that model and environment are.

There are some algorithms and tools that are only defined for formulae in positive normal form. We largely do not worry about whether formulae are in PSN here. However, we will at times refer to rewriting formulae to move negation inward, in which case we are applying these rules.

### 2.2.4 Alternation Depth

There are a number of different algorithms for checking whether a given modal $\mu$-calculus formula is satisfied by the initial state of some model. These algorithms are well outside the scope of this thesis. However, for the purposes of evaluating the complexity of the formulae we present, it is useful to introduce the concept of alternation depth. This is because for a many common algorithms, for example the Emerson-Lei algorithm [17], the alternation depth of the formula and the number of states in the model are the most important factors for how expensive it is to check the formula on the model.

In short, the alternation depth of a formula in positive normal form is a measure of how often least and greatest fixpoints alternate in a formula. A greatest fixpoint nested inside a least fixpoint gives a depth of two, but a least fixpoint inside of a least fixpoint only has a depth of one. It is important that the nested fixpoint actually depends on the outer fixpoint, for it to contribute to the alternation depth. For example, in $\nu X.(\langle a\rangle X \vee \mu Y.(\langle b\rangle Y \vee \langle c\rangle tt))$ the least fixpoint does not depend on the variable $X$, and as such the depth of this formula is one, rather than two. A formula is called alternation-free when it has an alternation depth of at most one [35]. Indeed, the formula presented here is alternation-free.

There are a number of different ways in which the alternation depth can be computed. In [9], the authors give a few examples of how the computation proposed in [38] differs from the one used in [17]. We are not concerned with the exact way in which the alternation depth of a formula is calculated, for more information on this topic we point to the cited papers. The intuition provided in this section is sufficient to understand the few remarks on complexity and alternation depth we make in this thesis.

### 2.2.5 Syntactic Extensions

In most of the formulae shown in this report the box and diamond operators are given with a regular expression over sets of actions instead of a single action. This is a rather common extension of the $\mu$-calculus syntax, based on the way operators are defined for Propositional Dynamic Logic (PDL) [18]. As long as we consider finite sets of actions (which as noted in Section 2.1, we do), adding regular expressions does not increase the expressivity of the $\mu$-calculus. It does, however, make it easier to write formulae. Instead of requiring the box and diamond operations to be over single actions, we let them be over regular formulae $R$:

$$\begin{aligned} R, Q &::= \varepsilon \mid \alpha \mid R \cdot Q \mid R + Q \mid R^\star \mid R^+ \\ \alpha, \beta &::= a \mid \mathit{true} \mid \mathit{false} \mid \overline{\alpha} \mid \alpha \cap \beta \mid \alpha \cup \beta \mid \alpha \setminus \beta \end{aligned}$$

Here $\varepsilon$ represents the empty sequence of actions, which is defined as $[\varepsilon]\phi = \langle\varepsilon\rangle\phi = \phi$. The $\alpha$ represents an action formula; $\cdot$ is the concatenations of two regular formulae; $+$ is the union; and finally $^\star$ gives the closure and $^+$ the positive closure of a regular formula. The closure in particular makes it easier to express certain properties, for example $[a^\star]\phi$ represents that $\phi$ must hold in every reachable state reachable with only $a$-transitions. The regular formulae build on top of action formulae rather than simple actions. For the action formulae, $a$ is the set containing exactly the action $a$; *true* and *false* here represent the set of all actions and the empty set respectively; $\overline{\alpha}$ is the set of all actions in *Act* except those in $\alpha$; and $\cap$, $\cup$ and $\setminus$ are the usual set operators. Note that because we have defined $a$ to represent the set containing exactly the action $a$, we will be using set operators on single actions in action formulae, without placing brackets around them.

We do not give new semantics for the box and diamond operator on regular formulae. Instead, we define these operators by the equivalent formulae in the standard modal $\mu$-calculus. As stated previously, when *Act* is finite these new operators do not increase the expressivity of the language, so such translations to the standard modal $\mu$-calculus are always possible.

- $[\alpha]\phi$ where $\alpha$ is an action formula can be expressed as $\bigwedge_{a\in\alpha}[a]\phi$.

- $\langle\alpha\rangle\phi$ as $\bigvee_{a\in\alpha}\langle a\rangle\phi$.
- $[R\cdot Q]\phi$ and $\langle R\cdot Q\rangle\phi$ can be expressed as $[R][Q]\phi$ and $\langle R\rangle\langle Q\rangle\phi$ respectively.
- $[R+Q]\phi$ is $[R]\phi\wedge[Q]\phi$ and $\langle R+Q\rangle\phi$ is $\langle R\rangle\phi\vee\langle Q\rangle\phi$.
- $[R^\star]\phi$ can be expressed as $\nu X.([R]X\wedge\phi)$, and $[R^+]\phi$ as $[R][R^\star]\phi$.
- $\langle R^\star\rangle\phi$ is $\mu X.(\langle R\rangle X\vee\phi)$ and $\langle R^+\rangle\phi$ is $\langle R\rangle\langle R^\star\rangle\phi$.

The CADP toolset's version of the $\mu$-calculus also contains some other extensions of the syntax that do not increase expressivity, such as the option (?) and infinite looping (@) operators [36]. None of the formulae we discussed in this thesis use these additional operators, so we do not include their definitions here.

## 2.3 Tools

Throughout this thesis we use a few tools for checking our results, we introduce them here. For all experiments reported, an HP ZBook Studio G4 (2018) was used with an Intel Core i7-7700HQ CPU, 2.81 GHz processor and 8 GB of RAM.

### 2.3.1 mCRL2

The mCRL2 toolset [11] contains tools that support the modelling and verification of concurrent systems. Modelling is done in the mCRL2 language [24], a process algebra based on the Algebra of Communicating Processes (ACP) [4] with some extensions including the ability to add data to action labels. These models can be interpreted as LTSs, which we can then check properties on. Properties are specified using the modal $\mu$-calculus, extended with the syntactic extensions we introduced in Section 2.2, the ability to quantify over the data in action labels and the addition of data to the formal variables used with fixpoint operators.

More information on mCRL2 can be found in [11, 24] or on the mCRL2 website[3]. We do not explain the mCRL2 language here further. We only show mCRL2 models in Chapter 8, which is the chapter dedicated to how to use our formulae when doing verification with this toolset. The discussion there is primarily of interest to those that are already familiar with mCRL2. We include short explanations of the example models in that chapter for those readers that are unfamiliar with the language but are still interested in what features are required to use our formulae.

[3]The mCRL2 website can be found at `https://www.mcrl2.org/web/index.html`.

Outside of Chapter 8, we also use mCRL2 to confirm our claims whenever we state a particular formula is or is not satisfied in a model in other chapters. We use the June 2022 release of mCRL2.

### 2.3.2 MLSolver

MLSolver [20] is a tool for checking the validity and satisfiability of formulae in a variety of modal fixpoint logics. The link to the tool given in [20] no longer works, but it can still be found on Github[4], together with installation instructions. We use version 1.4 of the tool.

Checking if a formula is valid means checking if the formula is satisfied by all possible models. We use the tool for satisfiability checking instead. A formula is satisfiable if there exists some model which satisfies it. MLSolver supports a number of modal fixpoint logics. Most important for our purposes is that it includes the modal $\mu$-calculus, albeit under the name labelled modal $\mu$-calculus. It also includes CTL, CTL$^{\star}$, PDL and the linear-time $\mu$-calculus with the distribution, and allows for users to add other logics as well. To our knowledge, MLSolver is the only currently existing tool for checking satisfiability of modal $\mu$-calculus formulae.

We use MLSolver several times in this thesis to test whether two formulae are equivalent. To do this, we use that observation that if formula $\phi$ and formula $\psi$ are equivalent, then both $\phi \wedge \neg\psi$ as well as $\neg\phi \wedge \psi$ are unsatisfiable. When we feed our formulae to MLSolver, we do need to modify them a bit. The syntactic extensions we use in this thesis do not appear to be allowed in MLSolver, so we need to reduce our formulae back to the basic $\mu$-calculus syntax. This means we need to use choose an *Act*, since we cannot eliminate expressions like *true* and $\overline{\alpha}$ without knowing the full alphabet. Consequently, when MLSolver reports that two formulae are equivalent, this result only holds for the alphabet we have chosen. The size of the alphabet we can experiment with differs depending on the formulae we are checking, we found that in some cases we could easily go up to an alphabet of size 8, whereas in some cases MLSolver threw an error with an alphabet as small as 3. Due to a lack of documentation, we did not manage to resolve this error.

Regardless of how large the alphabet we can test with, MLSolver will never tell us if two formulae are equivalent for an arbitrary choice of *Act*. We can still use MLSolver to quickly check if it is likely two formulae are equivalent. If the tool reports they are not equivalent, it also gives an example LTS demonstrating this fact, so we have proof of them not being equivalent. If it reports they are equivalent for some specific alphabet, we have strong reason to suspect they are indeed equivalent in general and we can work on proving it manually.

[4]The Github link is archived at `http://web.archive.org/web/20230811153633/https://github.com/tcsprojects/mlsolver`.

# Chapter 3

# Fairness Assumptions

In this chapter, we discuss fairness assumptions: which exist in the literature and which we will be presenting formulae for in the remainder of this thesis.

In Section 2.1.1, we mentioned how progress assumptions reduce the set of finite paths that are considered valid paths. Fairness assumptions work similarly, but for infinite paths. A fairness assumption describes a set of criteria a path must respect in order to be *fair*, if it does not respect these criteria it is *unfair*. By making the assumption, we state that only the fair paths should be considered valid. This is relevant when determining which properties hold, since we only require a property to be satisfied by all valid paths when determining whether it holds for a model.

There are many different fairness assumptions in the literature, distinguished by which paths they deem fair, but they all share the same basic structure: if some choice is possible sufficiently often, it is taken sufficiently often. The precise meanings of "choice", "possible" and "sufficiently often" vary depending on the exact fairness assumption [1]. In this thesis, we are considering LTSs for our models and so "choice" will usually refer to one or several transitions, and "possible" will usually be interpreted as the enabledness or reachability of those transitions. When different models are used, other qualities may be considered such as atomic propositions when using Kripke structures or the enabledness of guards when using programs in the language of guarded commands [14, 19].

Since fairness assumptions only limit the set of infinite paths that are considered, we enforce that regardless of the specifics of the assumption, all finite paths are fair. For some fairness assumptions, the fact that all complete finite paths are fair follows directly from their definition even if no special exception is made for finite paths. For others, this is not the case. We here give all fairness definitions specifically on infinite paths, with the understanding that by definition all finite paths are fair.

We will first introduce a number of different base types of fairness, namely *weak fairness*, *strong fairness*, *unconditional fairness* and *fair reachability*. These types define how the "sufficiently often" and "possible" parts of the fairness assumption should be interpreted. Some different examples of interpretations of "choice" are given afterwards in Section 3.2, answering the question of what it is exactly that is being treated fairly. We will expand on our reasons for choosing these four base types in Section 3.5. In short, weak, strong fairness and unconditional fairness are some of the most commonly discussed forms of fairness. We discuss fair reachability because there is precedent for this type of fairness being used when model checking with modal $\mu$-calculus formulae.

For all our definitions, we fix the model $M = (\mathcal{S}, s_{init}, \mathit{Act}, \mathit{Trans})$. The definitions we present in this chapter use the notion of *tasks* from [22].

**Definition 3.1** (Task)**.** A *task* is a set of transitions of the associated transition system $M$. A task $T \subseteq \mathit{Trans}$ is enabled in a state $s$ if $\exists_{t \in T}.\mathit{source}(t) = s$. A task $T$ *occurs* in $\pi$ if there is at least one transition $t \in T$ that is part of $\pi$.

A transition system can be extended with a set of tasks $\mathcal{T}$. Similar to how transitions can be perpetually or relentless enabled, so can tasks. To re-iterate: a task is perpetually enabled on path $\pi$ if it is enabled in every state of $\pi$; it is relentlessly enabled if each suffix of $\pi$ contains a state in which it is enabled. If we consider a task $T$ consisting only of a single transition $t$, then $t$ being perpetually/relentlessly enabled coincides with $T$ being perpetually/relentlessly enabled. The concept of reachability of transitions can also be extended to tasks: a task $T$ is reachable from a state $s$ if there exists a path starting in $s$ which ends in a state $s'$ where $T$ is enabled in $s'$. Once again, the concept of perpetual and relentless reachability also apply. Here too perpetual and relentless reachability coincide, with the exact same argument as given for Lemma 2.13.

## 3.1 Types of Fairness

Two of the most commonly discussed forms of fairness are *weak fairness* and *strong fairness*. They are based on the concepts of tasks being perpetually and relentlessly enabled respectively.

**Definition 3.2** (Weak fairness)**.** An infinite path $\pi$ is *weakly fair* if for every task $T$ for which there exists a state $s \in \pi$ such that from $s$ onwards $T$ is perpetually enabled, $T$ occurs infinitely often in $\pi$. Equivalently, an infinite path $\pi$ is weakly fair if for every suffix $\pi'$ of $\pi$, every task $T$ which is perpetually enabled in $\pi'$ occurs in $\pi'$. The *weak fairness* assumption states that all infinite paths are weakly fair. Hence, under the weak fairness assumption only weakly fair paths are considered valid paths.

**Definition 3.3** (Strong fairness)**.** An infinite path $\pi$ is *strongly fair* if for every task $T$ which is relentlessly enabled on $\pi$, $T$ occurs infinitely often. Equivalently, an infinite path $\pi$ is strongly fair if for every suffix $\pi'$ of $\pi$, every task $T$ which is relentlessly enabled in $\pi'$ occurs in $\pi'$. The *strong fairness* assumption states all infinite paths are strongly fair. Hence, under the strong fairness assumption only strongly fair paths are considered valid paths.

The concept of weak fairness is called *justice* in [33]. In the same paper, strong fairness is simply called *fairness*. Strong fairness is also sometimes called *compassion* [34]. The names *weak fairness* and *strong fairness* appear in [2].

One form of fairness is considered stronger than another when it labels more paths as unfair, meaning fewer paths are considered valid and hence properties need to be satisfied in fewer paths. Since every task that is perpetually enabled is also relentlessly enabled, every strongly fair path is also weakly fair, but the reverse does not hold. Hence, the conditions for a path to be strongly fair are strictly more restrictive than the conditions for it being weakly fair. Therefore strong fairness is stronger than weak fairness, hence the name.

In addition to weak and strong fairness, there is also *unconditional fairness*, which states that the selected events must happen infinitely often, regardless of how often they are enabled [19]. This type of fairness is often considered too strong in contexts where some events can become disabled. We still give a task-based definition for unconditional fairness because we will use it in our discussion of strong fairness in Chapter 4.

**Definition 3.4** (Unconditional fairness)**.** An infinite path $\pi$ is *unconditionally fair* if every task $T$ occurs infinitely often. Equivalently, an infinite path $\pi$ is *unconditionally fair* if every task $T$ occurs in every suffix $\pi'$ of $\pi$. The *Unconditional fairness* assumption states all infinite paths are unconditionally fair. Hence, if we assume unconditional fairness only unconditionally fair paths are considered valid.

Unconditional fairness appears in [33] under the name *impartiality*. It is in many ways the strongest fairness assumption one can make: the requirement for a choice to be possible sufficiently often is fully removed from the standard fairness assumption structure, leaving only that every "choice" must be made infinitely often. As we will see in Section 3.3, there are some downsides to unconditional fairness which is why we choose to discuss it less in this thesis than the other types of fairness we cover.

The final type of fairness we consider is what we call *fair reachability* (alternatively: fair reachability of tasks). This is an odd one out, since to our knowledge this fairness assumption is not presented under this name in any of the literature, although there are similarly-named related notions. We chose this name in reference to the fair reachability property in [35]. The formula given in that paper is meant to represent a property which is there called "fair reachability", rather than some property under a fairness assumption. However, this exact formula is given in other places such as [41] as a variant of the global response pattern (see Chapter 4) under a not otherwise specified fairness assumption. We reverse-engineered the underlying fairness assumption that one would need to make for this statement to be true. The result is what we present here.

**Definition 3.5** (Fair reachability)**.** An infinite path $\pi$ satisfies *fair reachability* if for every task $T$ for which there exists a state $s \in \pi$ such that from $s$ onward $T$ is perpetually reachable, $T$ occurs infinitely often in $\pi$. Equivalently, an infinite path $\pi$ satisfies fair reachability if for every suffix $\pi'$ of $\pi$, every task $T$ which is perpetually reachable in $\pi'$ occurs in $\pi'$. The fair reachability assumption states all infinite paths satisfy fair reachability. Hence, if we assume fair reachability then only paths satisfying fair reachability are considered valid.

Intuitively, fair reachability is similar to weak fairness except we look at tasks that are perpetually reachable rather than perpetually enabled. Of course, we could give a reachability version of strong fairness as well, but as discussed earlier, perpetual reachability and relentless reachability are equivalent. So the resulting notion would be equivalent to Definition 3.5.

In [40] an assumption called *fair reachability* is presented, which corresponds to our presentation here except it is on states: the fair reachability assumption presented there is that an infinite path is unfair with respect to a set of states $P$ if the states in $P$ are infinitely often reachable but only finitely often visited. This concept is generalised to *fair reachability of predicates*, which says that an infinite path is unfair if there exists some predicate on all states such that the path is unfair with respect to the set of states satisfying this predicate. We concede our use of the term fair reachability may be confusing, in light of this other definition of fair reachability. However, it is our view that the name accurately reflects the definition and gives a clear intuitive description of what the assumption entails. Additionally, the two concepts of fair reachability are similar enough that sharing a name will not lead to confusion, particularly since we do not consider fairness with respect to states in this thesis. Should one need to distinguish the fair reachability we present from fair reachability of predicates, we propose the name *fair reachability of tasks* for Definition 3.5.

Our fair reachability (of tasks) definition corresponds to [6]'s $\infty$-fairness and [31]'s hyperfairness. We did not use these names for the following reasons: $\infty$-fairness does not seem to be well-known, and we find the name not particularly descriptive of the actual definition. The name makes sense in the context of [6] because there $k$-fairness for any natural number $k$ is given as well, but it is not suitable to our discussion here. Hyperfairness is a clearer name, highlighting that this assumption is even stronger than strong fairness (see below). The reason we do not use it is that [45] introduces an alternate definition under the name hyperfairness, we could inspire confusion.

To add fair reachability to our comparison of the relative strength of the assumptions, observe that every task that is relentlessly enabled is also relentlessly reachable and, since relentless reachability is equivalent to perpetual reachability, also perpetually reachable. This means for an arbitrary path, the set of tasks that is relentlessly enabled is a subset of those that are perpetually reachable. Hence, every path that satisfies fair reachability is also strongly fair. Therefore we can say that fair reachability is stronger than strong fairness, and therefore also stronger than weak fairness. It is still weaker than unconditional fairness, of course, since the set of perpetually reachable tasks is still a subset of the set of all tasks. We formalise the arguments on strength we have given in this section here:

**Lemma 3.6.** *Unconditional fairness is stronger than fair reachability, which is stronger than strong fairness, which is stronger than weak fairness.*

*Remark.* At the start of this chapter, we stated that by definition all finite paths are fair and that fairness assumptions can only require conditions to be satisfied on infinite paths. We also noted that for some fairness assumptions, the fact that all finite paths are fair follows immediately from the definition, even if defined on all paths instead of only infinite ones. It turns out that, if we assume progress, this is the case for weak fairness, strong fairness and fair reachability. This can be observed easily: under progress, the only finite paths we consider are those ending in a deadlock state. In a deadlock state, no transitions are enabled and hence no tasks are enabled or reachable. This means that no task can be relentlessly or perpetually enabled or reachable in any suffix of $\pi$. This means that vacuously, every task that is perpetually enabled/ relentlessly enabled/ perpetually reachable in the suffix of a finite complete path will occur in that suffix.

Because of this observation, the restriction that the definition only applies to infinite paths can be scrapped from Definition 3.2, Definition 3.3 and Definition 3.5 when we assume progress. No such argument applies to unconditional fairness, since as long as there are any tasks, no finite path will satisfy the condition that every task occurs infinitely often, regardless of whether we assume progress. Hence, the restriction to infinite paths in Definition 3.4 is important.

## 3.2 Defining Tasks

The tasks referenced in the definitions of weak, strong and unconditional fairness, as well as fair reachability, can be chosen uniquely to the model under consideration. Depending on domain knowledge of the system being modelled, one may choose certain tasks. This is *local fairness*. Alternatively, we can use the specification of $M$ and some set of rules to determine the set of tasks $\mathcal{T}$. This approach is called *global fairness* [22]. We here repeat the global fairness definitions from [22]. These can then be combined with the notions of strong, weak and unconditional fairness, as well as fair reachability, to specify a fairness assumption.

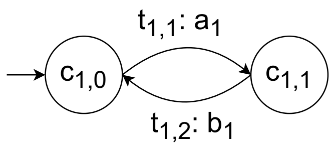
As stated, these global fairness definitions rely on information in the specification of the model. This can be the set of transitions, or the actions those transitions are labelled with. Depending on the model under consideration, there may also be additional information available. A single model $M$ may, for example, be the result of the synchronisation of several *components* taken from a set $\mathcal{C}$, each with their own individual models. We refer to the transitions executed by each component in its own model as *instructions* (taken from a set $\mathcal{I}$). When the components are synchronised, the instructions contribute to the transitions in the resulting model. A transition in $M$ may originate from some specific instruction by one component, but sometimes several components need to synchronise for a particular transition to occur, in which case several components and several instructions contribute to a single transition in $M$. In our discussions, we always let the instruction we discuss correspond to the transitions taken by each component, but the concept of instructions can also be interpreted differently. For example, if $M$ models the execution of some code we may say the instructions corresponding to a transition in $M$ are the lines of code that give rise to that transition.

We add functions *instr* from the set of transitions *Trans* to a subset of the set of instructions $\mathcal{I}$, and *comp* from *Trans* to a subset of the set of components $\mathcal{C}$.

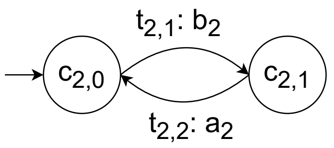
Depending on how *instr* and *comp* are defined for a particular model, the global fairness definitions may result in empty tasks. In fact, certain global fairness definitions are guaranteed to produce at least one empty task. This is not a problem for weak and strong fairness, nor for fair reachability, since an empty task will never be enabled or reachable. However, for unconditional fairness it causes an issue since the empty task can never occur. Since this effect is undesirable, empty tasks are always eliminated from $\mathcal{T}$.

Consider the following example, which we also use as a running example for our task-based definitions.

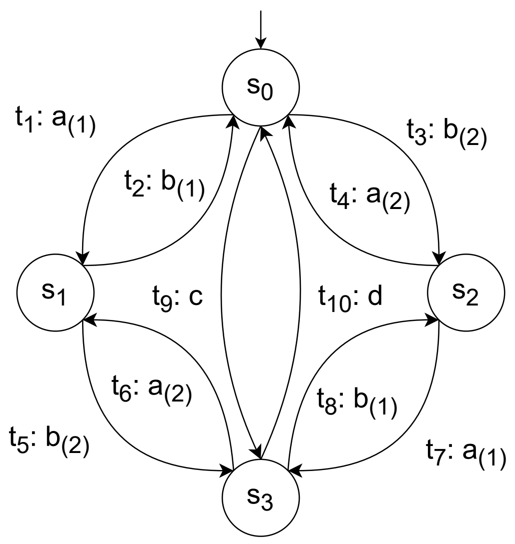
**Example 3.7.** We have a component $C_1$ which can do an $a_1$-action, followed by $b_1$-action which returns it to its initial state. Additionally, we have the component $C_2$ which does a $b_2$-action, then an $a_2$-action, at which point it is back in its initial state. In Figure 3.1, we show both LTSs as well as one way in which they could be synchronised. This synchronisation has an $a$-transition whenever $C_1$ does an $a_1$ action or $C_2$ does an $a_2$ action. Similarly, the synchronisation has a $b$-transition whenever $C_1$ does a $b_1$ or $C_2$ does a $b_2$. In the figure, we have annotated the occurrences of $a$ and $b$ with which component is responsible for that transition,



(a) LTS for $C_1$.



(b) LTS for $C_2$.



(c) LTS for the synchronisation. Each transition has a unique identifier and an action. To each action, we added between brackets which component's instruction corresponds to the action, so $t_1 : a_{(1)}$ represents the first transition, which has action label $a$ and was caused by instruction $a_1$ from component $C_1$.

Figure 3.1: The LTSs described in Example 3.7.

but this is only for illustrative purposes. We have added the action $c$ which can occur when $a_1$ and $b_2$ occur simultaneously, and $d$ which can occur when $a_2$ and $b_1$ occur simultaneously.

The synchronisation, Figure 3.1c, is what we use as our model in this example. The instructions are $a_1$, $a_2$, $b_1$ and $b_2$, the actions are $a, b, c$ and $d$. The function *instr* in this example has $t_1 \mapsto \{a_1\}$, $t_2 \mapsto \{b_1\}$, $t_3 \mapsto \{b_2\}$, $t_4 \mapsto \{a_2\}$, $t_5 \mapsto \{b_2\}$, $t_6 \mapsto \{a_2\}$, $t_7 \mapsto \{a_1\}$, $t_8 \mapsto \{b_1\}$, $t_9 \mapsto \{a_1, b_2\}$ and $t_{10} \mapsto \{a_2, b_1\}$. As for *comp*, $t_1, t_2, t_7$ and $t_8$ all map to $\{C_1\}$, $t_3, t_4, t_5, t_6$ all map to $\{C_2\}$ and $t_9, t_{10}$ both map to $\{C_1, C_2\}$.

We now give the six global fairness definitions from [22]. In these we refer to the mappings *comp* and *instr*. We use Example 3.7 to give examples for each global fairness definition.

## Fairness of Transitions

Fairness of transitions assigns every transition to its own task, giving us

$$\mathcal{T} = \{\{t\} \mid t \in \mathit{Trans}\}$$

For Example 3.7: $\mathcal{T} = \{\{t_1\}, \{t_2\}, \{t_3\}, \{t_4\}, \{t_5\}, \{t_6\}, \{t_7\}, \{t_8\}, \{t_9\}, \{t_{10}\}\}$.

## Fairness of Actions

Fairness of actions assigns transitions to tasks based on the action the transition is labelled with. We define

$$\mathcal{T} = \{T_a \mid a \in \mathit{Act}\}$$

where

$$T_a = \{t \in \mathit{Trans} \mid \mathit{action}(t) = a\}$$

For Figure 3.1c, note that the (1) and (2) annotations to $a$ and $b$-actions are only added to distinguish where the actions came from, they are not part of the action label: $\mathit{Act} = \{a, b, c, d\}$ and hence $\mathcal{T} = \{\{t_1, t_4, t_6, t_7\}, \{t_2, t_3, t_5, t_8\}, \{t_9\}, \{t_{10}\}\}$.

## Fairness of Instructions

Fairness of instructions separates tasks based on if a particular instruction contributed to that task. We define

$$\mathcal{T} = \{T_I \mid I \in \mathcal{I}\}$$

where

$$T_I = \{t \in \mathit{Trans} \mid I \in \mathit{instr}(t)\}$$

In the case of Example 3.7, we get sets of tasks based on the instructions $a_1$, $a_2$, $b_1$ and $b_2$: $\mathcal{T} = \{\{t_1, t_7, t_9\}, \{t_4, t_6, t_{10}\}, \{t_2, t_8, t_{10}\}, \{t_3, t_5, t_9\}\}$.

## Fairness of Synchronisations

For fairness of instructions, we look at whether an instruction contributed to a transition. For fairness of synchronisations, we instead look at the exact subset of $\mathcal{I}$ that is responsible for a transition. We define

$$\mathcal{T} = \{T_S \mid S \subseteq \mathcal{I}\}$$

where

$$T_S = \{t \in \mathit{Trans} \mid \mathit{instr}(t) = S\}$$

Recall that empty tasks are ignored, so $T_\emptyset$, as well as the task for any combination of instructions that do not contribute to a transition together, should not be included in $\mathcal{T}$.

In Example 3.7, we get $\mathcal{T} = \{\{t_1, t_7\}, \{t_4, t_6\}, \{t_2, t_8\}, \{t_3, t_5\}, \{t_9\}, \{t_{10}\}\}$.

**Fairness of Components**

Fairness of components is similar to fairness of instructions, but we look at which components contribute to a transition rather than which instructions. We define

$$\mathcal{T} = \{T_C \mid C \in \mathcal{C}\}$$

where

$$T_C = \{t \in \mathit{Trans} \mid C \in \mathit{comp}(t)\}$$

In Example 3.7, we have the components $C_1$ and $C_2$, and we get two corresponding tasks. Hence, $\mathcal{T} = \{\{t_1, t_2, t_7, t_8, t_9, t_{10}\}, \{t_3, t_4, t_5, t_6, t_9, t_{10}\}$.

**Fairness of Groups of Components**

Fairness of groups of components is to fairness of components what fairness of synchronisations is to fairness of instructions. We look at subsets of components instead of specific components to construct our sets. We define

$$\mathcal{T} = \{T_G \mid G \subseteq \mathcal{C}\}$$

where

$$T_G = \{t \in \mathit{Trans} \mid \mathit{comp}(t) = G\}$$

Similarly to fairness of synchronisation, we ignore empty tasks.

In Example 3.7, we get $\mathcal{T} = \{\{t_1, t_2, t_7, t_8\}, \{t_3, t_4, t_5, t_6\}, \{t_9, t_{10}\}\}$.

*Remark.* We will often refer to fairness assumptions with acronyms, describing both the type of fairness and which global task definition we are considering. For weak fairness, acronyms start with WF; for strong fairness with SF; for unconditional fairness with UF; and finally for fair reachability with FR. The end of the acronym indicates the task definition: T for fairness of transitions, A for fairness of actions, I for fairness of instructions, S for fairness of synchronisation, C for fairness of components, and finally G for fairness of groups of components. For example, we refer to weak fairness of actions as WFA, or to fair reachability of synchronisation as FRS.

We use this notation both to describe assumptions we are making, e.g "we assume WFA", and to describe that a path satisfies a particular assumption. For example, we may say "$\pi$ is an WFA path" to mean that the path $\pi$ satisfies weak fairness of actions.

## 3.3 Feasibility

There are a number of qualities one may judge fairness assumptions on. While a discussion of such qualities falls outside the scope of this thesis, one of them is relevant in some of our later discussions, specifically Chapter 6: *feasibility*.

As repeatedly stated, fairness assumptions limit the set of infinite paths we consider valid and progress assumptions limit the set of finite paths. One may wonder what happens if the combination of assuming and fairness results in there not being *any* valid paths anymore. Take the following example:

**Example 3.8.** Consider Figure 3.2: by the progress assumption from Definition 2.15, only $s_0t_1(s_1t_2)^\omega$ is a valid path. But if we also assume unconditional fairness of actions, then this is not a fair path because $r$ occurs only finitely often. Hence, if we assume both progress and UFA, this system has no paths at all.

Feasibility is a property of fairness assumptions that states such situations cannot occur. We consider feasibility with respect to the different base types of fairness – weak fairness, strong fairness, unconditional fairness and fair reachability – independent from how the tasks as chosen. Adapting the definition of feasibility from [1] to tasks, we give the following definition for feasibility:

**Definition 3.9** (Feasibility)**.** A fairness assumption is *feasible* if, and only if, for any chosen set of tasks, any finite partial path in any model can be extended to a fair complete path.

To prove a fairness assumption infeasible, we merely need to provide a model and set of tasks and demonstrate that some finite partial path cannot be extended to a fair complete path.

**Lemma 3.10.** *Unconditional fairness is infeasible.*

*Proof.* This has been demonstrated by the example above: $s_0t_1s_1$ is a finite partial path but it cannot be extended to a fair path. This example uses unconditional fairness of actions. ☐

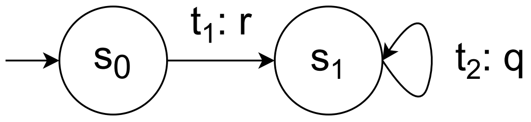


Figure 3.2: An LTS demonstrating how combining progress and unconditional fairness of actions can eliminate all paths from a system.

In [22] it is proven that weak fairness and strong fairness are feasible if there are only countably many tasks enabled in each state. Since we assumed the set of transitions is finite, there are only ever finitely many tasks enabled in each state. The results from [22] therefore apply for all formulae we discuss in this thesis.

**Lemma 3.11.** *Weak fairness is feasible ([22]).*

**Lemma 3.12.** *Strong fairness is feasible ([22]).*

We claim the same holds for fair reachability, also under the assumption $\mathcal{T}$ is finite, which is the case for all formulae and models we consider.

**Lemma 3.13.** *Fair reachability is feasible.*

The proof of this claim is given in Appendix A.4.

## 3.4 Other Fairness Assumptions

The fairness assumptions we have covered up until this point do not cover all types of fairness that appear in the literature. For one reason or another, the other types of fairness we have come across are not relevant to our discussions here.

For example, *equifairness* [25] requires that tasks that meet some condition (depending on whether it is weak equifairness, strong equifairness or unconditional equifairness) must not only *occur* infinitely often, but that all tasks that meet the condition must infinitely often have occurred *the same number of times*. The idea is that if a choice between different options has to be made infinitely often, then it is only fair if every choice gets made the same number of times. This idea is interesting, but impractical to design modal $\mu$-calculus formulae for. We would need to incorporate a counter in our formulae, tracking how much more often one task has been taken than another so far. If the value in this counter can grow unboundedly (but not infinitely) large, we get unboundedly large formulae using the modal $\mu$-calculus syntax we provide here. In future work, we might look at fairness notions similar to equifairness, for example requiring that the difference in occurrence may be at most $k$ for a chosen value of $k$.

In addition to giving an overview of existing fairness assumptions, [22] also proposes two new ones: strong weak fairness and justness. Unsurprisingly, strong weak fairness is weaker than strong fairness and stronger than weak fairness. Strong weak fairness says that if a task is perpetually requested from some point onwards and is relentlessly enabled, it must occur infinitely often. The notions of a task being enabled and occurring have been defined already, but we have not described what it means for a task to be requested. Based on the presentation in [22], this will depend on the model under consideration. It is not clear to us at this time how to generalise the idea of a task being requested in such a way we can easily add it to the list of fairness assumptions we cover in this thesis, so we do not further cover strong weak fairness.

Justness is not exactly a fairness assumption according to Van Glabbeek and Höfner. Instead it is intended as a middle ground between progress and fairness assumptions. Justness excludes infinite paths, so it fits with fairness assumptions in that respect, but it is weaker than all other fairness assumptions we have covered. This makes justness a less controversial assumption to make than other fairness assumptions. Additionally, much like progress assumptions, liveness properties rarely hold on models without assuming justness. The justness assumption boils down to assuming progress for all components underlying a model: we assume each individual component will continue making progress, unless it is interfered with by the actions of another component. This is stronger than just assuming progress, since we also exclude scenarios where the model as a whole makes progress but an individual component does not, even when nothing has interfered with the ability for that component to make progress. Since it does not quite fall under the header of fairness assumptions, we consider justness outside of the scope of this thesis. However, a modal $\mu$-calculus formula for starvation freedom under justness is presented in [7]. In fact, this formula serves as a major inspiration for our weak fairness and fair reachability formulae.

There are also concepts of fairness that rely on different models than LTSs as we present them in Section 2.1. For example, if probabilistic transitions are included in the LTS then we might consider fairness differently. A fair violating path shows a worst-case scenario which results in our desired property not being satisfied. Often, such a path can only exist if the path always makes the "bad" choice whenever a nondeterministic choice has to be made. We accept this for nondeterminism, but when considering probability no "choices" are being made, it is –metaphorically– a die roll. And when we consider an infinite path, the probability of the dice always landing on the "bad" choice becomes zero. At least, as long as there is always an alternative. In [39], Pnueli considers which fairness assumption needs to be made to analyse the nondeterministic version of a probabilistic program in such a way that the results apply to the probabilistic program as well. He introduces *extreme fairness*, which in [22] is explained as letting tasks be defined by any predicate in first-order logic. In this thesis, we do consider arbitrarily chosen tasks but do not take the step of saying something must be fair for all possible choices of tasks simultaneously.

We have previously mentioned fairness assumptions that consider state information, such as fair reachability of predicates [40]. These require the use of a model where state information is available, such as atomic propositions in Kripke structures.

## 3.5 Focus

All these fairness definitions are very interesting, and could make for valuable future work. However, in this thesis we restrict ourselves to the fairness types given in Section 3.1. From our literature study, weak and strong fairness appear to be the two most commonly discussed types of fairness. Unconditional fairness is also commonly discussed, and in [19] it is one of the three classes of fairness that are primarily discussed. The other two being, of course, weak and strong fairness. We also discuss unconditional fairness because we find it helpful for designing formulae for strong fairness. Finally, fair reachability of tasks seems to be a type of fairness that has been considered for modal $\mu$-calculus formulae in the past. As explained in Section 3.1, we actually came to the fair reachability assumption by reverse-engineering the formula for global response under an otherwise unspecified fairness assumption presented in [41]. Hence, there is precedence for fair reachability being used in this context, and the formulae from [41] give us an interesting point of comparison for the formulae we design. Of these four types of fairness, we discuss unconditional fairness less than the other three due to it being infeasible.

Regarding the types of global fairness introduced in Section 3.2, this thesis will primarily discuss fairness of actions. We will discuss WFA, SFA, UFA and FRA in Chapter 4 and Chapter 6. The other task-based definitions, as well as more general formulae for any choice of tasks, will be discussed briefly in Section 7.2. We will go into why we made this choice in Chapter 7 in more detail. For now, we will note that from the syntax of the modal $\mu$-calculus presented in Section 2.2, information on action labels can be referenced directly in formulae, but information on transitions, components and instructions cannot. Our definition of labelled transition systems from Section 2.1 also does not include information on components and instructions. The consequence is that in order to use information on transitions, components and instructions in modal $\mu$-calculus formulae, this information needs to somehow be incorporated into the LTS under consideration and made available in the form of action labels to be usable. This makes fairness of actions the task definition that lends itself the best to being incorporated in modal $\mu$-calculus formulae, as the others require more circuitous methods to be usable.

# Chapter 4

# Global Response Formulae

In this thesis, we consider formulae that follow the property specification patterns from [15]. We briefly mentioned these patterns in Section 1.2, and give a detailed overview of them in Chapter 5. Before exploring all the different patterns, we first spend this chapter on how to design formulae for one specific one: global response. We explain the global response pattern in more detail in Section 4.1. In short, global response states that along every path, whenever the action $q$ occurs it must eventually be followed by the action $r$. The $q$ and $r$ actions are placeholders, how they are filled in depends on the modal and the property.

We start with a specific pattern, rather than directly covering all patterns, so that we can first focus on how the fairness assumptions can be integrated into a formula. We choose global response as the first pattern for two reasons. Firstly, this pattern is frequently used in practice. In the survey of common properties in [15] global response was found to be used to most often. In a similar survey in [41], global response is the third most common, with global absence and global precedence being more frequent. However, as we will argue in Chapter 6, fairness is irrelevant for these patterns. Our second reason is personal interest: one common property that fits the global response pattern is starvation freedom of a mutual exclusion protocol. In different research, we have run into the lack of proven formulae for starvation freedom under weak and strong fairness in particular.

In this section, we present formulae for properties fitting the global response pattern under weak fairness of actions (Section 4.3), fair reachability of actions (Section 4.4 and strong fairness of actions (Section 4.6). We also briefly discuss unconditional fairness of actions (Section 4.5) as a prelude to the strong fairness formula. For all our formulae, we fix the model $M = (\mathcal{S}, s_{init}, Act, Trans)$. Since we are considering fairness of actions, every task is made up of exactly those transitions that are labelled with the same action. For convenience, we therefore abstract away from the tasks themselves and simply speak about actions. In our arguments, we assume progress as defined in Definition 2.15.

## 4.1 The Global Response Pattern

Since we have not yet discussed patterns in detail, we give a more formal introduction to the global response pattern here.

**Definition 4.1** (Global response)**.** A *global response* property holds along a path $\pi$ if for every transition $t_n$ in $\pi$ with $\mathit{action}(t_n) = q$, there exists a transition $t_m$ in $\pi$ with $\mathit{action}(t_m) = r$ and $m > n$.

A global response property under a chosen fairness assumption holds on a model $M$ if it holds on all fair and complete paths of $M$. A path violates global response if there is an occurrence of $q$ which is not followed by $r$.

If we do not make any fairness assumptions, meaning all paths are fair, then the property can be stated in the $\mu$-calculus as follows:

$$[\mathit{true}^\star \cdot q]\mu Y.(\langle \mathit{true}\rangle \mathit{tt} \wedge [\overline{r}]\, Y) \tag{4.1}$$

This can be interpreted as: whenever a $q$-action occurs along any path, there may be at most finitely many non-$r$ steps until we reach a state where $r$, and only $r$, is enabled. The $\langle \mathit{true}\rangle \mathit{tt}$ part of the formula is included to ensure that paths that end in a deadlock without an $r$ having occurred are considered violating paths.

An equivalent formula, stating there does not exist any violating path, is:

$$\neg(\langle \mathit{true}^\star \cdot q\rangle \nu X.([\mathit{true}]\mathit{ff} \vee \langle \overline{r}\rangle X) \tag{4.2}$$

Which expresses that there does not exist a path on which eventually a $q$-action is done, followed by a sequence of non-$r$-actions that is either infinite or eventually reaches a deadlock state. That Formula 4.1 and Formula 4.2 are equivalent can be seen by moving the negation in Formula 4.2 inwards, as described in Section 2.2.3.

Our goal is to modify these formulae so that unfair violating paths are ignored.

## 4.2 Design Approaches

Before designing our own formulae, it is prudent to consider how others have incorporated fairness assumptions into formulae. For this, we looked at papers where properties are used that fit the global response pattern, such as starvation freedom. In our literature study, we identified three general approaches to adding a fairness assumption to a global response property:

1. **Model-specific**: when considering a specific model, it may be possible to identify exactly the unfair violating paths. The model or the property can then be modified to eliminate or ignore those traces. This approach can be

seen in [23]. In this paper an unfair violation path for starvation freedom is found while analysing Dekker's mutual exclusion algorithm. This specific violation is then added to the formula as an allowed path. The new formula detects paths that violate the starvation freedom property but are different from the previously found violation.

2. **Non-violate**: while characterising a fair path and an unfair path are theoretically equally difficult problems, in practice it is often the case that formulating "no fair violation exists" is easier than formulating "all violations are unfair". This approach is to first characterise a fair, violating path and then use a formula to express no such path exists. An example can be seen in [7] where starvation freedom under justness is presented as a $\mu$-calculus formula in this style.

3. **Precondition**: many fairness assumptions are of the form "if something is possible often enough, it must occur". If we have a property that requires a particular event to occur, and we know for certain that event is possible often enough, we can conclude that under this fairness assumption the property will be satisfied. This means that in some cases it is sufficient to express that the event is possible often enough. The conclusion that it occurs then follows directly. This can be seen in the fair reachability formula presented in [35]: it says that after a *send*-action, as long as there has been no *receive*-action, a *receive* action is always reachable. If we assume that every action that is perpetually reachable must eventually occur (FRA) and the formula is satisfied, we can conclude that after every *send* there is eventually a *receive*.

We do not consider the model-specific approach further, since it is not a general solution to the problem of expressing a property holds under a fairness assumption. We do consider both the non-violate and the precondition approaches. Our general preference goes to the non-violate approach, since it is more flexible. We also find that for many fairness assumptions, the precondition approach is difficult to formulate. As such, we only present the non-violate version for most formulae. Where we do give a precondition formula, it is because it has been used by others or it serves as a point of comparison for our non-violate formulae.

## 4.3 Weak Fairness of Actions

Recall that the weak fairness of actions (WFA) assumption states that a path is unfair if there is an action that is perpetually enabled in some suffix of the path but does not occur in that suffix. This means that on all fair paths, if an action is perpetually enabled from some point onwards it is guaranteed to occur.

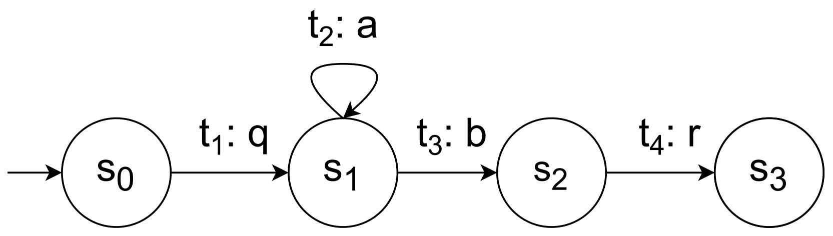


Figure 4.1: A transition system on which Formula 4.3 is false but which does satisfy global response under WFA.

### 4.3.1 Precondition Approach

To express global response is satisfied under WFA we need to express that for all WFA paths, a $q$ action is always followed by an $r$ action. Following the precondition approach, we need to express that, after any $q$-action, $r$ will eventually occur *or* $r$ will be perpetually enabled. After all, if we assume WFA then if $r$ is perpetually enabled it must also occur. This can be expressed in the $\mu$-calculus as follows:

$$[\mathit{true}^\star \cdot q]\mu Y.((\langle \mathit{true}\rangle \mathit{tt} \wedge [\overline{r}]Y) \vee \nu X.((\langle r\rangle \mathit{tt} \vee Y) \wedge [\overline{r}]X)) \tag{4.3}$$

This expresses that after any $q$, as long as no $r$ occurs, we will in finitely many steps reach a state in which $r$ is enabled and from which, as long as we do not do an $r$-step, $r$ will remain enabled. That brief intuitive explanation does not explain why we write $\langle r\rangle \mathit{tt} \vee Y$ instead of simply $\langle r\rangle \mathit{tt}$ inside the greatest fixpoint. The reason for including this is difficult to explain intuitively, but serve as a good example of the complexities of designing modal $\mu$-calculus formulae. A discussion of this detail in the formula is given in Appendix C, for the interested reader.

However, it turns out this formula does not express quite what we want: namely that every action is treated fairly. Instead, it only expresses that $r$ must be treated fairly. Consider the following example.

**Example 4.2.** See Figure 4.1 for a transition system on which Formula 4.3 is false. This is the case because after doing a $q$-action, infinitely many non-$r$ actions (namely $a$) can be done without $r$ ever becoming enabled. However, this is not a weakly fair path since along this path $b$ is perpetually enabled after $q$ occurs but $b$ is never actually taken. Indeed, under WFA the $b$-action would have to occur eventually, which would also ensure $r$ occurs.

Encoding that *every* action must be treated fairly is much more difficult in the precondition approach then that *one* action must be treated fairly. In fact, it is somewhat counter to the core principle of the approach: we want a formula that, combined with the assumption, directly allows us to conclude the property we care about. Incorporating that every action must be treated fairly brings us much closer to fully representing the exact structure of fair and unfair paths, for which the non-violate approach is much more suitable.

Therefore we abandon the precondition approach for weak fairness here. In some cases it may be useful to have a formula where only one action is treated fairly, but that is not what this chapter is addressing. We will return to the concept of expressing that only the $r$-action is treated fairly in Chapter 7.

### 4.3.2 Non-Violate Approach

Inspired by the justness formula in [7], we construct the following formulae for global response under WFA[1]:

$$\begin{aligned}\neg(\langle \mathit{true}^\star \cdot q\rangle \nu X.( \bigwedge_{\lambda \in Act} (\langle \lambda \rangle tt \Rightarrow ( \\ \mu Y.(([\lambda]\mathit{ff} \wedge X) \vee \langle \lambda \setminus r\rangle X \vee \langle \overline{\lambda \cup r}\rangle Y)))))\end{aligned} \tag{4.4}$$

This formula expresses that there does not exist a path such that, after the $q$ action is done, the remainder of the path satisfies WFA but the $r$-action never occurs

The idea behind this formula is that whenever an action $\lambda$ is enabled, we must within a finite number of steps reach a state where $\lambda$ is no longer enabled (then it is not perpetually enabled) or we must take the $\lambda$-action. We do not allow taking the $r$-action at any point so that the resulting path is $r$-free. Whenever the conditions for one $\lambda$ are satisfied, we again check all enabled actions, so $X$ needs to hold again.

Since each action that is enabled must be treated fairly, we know that if this formula is true then we can construct a path where, as long as actions are still enabled, we can extend the path with sequences where these actions either occur or are not perpetually enabled. Should we reach a deadlock state, then $\langle\lambda\rangle tt$ is false for all actions $\lambda \in Act$, and hence the implication is trivially true.

#### Proof

We prove that Formula 4.4 indeed expresses global response under WFA by adapting the proof given in [7] for their justness formula. Note that this proof specifically relies on the set of action $Act$ being finite, and we are assuming progress. While the proofs in this section are quite thorough, we avoid going into extreme detail here. In Chapter 6, we present more general formulae and those get highly detailed correctness proofs in Appendix B. The proof here is a more specialised and slightly less detailed version of the WFA proof given there.

[1]This formula was designed together with Bas Luttik while working on [44].

There are a few propositions and theorems we need for this proof. We do not include the proofs of these propositions and theorems here, they are in Appendix A instead.

**Theorem 4.3.** *For all environments $\epsilon$, states $s \in \mathcal{S}$, modal $\mu$-calculus formulae $\phi$ that do not depend on $Y$, and action formulae $\alpha$, it holds that: $s$ is in $[\![\mu Y.(\phi \vee \langle \alpha \rangle Y)]\!]_\epsilon$ if, and only if, $s$ admits a finite, possibly partial, path on which only actions in $\alpha$ occur and which ends in a state in $[\![\phi]\!]_\epsilon$.*

This proof is given in Appendix A.1.

**Proposition 4.4.** *Let $\pi$ be a WFA path , then $s_0 t_1 s_1 ... t_n \pi$ is a WFA path.*

This is proven in Appendix A.2

**Proposition 4.5.** *Let $\pi = s_0 t_1 s_1...$ be a finite or infinite path. If $\pi$ satisfies WFA then also any suffix of $\pi$ satisfies WFA.*

This is proven in Appendix A.3

We move on to proving things about the formula. We first break the formula up into multiple parts.

$$\begin{aligned}
&\mathit{violate}_{\mathit{WFA}} && = \langle \mathit{true}^\star \cdot q \rangle \mathit{invariant}_{\mathit{WFA}} \\
&\mathit{invariant}_{\mathit{WFA}} && = \nu X.( \bigwedge_{\lambda \in Act} (\langle \lambda \rangle tt \Rightarrow \mathit{satisfy}_{\mathit{WFA}}(\lambda))) \\
&\mathit{satisfy}_{\mathit{WFA}}(\lambda) && = \mu Y.(([\lambda]\mathit{ff} \wedge X) \vee \langle \lambda \setminus r \rangle X \vee \langle \overline{\lambda \cup r} \rangle Y)
\end{aligned}$$

We want to identify states that admit an $r$-free, WFA path after a $q$-action has been done. We prove the formula from the smaller scope to the larger scope, starting with $\mathit{satisfy}_{\mathit{WFA}}(\lambda)$ and ending with $\mathit{violate}_{\mathit{WFA}}$. We first prove that $\mathit{satisfy}_{\mathit{WFA}}(\lambda)$ characterises all states that admit an $r$-free path where $\lambda$ occurs or becomes disabled, and that end in a state that is in the set of states represented by $X$. The idea is as follows: if an action is enabled in some state $s$, then it must either become disabled eventually, in which case the action is not enabled perpetually; or it must be taken within a finite number of steps, in which case it occurs in the suffix starting in $s$.

**Lemma 4.6.** *For all environments $\epsilon$, states $s \in \mathcal{S}$ and actions $\lambda \in Act$ and sets $\mathcal{F} \subseteq \mathcal{S}$, it holds that $s \in [\![\mathit{satisfy}_{\mathit{WFA}}(\lambda)]\!]_{\epsilon[X:=\mathcal{F}]}$ if, and only if, $s$ admits a finite, possibly partial, path $\pi$ that is $r$-free, ends in a state in $\mathcal{F}$ and either*

1. *$\pi$ is $\lambda$-free and ends in a state where $\lambda$ is disabled, or*

*2. the last transition of $\pi$ is labelled with $\lambda$, and $\pi$ is otherwise $\lambda$-free.*

*Proof.* First, by Theorem 4.3, $s \in [\![\mu Y.(([\lambda]\mathit{ff} \wedge X) \vee \langle\lambda \setminus r\rangle X \vee \langle\overline{\lambda \cup r}\rangle Y)]\!]\epsilon[X := \mathcal{F}]$ if, and only if, $s$ admits a finite, possibly partial path $\pi'$ that ends in a state $s'$ such that only actions in $\overline{\lambda \cup r}$ occur in $\pi'$ and $s' \in [\![([\lambda]\mathit{ff} \wedge X) \vee \langle\lambda \setminus r\rangle X]\!]_{\epsilon[X:=\mathcal{F}]}$. Since only actions in $\overline{\lambda \cup r}$ occur in $\pi'$, $\pi'$ is both $\lambda$-free and $r$-free. From $s' \in [\![([\lambda]\mathit{ff} \wedge X) \vee \langle\lambda \setminus r\rangle X]\!]_{\epsilon[X:=\mathcal{F}]}$, we know that $s'$ is either a state where $\lambda$ is disabled and $s' \in \mathcal{F}$, or $s'$ admits a $\lambda$-transition $t$ to a state $s''$ in $\mathcal{F}$ and, $\lambda \neq r$. In the former case, $\pi'$ is a finite, $r$-free path that ends in a state in $\mathcal{F}$, is $\lambda$-free and ends in a state where $\lambda$ is disabled. In the latter case, $\pi'$ extended with $t$ and $s''$ is a finite, $r$-free path ending in a state in $\mathcal{F}$ such that the last transition is labelled with $\lambda$ and the path is otherwise $\lambda$-free. Hence, the semantics of $\mathit{satisfy}_{\mathit{WFA}}$ are indeed those states that admit a path meeting the requirements stated in the lemma. $\square$

Using Lemma 4.6, we prove that $\mathit{invariant}_{\mathit{WFA}}$ exactly characterises those states that admit an $r$-free, WFA path. We call the set of all such states $S_{r,\mathit{WFA}}$. We do this in two steps.

**Lemma 4.7.** *The set of states admitting $r$-free, WFA paths, $S_{r,\mathit{WFA}}$, is a fixed point of the following transformer $T_{\mathit{WFA}}$:*

$$T_{\mathit{WFA}}(\mathcal{F}) = \bigcap_{\lambda \in \mathit{Act}} \{s \in \mathcal{S} \mid s \in [\![\langle\lambda\rangle \mathit{tt}]\!]_{\epsilon[X:=\mathcal{F}]} \Rightarrow s \in [\![\mathit{satisfy}_{\mathit{WFA}}(\lambda)]\!]_{\epsilon[X:=\mathcal{F}]}\}$$

*for arbitrary environment $\epsilon$.*

*Proof.* We prove $S_{r,\mathit{WFA}}$ is a fixed point of $T_{\mathit{WFA}}$ by showing that $T_{\mathit{WFA}}(S_{r,\mathit{WFA}}) = S_{r,\mathit{WFA}}$. We prove this through mutual set inclusion of $S_{r,\mathit{WFA}}$ and $T_{\mathit{WFA}}(S_{r,\mathit{WFA}})$.

- Let $s$ be an arbitrary state in $S_{r,\mathit{WFA}}$. We prove $s \in T_{\mathit{WFA}}(S_{r,\mathit{WFA}})$. Since $s \in S_{r,\mathit{WFA}}$, we know $s$ admits an $r$-free, WFA path. Let $\pi$ be such a path starting in $s$. We need to prove $s \in T_{\mathit{WFA}}(\mathcal{F})$ by showing that for all $\lambda \in \mathit{Act}$, if $\lambda$ is enabled in $s$ then $s \in [\![\mathit{satisfy}_{\mathit{WFA}}(\lambda)]\!]_{\epsilon[X:=S_{r,\mathit{WFA}}]}$. Since the condition in the transformer is an implication, $s$ is trivially in the set associated with any action not enabled in $s$. Let $\lambda$ be an arbitrary action that is enabled in $s$. We must show that $s \in [\![\mathit{satisfy}_{\mathit{WFA}}(\lambda)]\!]_{\epsilon[X:=S_{r,\mathit{WFA}}]}$ for this $\lambda$. We do a case distinction on whether $\lambda$ is perpetually enabled on $\pi$.
  - If $\lambda$ is perpetually enabled on $\pi$, then by definition of weak fairness of actions $\lambda$ must occur in $\pi$. Let $s'$ be the first state in $\pi$ reached by a $\lambda$-transition. The path from $s$ to $s'$, $\pi'$ is finite and $r$-free because $\pi$ is $r$-free. Additionally, through our choice of $s'$ we know that none but the last transition on this path are $\lambda$-transitions. To prove $s \in$

$[\![\mathit{satisfy}_{\mathit{WFA}}]\!]_{\epsilon[X:=S_{r,\mathit{WFA}}]}$, it only remains to prove that $s' \in S_{r,\mathit{WFA}}$. We can then apply Lemma 4.6.

By Proposition 4.5, the suffix $\pi''$ of $\pi$ starting in $s'$ satisfies WFA, and since $\pi$ is $r$-free so is $\pi''$. Hence $s'$ admits an $r$-free, WFA path and therefore $s' \in S_{r,\mathit{WFA}}$. We conclude $s \in [\![\mathit{satisfy}_{\mathit{WFA}}(\lambda)]\!]_{\epsilon[X:=S_{r,\mathit{WFA}}]}$.

  – If $\lambda$ is not perpetually enabled on $\pi$, then in finitely many steps of $\pi$ a state $s'$ is reached where $\lambda$ is disabled. None of these steps are $r$-transitions, since $\pi$ is $r$-free. We do a case distinction on whether there is an occurrence of $\lambda$ on $\pi$ before $s'$.

    ∗ If there is such a transition, then we can apply the same argument for $s$ being in $[\![\mathit{satisfy}_{\mathit{WFA}}(\lambda)]\!]_{\epsilon[X:=S_{r,\mathit{WFA}}]}$ as we did in the case that $\lambda$ is perpetually enabled. After all, we only used $\lambda$ being perpetually enabled to conclude it must occur.

    ∗ If there is no occurrence of $\lambda$ before $s'$, then then let $\pi'$ be the prefix of $\pi$ ending in $s'$. Not only is $\pi'$ $r$-free, it is also $\lambda$-free. Additionally, $\lambda$ is disabled in $s'$. To be able to apply Lemma 4.6 to conclude $s \in [\![\mathit{satisfy}_{\mathit{WFA}}(\lambda)]\!]_{\epsilon[X:=S_{r,\mathit{WFA}}]}$, we only need to prove $s' \in S_{r,\mathit{WFA}}$.
    By Proposition 4.5, the suffix $\pi''$ of $\pi$ starting in $s'$ satisfies WFA, and since $\pi$ is $r$-free so is $\pi''$. Hence $s'$ admits an $r$-free, WFA path and therefore $s' \in S_{r,\mathit{WFA}}$. Hence, $s \in [\![\mathit{satisfy}_{\mathit{WFA}}(\lambda)]\!]_{\epsilon[X:=S_{r,\mathit{WFA}}]}$.

We conclude that $s \in [\![\mathit{satisfy}_{\mathit{WFA}}(\lambda)]\!]_{\epsilon[X:=S_{r,\mathit{WFA}}]}$ and hence also that for all actions enabled in $s$, $s$ is in the associated set. We conclude that $s \in T_{\mathit{WFA}}(S_{r,\mathit{WFA}})$.

- Let $s$ be an arbitrary state in $T_{\mathit{WFA}}(S_{r,\mathit{WFA}})$. We prove $s \in S_{r,\mathit{WFA}}$. From $s \in T_{\mathit{WFA}}(S_{r,\mathit{WFA}})$ we know that for all actions $\lambda$ that are enabled in $s$, $s \in [\![\mathit{satisfy}_{\mathit{WFA}}(\lambda)]\!]_{\epsilon[X:=S_{r,\mathit{WFA}}]}$. If $s$ is a deadlock state, then it admits the path consisting of only itself. This path is trivially $r$-free and, since it is finite, WFA. Hence, we have a witness for $s \in S_{r,\mathit{WFA}}$.

  If $s$ is not a deadlock, let $\lambda$ be an arbitrary action that is enabled in $s$. Then $s \in [\![\mathit{satisfy}_{\mathit{WFA}}(\lambda)]\!]_{\epsilon[X:=S_{r,\mathit{WFA}}]}$. By Lemma 4.6, there must be a finite $r$-free path $\pi$ from $s$ to some $s'$ such that $s' \in S_{r,\mathit{WFA}}$. Since $s' \in S_{r,\mathit{WFA}}$ we know $s'$ admits some $r$-free, WFA path $\pi'$. By Proposition 4.4, $\pi$ extended with $\pi'$ satisfies WFA and since both $\pi$ and $\pi'$ are $r$-free, so is their combination. Hence $s$ admits an $r$-free, WFA path and therefore $s \in S_{r,\mathit{WFA}}$.

We conclude that the set of all $r$-free, WFA paths $S_{r,\mathit{WFA}}$ is a fixed point of $T_{\mathit{WFA}}$. □

Note that the semantics of $invariant_{WFA}$ is the greatest fixpoint of the transformer $T_{WFA}$. Hence, we need to prove this greatest fixpoint equals $S_{r,WFA}$.

**Lemma 4.8.** *The set of all $r$-free, WFA paths $S_{r,WFA}$ is the greatest fixed point of $T_{WFA}$ as defined in Lemma 4.7.*

*Proof.* We show that for any $\mathcal{F}$ satisfying $T_{WFA}(\mathcal{F}) = \mathcal{F}$, we have that $\mathcal{F} \subseteq S_{r,WFA}$. Let $\mathcal{F}$ be an arbitrary subset of $\mathcal{S}$ that is a fixed point for $T_{WFA}$. Let $s \in \mathcal{F}$. We prove $s \in S_{r,WFA}$. We do this by constructing an $r$-free, WFA path $\pi$ from $s$. Observe that since $s \in \mathcal{F}$ and $\mathcal{F} = T_{WFA}(\mathcal{F})$, we also have $s \in T_{WFA}(\mathcal{F})$. If there are no enabled actions in $s$, then trivially it admits an $r$-free, WFA path in the path consisting only of $s$. We therefore assume that there are enabled actions in $s$. Let $L$ be the set of all enabled actions in $s$.

Consider an arbitrary but fixed order $<$ on the actions in $L$ and let $\lambda$ be the least of these actions. From $s \in T_{WFA}(\mathcal{F})$ we conclude that there exists some finite, $r$-free path $\pi'$ from $s$ to a state $s'$ such that $s' \in \mathcal{F}$ and either $\lambda$ is not enabled in $s'$ or $\lambda$ is the last action in $\pi'$. We will let $\pi'$ be the start of our constructed path $\pi$.

Denote the set of actions enabled in $s'$ by $L'$. We next choose the least $\lambda' \in \{\lambda'' \in L \cap L' \mid \lambda < \lambda''\}$, so the smallest action that is larger than $\lambda$ and is enabled in both $s$ and $s'$. Since $s' \in \mathcal{F}$ and $\mathcal{F} = T_{WFA}(\mathcal{F})$ we can apply the same construction to find a finite $r$-free path $\pi''$ from $s'$ to $s''$ such that either $\lambda'$ is not enabled in $s''$ or $\lambda'$ is the last action in $\pi''$. We add $\pi''$ to our constructed path $\pi$.

We repeat this construction until there are no more actions that were enabled in $s$ as well as every final state of our path segments. This construction will terminate, since there are finitely many actions. Once the construction stops in a state $s_{\mathit{final}}$, we continue extending the path constructed so far with the same construction method, now letting $L$ be the set of actions enabled in $s_{\mathit{final}}$. This leads to either an infinite path, or a finite path where there are no actions enabled in the final state.

A path constructed in this manner is trivially $r$-free, since we only ever add path segments that are $r$-free. It is less trivial to see that the resulting path satisfies WFA. Let $\pi_{con}$ be the final path we have constructed. If it is finite, it is trivially WFA. Hence, we henceforth assume $\pi_{con}$ is infinite. Consider a suffix $\pi'_{con}$ of $\pi_{con}$ where an action $\alpha$ is enabled perpetually. Since $\pi'_{con}$ is infinite and our construction procedure is finite until we refresh the set of actions $L$ that need to be satisfied, we know that starting in the first state of $\pi'_{con}$ there will be finitely many steps until we reach a state in which the set of actions $L$ is refreshed. Since $\alpha$ is perpetually enabled in $\pi'_{con}$, $\alpha$ will be part of the new set $L$. During construction of the next stretch of the path, we at some point add a path segment in which $\alpha$ either becomes disabled or occurs. If $\alpha$ could become disabled, it would not be perpetually enabled, hence we are sure we have added a path segment in which $\alpha$ occurs. Hence, $\alpha$ occurs in $\pi'_{con}$.

We conclude that, since every action that is perpetually enabled in some suffix of $\pi_{con}$ is guaranteed to occur in that suffix, our constructed path satisfies WFA. We have therefore proven we can construct a path from $s$ which is both $r$-free and satisfies WFA. Hence, $s \in S_{r,WFA}$ and thus $S_{r,WFA}$ is the greatest fixed point of $T_{WFA}$. □

**Lemma 4.9.** *For all environments $\epsilon$ and states $s \in \mathcal{S}$, we have that $s$ is in $[\![invariant_{WFA}]\!]_\epsilon$ if and only if $s$ admits an $r$-free, WFA path.*

*Proof.* As stated informally previously, the semantics of $invariant_{WFA}$ is exactly the greatest fixpoint of $T_{WFA}$. As proven in Lemma 4.8, the semantics of *invariant* therefore equals $S_{r.WFA}$, the set of all states which admit $r$-free, WFA paths. □

Next, we need to prove that $violate_{WFA}$ indeed characterises the existence of a WFA path that violates global response.

**Lemma 4.10.** *For all environments $\epsilon$ and states $s \in \mathcal{S}$, we have that $s$ is in $[\![violate_{WFA}]\!]_\epsilon$ if, and only if, $s$ admits a WFA path that violates global response.*

*Proof.* This follows directly from the definition of $violate_{WFA}$ and Lemma 4.9. By the semantics of $violate_{WFA}$, the states $s \in [\![violate_{WFA}]\!]_\epsilon$ are exactly those states that admit a finite, possibly partial, path $\pi$ where the final transition is labelled with a $q$-action, and the subsequent state is in $[\![invariant_{WFA}]\!]_\epsilon$, hence admits a path $\pi'$ that is $r$-free and satisfies WFA. By Proposition 4.4, $\pi\pi'$ satisfies WFA. So we have a WFA path violating global response. □

The final conclusion then follows quite directly.

**Theorem 4.11.** *A state satisfies Formula 4.4 if, and only if, it satisfies global response under weak fairness of actions.*

*Proof.* The formula Formula 4.4 is the negation of $violate_{WFA}$. By Lemma 4.10, a state satisfies $violate_{WFA}$ if, and only if, it admits a WFA path that violates global response. Hence, a state satisfies Formula 4.4 if, and only if, it does not admit such a path. □

## 4.4 Fair Reachability of Actions

Recall that the fair reachability of actions (FRA) assumption states that a path is unfair if there is an action that is perpetually reachable on the path but occurs finitely often.

### 4.4.1 Precondition Approach

The fair reachability formula from [35] is the example we used to identify the precondition approach. It was not designed to be global response under fair reachability of actions, it was instead designed as the "fair reachability" *property.* However, it corresponds to global response under FRA. Indeed, it is given as global response under fairness assumption in [41]. While it is not specified which fairness assumption is made, at least for this formula it corresponds to our FRA assumption. In fact, this is the formula from which we reverse-engineered the FRA assumption. We give Remenska's variant here:

$$[\mathit{true}^\star \cdot q \cdot \overline{r}^\star]\langle \mathit{true}^\star \cdot r\rangle \mathit{tt} \tag{4.5}$$

This formula expresses that whenever a $q$ is done, as long as no $r$ has been done it must always be the case that there exists a path along which $r$ becomes enabled. With the assumption that any action that perpetually reachable must occur, this means $r$ always occurs after $q$.

### 4.4.2 Non-Violate Approach

We also adapt our global response under WFA formula for a non-violate version of global response under FRA. We do this by changing the condition for an action to be a candidate for $\lambda$ from it being enabled to it being reachable. Additionally, an action is now only "satisfied" if it occurs or no longer reachable, rather than if it occurs or no longer enabled.

$$\begin{aligned}\neg(\langle \mathit{true}^\star \cdot q\rangle \nu X.( \bigwedge_{\lambda \in Act} \langle \mathit{true}^\star \cdot \lambda\rangle \mathit{tt} \Rightarrow (\\ \mu Y.(([\mathit{true}^\star \cdot \lambda]\mathit{ff} \wedge X) \vee \langle \lambda \setminus r\rangle X \vee \langle \overline{\lambda \cup r}\rangle Y))))\end{aligned} \tag{4.6}$$

The proof that the semantics of Formula 4.6 indeed captures exactly those states that admit a violating path satisfying fair reachability of actions is very similar to the proof in Section 4.3.2. We therefore do not repeat it here.

**Theorem 4.12.** *A state satisfies Formula 4.6 if, and only if, it satisfies global response under fair reachability of actions.*

In Appendix A.5, Theorem 4.12 is proven.

### 4.4.3 Comparison of Approaches

In the section of weak fairness of actions, we stated the precondition formula corresponds to only assuming $r$ is treated fairly, whereas the non-violate formula corresponds to assume all actions are treated fairly. At first glance, this is also the case for the two fair reachability of action formulae: the precondition formulae only has a requirement that $r$ should be reachable, whereas the non-violate formula considers all actions. However, we argue here that for this specific fairness assumption, the two approaches actually result in equivalent formulae.

We first used the MLSolver tool, described in Section 2.3.2, to do an initial check of our intuition that the two formulae are equivalent. In this case, we could only go up to an alphabet of size 2, specifically $\{q, r\}$, since MLSolver threw an error when we tried to go up to size 3. We could conclude that with an alphabet of size 2, the two formulae are indeed equivalent. We would be more confident the formulae are equivalent if we could test with a larger alphabet, but this still gave an indication they are likely to be equivalent.

This gives us the confidence to manually prove the two formulae equivalent.

**Theorem 4.13.** *The formulae Formula 4.5 and Formula 4.6 are equivalent, they are satisfied by exactly the same models.*

*Proof.* For this proof, we use Theorem 4.12: we know that Formula 4.6 exactly describes the non-existence of FRA paths that violate global response. All we then need to do is prove that when Formula 4.5 is satisfied in model $M$, that model satisfies global response under FRA and that when Formula 4.5 is not satisfied, $M$ contains a fair violating path.

- Let $M$ be an arbitrary model that satisfies Formula 4.5, we prove $M$ must then satisfy global response under FRA. Note that, as we previously described, the formula states that after every $q$-action, as long as no $r$-action has occurred, $r$ must always be reachable. Towards a contradiction, assume there exists a fair and complete path $\pi$ in $M$ which violates global response. Then this path contains an occurrence of the action $q$ that is never followed by an occurrence of $r$. Since $M$ satisfies Formula 4.5, we know that after the occurrence of this $q$ on $\pi$, as long as $r$ does not occur, $r$ will be reachable. Since $r$ does not occur after $q$ on $\pi$ at all, this means $r$ is reachable in every state on $\pi$ after this occurrence of $q$. Consider the suffix $\pi'$ of $\pi$ which starts directly after this occurrence of $q$. We know $r$ is perpetually reachable on $\pi'$, and that there is no occurrence of $r$ on $\pi'$. This violates the assumption that $\pi$ is a path that satisfies FRA. Hence, no such path $\pi$ can exist and there must not be any fair complete paths violating global response in $M$.

- Let $M$ be an arbitrary model that does not satisfy Formula 4.5, we prove $M$ must then contain an FRA path violating global response. If $M$ does not satisfy the formula, we know there must be a path $\pi$ on which a $q$ occurs, and after this $q$ there is a sequence of zero or more non-$r$ actions which take us to a state $s$ from which $r$ is not reachable. This follows directly from the formula. Now consider the possibly partial path $\pi'$ from the initial state of $M$ up to $s$: on $\pi'$, a $q$-action occurs and for the rest of $\pi'$ until its final state, $s$, no $r$-actions occur. We know fair reachability is feasible, and so there exists a complete fair path $\pi''$ which extends the path $\pi'$. We know no $r$ occurs after the $q$ on $\pi'$, and we also know that since $r$ is unreachable from $s$, that there cannot be any $r$-actions after $s$ on $\pi''$, regardless of how the extension is done. Hence, we know there is a fair and complete path $\pi''$ on which a $q$-action occurs which is not followed by an $r$-action.

We conclude that Formula 4.5 is satisfied in exactly those models that satisfy global response under FRA, and hence its semantics are the same as Formula 4.6. Hence, the two formulae are equivalent. □

## 4.5 Unconditional Fairness of Actions

Unconditional fairness of actions is the assumption every action is taken infinitely often. To our knowledge this is not a form of fairness that is often assumed in practice, it seems strange to require every action occurs infinitely often without taking into account if the action is even enabled. As discussed in Section 3.3, unconditional fairness is also infeasible, which according to some means it can be disregarded as a fairness assumption. In [22], for example, the choice is made to only consider feasible fairness assumptions. We largely support the idea that infeasible fairness assumptions are not sensible to use in most cases, so we will not discuss unconditional fairness after this chapter, save for a handful of brief remarks. We do present formulae for global response under UFA in this section, mainly because the last of these formulae will form the basis for the SFA formula presented in the next section.

It should be noted that the UFA assumption is certainly not much use for the global response pattern. Consider that a path violating global response is a path along which a $q$ action is taken at some point, followed by either a finite $r$-free path ending in a deadlock state or an infinite $r$-free path. Assuming UFA would mean that every action is taken infinitely often along an infinite path, so under this assumption there will never be an infinite path along which no $r$-action occurs. Hence, if we assume UFA all infinite violating paths are disregarded as unfair. The only valid violating paths for global response under UFA are therefore paths along which, after some $q$-action, a deadlock can be reached without doing any $r$-actions.

The absence of such paths can be expressed as shown in Formula 4.7. This formula is in neither the precondition nor non-violate style. We are describing the absence of a violating path, as we do in the non-violate style; yet we are also including the assumption that any infinite path will by assumption satisfy global response, which is more similar to the precondition style.

$$\neg(\langle \mathit{true}^{\star} \cdot q \cdot \overline{r}^{\star}\rangle[\mathit{true}]\mathit{ff}) \tag{4.7}$$

We can rewrite this to be properly in the precondition style by moving the negation inward, see Formula 4.8.

$$[\mathit{true}^{\star} \cdot q \cdot \overline{r}^{\star}]\langle \mathit{true}\rangle \mathit{tt} \tag{4.8}$$

This expresses that after a $q$, as long as no $r$ has occurred we are not in a deadlock.

More interesting is designing a proper non-violate style formula, in which we explicitly encode that all actions must occur infinitely often on infinite paths. With a least fixpoint operator, we can straightforwardly express that some action must occur in finitely many steps. To express that the action $\lambda$ can occur in finitely many steps without taking an $r$-action, we use $\mu Y.(\langle\overline{r}\rangle\, Y \vee \langle\lambda \setminus r\rangle \mathit{tt})$. If we wish to express both $\lambda_1$ and $\lambda_2$ must occur in finitely many steps, and we know $\lambda_1$ must occur first, we can nest the occurrence of $\lambda_2$ inside the formula for the occurrence of $\lambda_1$ as follows:

$$\mu Y.(\langle\overline{r}\rangle\, Y \vee \langle\lambda_1 \setminus r\rangle \mu W.(\langle\overline{r}\rangle\, W \vee \langle\lambda_2 \setminus r\rangle \mathit{tt}))$$

This expresses that after the $\lambda_1$ has occurred, it must be possible to reach $\lambda_2$ in finitely many steps. We can continue this nesting to an arbitrary depth, for all actions in *Act*. This of course increases the number of formal variables used linearly with respect to the number of actions, but since these are all least fixpoint operators the alternation depth of the formula is constant.

The formula above does rely on knowing for sure that $\lambda_2$ will occur after $\lambda_1$. Luckily, if we are describing an infinite path along which all actions will occur infinitely often, then there will be *some* occurrence of $\lambda_2$ after $\lambda_1$, even if there is also an occurrence of $\lambda_2$ before $\lambda_1$. This insight is what lets us keep the formula relatively simple.

Expressing that there is no path such that, after a $q$, all actions occur but $r$ does not occur can be done as:

$$\neg(\langle \mathit{true}^{\star} \cdot q\rangle \nu X.\mathit{inf}_{\mathit{UFA}})$$

Where $\mathit{inf}_{\mathit{UFA}}$ is defined as follows, where we fix some arbitrary order on the actions in *Act* such that $\alpha_1$ is the first and $\alpha_n$ is the last:

$$\begin{aligned}
&\mathit{inf}_{\mathit{UFA}} && = \mathit{exec}_{\mathit{UFA}}(n)\\
&\mathit{exec}_{\mathit{UFA}}(0) && = X\\
&\mathit{exec}_{\mathit{UFA}}(k+1) && = \mu W_{k+1}.(\langle\overline{r}\rangle W_{k+1} \vee \langle\alpha_{k+1} \setminus r\rangle \mathit{exec}_{\mathit{UFA}}(k))
\end{aligned}$$

Here we use the same nesting we previously demonstrated with $\lambda_1$ and $\lambda_2$, now applied to all actions $\alpha_1, \dots, \alpha_n \in Act$. The greatest fixpoint operator is used so that whenever we have seen at least one occurrence of every action, the cycle repeats.

The above formula is not quite correct to express global response under UFA, however. Specifically, we have not allowed for the possibility that we reach a deadlock without having done $r$, which would mean we have a fair, violating path. This will need to be included explicitly as an option after the $q$ has occurred. This gives us the final non-violate style formula shown in Formula 4.9.

$$\neg(\langle true^\star \cdot q\rangle((\langle \overline{r}^\star\rangle[true]ff) \vee \nu X.inf_{UFA})) \tag{4.9}$$

With the same definition of $inf_{UFA}$ as given previously.

We may wonder how does this, much more complicated-looking, formula compares to the formulae we have presented previously in this section. We expect them to be equivalent, since they all represent global response under UFA, but we should confirm this. It turns out that not only is Formula 4.9 equivalent to the other two formulae we have presented, we can quite easily reduce it to Formula 4.7. To do this, we note that $\nu X.inf_{UFA}$ can never be true: it describes a path along which all actions infinitely often occur, and yet simultaneously the $r$ action never occurs. This is an impossibility. Hence, $\nu X.inf_{UFA}$ is always false, and we can eliminate it from the disjunction in Formula 4.9. Doing this results in $\neg(\langle true^\star \cdot q\rangle((\langle \overline{r}^\star\rangle[true]ff)))$, which, after removing some unnecessary brackets and using the fact that $\langle\alpha\rangle\langle\beta\rangle\phi = \langle\alpha \cdot \beta\rangle\phi$, is exactly the same as Formula 4.7.

We will not discuss unconditional fairness much in subsequent chapters. However, as we will see in the next section, the SFA formula can easily be turned into the UFA formula. Should one ever need an unconditional fairness formula, the strong fairness formulae we present throughout this thesis can easily be turned into unconditional fairness formulae.

**Theorem 4.14.** *A state satisfies Formula 4.9 if, and only if, it satisfies global response under unconditional fairness of actions.*

We do not provide a formal proof of this theorem: we have argued the correctness of Formula 4.7, and its equivalence to Formula 4.9. Correctness of Formula 4.9 follows from this.

## 4.6 Strong Fairness of Actions

Recall that the strong fairness of actions assumption labels any path unfair for which there is an action that is relentlessly enabled yet only occurs finitely often. In other words, on all fair paths all relentlessly enabled actions must occur infinitely often. The formula for global response under SFA was by far the most complex to design, and while we made many attempts at designing an elegant and efficient formula, we had to settle for an inefficient one and rather ugly one. Here, we present a formula in the non-violate style that is correct, but as will be demonstrated in Chapter 8, is not usable in practical model checking scenarios due to its inefficiency, at least not when there are many tasks. We do not have a formula in the precondition style for SFA.

The main challenge in designing a formula for SFA is that when an action is enabled along a path, SFA can be satisfied both by the action being taken within finitely many steps, or by the action eventually never being enabled again. This can be expressed for a single action in the non-violate style relatively easily. Consider Formula 4.10, which expressed that there does not exist an $r$-free path after a $q$-action, on which the $\lambda$-action is treated fairly. This formula is based on a formula presented in [8].

$$\begin{aligned}\neg(&\langle \mathit{true}^\star \cdot q\rangle \nu X.\mu Y.\nu Z.( \\ &\langle \lambda \setminus r\rangle X \vee (\langle\lambda\rangle \mathit{tt} \wedge \langle\overline{\lambda \cup r}\rangle Y) \vee ([\lambda]\mathit{ff} \wedge \langle\overline{\lambda \cup r}\rangle Z) \vee [\mathit{true}]\mathit{ff}))\end{aligned} \tag{4.10}$$

The triple-nested fixpoints in this formula make it difficult to understand intuitively. The core idea is that you may infinitely often do the $\lambda$-action ($X$), and that you may only finitely often be in a state where $\lambda$ is enabled but not taken ($Y$). However, you may infinitely often go through a state where $\lambda$ is simply not enabled ($Z$), even while within the least fixpoint.

We have found extending this formula to cover a path along which *all* actions are treated fairly to be rather difficult. The main problem we run into is the following: when we are in an infinite trail along which some action $\lambda$ is never enabled ($Z$), we must still consider all actions that may or may not be enabled along this path and how often we have seen them, and maintain that information even when $\lambda$ becomes enabled again. We would conceptually need separate $X$, $Y$ and $Z$ fixpoints for every action, but also nested together in every possible arrangement. Even then, we would need to be very careful to not accidentally exclude any scenarios, and correctly maintain all the information in the formula. We were unable to make this work, and either way it was quite clear this approach would not result in an elegant formula. Perhaps there exists a way to turn Formula 4.10 into a nice formula that covers all actions, but we are not aware of it.

Instead, we came upon the following idea: along a strongly fair path, there are some actions that occur infinitely often and some actions that are only finitely often enabled. In fact, every action in *Act* must fall into exactly one of those two categories. After all, actions are either enabled finitely often or infinitely often, and under SFA if they are enabled infinitely often they must also occur infinitely often. Using Formula 4.9, the UFA formula, we already have a way of expressing that some set of actions must occur infinitely often. To express that some set of actions is enabled only finitely often, we need only observe that if an action $\lambda$ is finitely often enabled in a path $\pi$, then $\pi$ has some suffix in which $\lambda$ is perpetually disabled. Combining these insights, we can conclude that for a strongly fair path, there is a suffix on which some subset of *Act* occurs infinitely often and the complement of that subset is perpetually disabled. Using this observation, we design Formula 4.11.

$$\neg(\langle \mathit{true}^\star \cdot q\rangle ( \bigvee_{F \subseteq \mathit{Act}} (\mu Y.(\langle \overline{r}\rangle Y \vee \nu X.\mathit{inf}_{\mathit{SFA}}(F))))) \tag{4.11}$$

Where $\mathit{inf}_{\mathit{SFA}}(F)$ is defined as follows. For this definition, we fix an arbitrary order on the actions in $F$ such that $\alpha_1$ is the first action, $\alpha_2$ the second, etc. Let $\alpha_n$ be the last element of $F$.

$$\begin{aligned}
\mathit{inf}_{\mathit{SFA}}(F) &= \mathit{exec}_{\mathit{SFA}}(F, n)\\
\mathit{exec}_{\mathit{SFA}}(F, 0) &= [\overline{F}]\mathit{ff} \wedge X\\
\mathit{exec}_{\mathit{SFA}}(F, k+1) &= \mu W_{k+1}.([\overline{F}]\mathit{ff} \wedge\\
&\qquad (\langle \overline{r}\rangle W_{k+1} \vee \langle \alpha_{k+1} \setminus r\rangle \mathit{exec}_{\mathit{SFA}}(F, k)))
\end{aligned}$$

There are a few notable differences with Formula 4.9. Firstly, we obviously quantify over all subsets $F$ of *Act*. It is those actions that we require to occur infinitely often with $\mathit{inf}_{\mathit{SFA}}$ instead of requiring this for all of *Act*. Additionally, the definition of $\mathit{exec}_{\mathit{SFA}}$ contains that the actions in $\overline{F}$ may not ever be enabled. This way we include the observation that *Act* is split into actions that occur infinitely often and actions that are finitely often enabled.

Secondly, we do not require the greatest fixed point to be satisfied immediately after the $q$, a finite sequence of non-$r$ steps is allowed to be taken first. The reason for this is that it may take some finite number of steps before all the actions in $\overline{F}$ are perpetually disabled.

Lastly, the separate option for reaching a deadlock state has been removed. This is because when $F = \emptyset$, $\mathit{inf}_{\mathit{SFA}}(F) = \mathit{exec}_{\mathit{SFA}}(\emptyset, 0) = [\mathit{true}]\mathit{ff} \wedge X$, which when taking the greatest fixed point of $X$ is satisfied in a deadlock state. Therefore, the option of reaching a deadlock is already built into this formula and we do not need to make a special exception for it.

We claim Formula 4.11 accurately captures exactly those states that satisfy global response under SFA.

**Theorem 4.15.** *A state satisfies Formula 4.11 if, and only if, it satisfies global response under strong fairness of actions.*

The proof of this theorem is included in Appendix A.6.

## Efficiency

The formula we present here is exponential in size to the number of actions that are treated fairly. More precisely, Formula 4.11 requires $\mathcal{O}(n \cdot 2^n)$ variables, where $n$ is the number of actions in *Act*: there are $2^n$ subsets, and for each we need a new variable for every action in the subset. This can be reduced to $\mathcal{O}(n \cdot 2^{n-1})$ if we note that $r$ must always be outside of $F$, since it will obviously not be taken infinitely often on an $r$-free path.

It should be noted that the alternation depth of the formula is still 3: $\mathit{inf}_{\mathit{SFA}}(F)$ consists of a sequence of $|F|$ least fixed points, which together only contribute to the alternation depth once. However, the quantification over all subsets of *Act* is still extremely expensive. Consequently, this formula is not usable in most practical applications. In Chapter 8, we will discuss a case study where even with only 18 actions, computing global response under strong fairness of actions took around 48 hours on our computer.

At the time of designing this formula, we had not encountered any other strong fairness formulae that accounted for all actions being treated fairly along a single path. However, we later came across [19, Chapter 6.3], which describes this exact approach of allowing every action to either be taken infinitely often or be perpetually disabled after some finite number of steps. This is a small indication that this may indeed be the best we can do when it comes to expressing this property.

In doing research for this thesis, we have not found definite proof that this property necessarily requires an exponential solution. However, it is not unreasonable to think that this may very well be the best we can do. One thing we considered was how this property could be expressed in a different logic, for example LTL. LTL formulae express properties over all paths of a model. Say that we wish to express the property that there is at least one fair path in the model – we ignore the $r$-free aspect for now since it is irrelevant to the underlying complexity of the problem – we would instead have to express the property that all paths are unfair. If this property is false, there is at least one fair path. That a single action $\alpha$ is treated unfairly is expressed in LTL as follows:

$$\Box \Diamond \mathit{enabled}\ \alpha \Rightarrow \Diamond \Box \neg(\mathit{occurs}\ \alpha)$$

This formula expresses that $\alpha$ is infinitely often enabled, but that there is a suffix of all paths on which $\alpha$ does not occur, hence $\alpha$ does not occur infinitely often. LTL is a logic based on atomic propositions, rather than actions like the $\mu$-calculus is, so we treat enabledness of $\alpha$ and occurrence of $\alpha$ as atomic propositions, for the sake of this example. A path is unfair if a single action is treated unfairly, so we could extend this formula to cover all unfair paths by simply quantifying over all actions.

$$\bigvee_{\alpha \in Act} \Box \Diamond \, \mathit{enabled} \; \alpha \Rightarrow \Diamond \Box \neg (\mathit{occurs} \; \alpha)$$

The size of this formula grows linearly with respect to the number of actions in *Act*. However, that does not mean computing whether a model satisfies this formula is a linear problem. In fact, establishing if a model satisfies an LTL formula is PSPACE-complete in the size of the formula [43]. There are ways of translating LTL formulae to the modal $\mu$-calculus, but once again doing model checking with those formulae is exponential in the size of the resulting $\mu$-calculus formula [13].

Of course, the above argument could also be applied to weak fairness of actions: that too requires a linear number of variables to express in LTL, yet we found a more efficient formula in Formula 4.4. So this observation on strong fairness is not proof that there cannot exist a more efficient formula, it merely indicates that finding this formula is not a trivial exercise and that translations from other logics are unlikely to help us.

*Remark.* As we have stated repeatedly, the formula presented as Formula 4.11 is far from elegant. In addition to being inefficient due to the quantification over subsets of *Act*, it is also presented in a pretty clunky manner. After we finished proving this formulae, we did think of a way to write it more elegantly. We were unable to finish to proof for this prettier formula within the time allocated for this thesis, so Formula 4.11 is the strong fairness formula we use throughout this thesis. However, we do include a brief discussion of the unproven formula in Appendix D. Note that this formula still has the quantification over subsets of *Act*, so it is still very inefficient. This appendix also has slight variations on the weak fairness and fair reachability formulae.

# Chapter 5

# Property Specification Patterns

We want to generalise our results from Chapter 4 to cover the other patterns from [15] as well. To this end, we first introduce the patterns here.

Dwyer, Avrunin and Corbett noted in [15] that one of the barriers for the adoption of formal methods and model checking in practical applications is that specifying the system requirements is often quite difficult and requires expert knowledge of the logic used. As a solution to this problem, they propose a specification pattern system, which we here call the *property specification patterns* (PSP). The system consists of a collection of high-level abstract descriptions of *behaviour* one may wish to express. These abstract descriptions are independent of any specific logic. These behaviours are then combined with a variety of *scopes*[1], which indicate where in a system execution (path) this behaviour must be satisfied. Formulae for these patterns are then provided in a few different logics, so that someone who wants to express a particular pattern in one of these logics need only copy the given formula and fill in the placeholder variables appropriately.

The patterns provided in [15] are based on a survey of over 500 specifications occurring in the literature, of which 92% are covered by the patterns given. In [41], Remenska did another survey of 178 specifications, and found that the original patterns by Dwyer et al. cover 70% of those specifications. She added extensions to the patterns, which resulted in the coverage being raised to 80%.

Dwyer et al. provide formulae for their patterns in Linear Temporal Logic (LTL), Computation Tree Logic (CTL), Graphical Interval Logic (GIL), Quantified Regular Expressions (QRE) and INCA Queries[2]. The CADP team, specifically Radu Mateescu and Mihaela Sighireanu, presented Action Computation Tree

[1]In [15], what we call behaviours are called patterns, which are then modified with pattern scopes. We instead use the term pattern to refer to the combination of a behaviour and a scope.

[2]Dwyer et al.'s formulae can be found at `https://web.archive.org/web/20230725153534/https://matthewbdwyer.github.io/psp/patterns.html`.

Logic (ACTL) formulae for the patterns[3]; Radu Mateescu also created formulae in regular alternation-free $\mu$-calculus[4], which is a subset of the modal $\mu$-calculus. Remenska extended Mateescu's $\mu$-calculus formulae to also cover her newly introduced patterns[5].

Dwyer et al.'s patterns are intentionally quite generic, and can be instantiated in a variety of contexts. In particular, the patterns do not specify whether the observable information is state-information or event-information. Since we are using the modal $\mu$-calculus, which uses action labels (i.e. events) instead of atomic propositions (i.e. state information), we interpret all the patterns over actions in this thesis. This is in line with the formulae from Mateescu and Remenska.

We describe the different scopes and behaviours here. We in most cases use $q$ and $r$ to represent behaviour actions. As a mnemonic, one can use "question" and "response" although what actions are filled in for these is up to the user, it need not be actions indicating a question or a response. For scope deliminators, we use the variables $a$ and $b$, which can be easily remembered as "after" and "before".

## 5.1 Behaviours

From [15], we have the following behaviours:

- **Absence**: the $r$-action does not occur in the scope.
- **Existence**: the $r$-action occurs in the scope.
- **Bounded Existence**: the $r$-action must occur at most/at least/exactly $k$ times in the scope.
- **Universality**: the $r$-action occurs throughout the scope.
- **Precedence**: the $r$-action must always be preceded by the $q$-action within a scope. Note that the $q$ must fall within the same scope as the $r$.
- **Response**: the $q$-action must always be followed by the $r$-action within a scope. Once again note the $r$ must also occur within the same scope.

[3]Mateescu and Sighireanu's ACTL formulae can be found at `https://web.archive.org/web/20230725154156/https://cadp.inria.fr/resources/evaluator/actl.html`.

[4]Mateescu's regular altnernation-free $\mu$-calculus formulae can be found at `https://web.archive.org/web/20230725153928/https://cadp.inria.fr/resources/evaluator/rafmc.html`.

[5]Sadly, the HTML page where Remenska's patterns were publicly shared is unreachable and has not been archived. Her Github page contains an HTML file which describes several patterns, `http://web.archive.org/web/20230901054952/https://raw.githubusercontent.com/remenska/remenska.github.io/master/patterns/index.html`. This is not an ideal source, hence we will primarily refer to what is discussed in [41] explicitly.

- **Chain-Precedence**: the sequence $r_0, r_1, \ldots r_n$ must always be preceded by the sequence $q_0, q_1, \ldots, q_m$ within a scope. It should be noted this behaviour is on specific sequences, so if $r_1$ occurs before $r_0$ instead of after, for instance, the behaviour does not apply. There may be other actions between the actions that make up the sequences.

- **Chain-Response**: the sequence $q_0, q_1, \ldots, q_m$ must always be followed by the sequence $r_0, r_1, \ldots, r_n$ within a scope. Here too, the order in the sequences is relevant and there may be actions in-between.

The first four are known as occurrence patterns, the latter four as order patterns. There is a fifth order pattern that is not in [15] but is mentioned on Dwyer's Github page: the **Constrained-Chain**. This is a variant of the Chain-Precedence and Chain-Response, where in addition to requiring certain sequences of actions precede/follow other sequences, certain actions may not occur in between parts of these sequences. This behaviour is the least clearly specified. An example of the behaviour is given by Dwyer as: the $p$-action must be followed by an $s$-action and subsequently a $t$ action without a $z$-action occurring between the $s$ and the $t$.

From [41] we have the following extensions:

- **Always-Enabled**: the $r$-action must always be enabled within a scope.

- **Precedence-Variant**: the $r$-action must always be preceded by the $q$-action within a scope, and $q$ must eventually occur within that scope.

- **Response-Variant**: the $q$ action must always be followed by the $r$-action within a scope, and $q$ must occur within that scope.

For both existence and (variant and non-variant) response, Remenska also provides an "under fairness" alternative. It is not explicitly stated exactly which fairness assumption is used. For global existence and response, it seems to correspond to our fair reachability of actions assumption. However, for the other patterns the formulae presented do not correspond to that fairness assumption. A bit more discussion of her formulae for fairness is included in Appendix E, where we compare her formulae to the formulae we propose for fair reachability of actions.

## 5.2 Scopes

From [15], we have the following scopes, which we visualise in Figure 5.1:

- **Global**: along the full path.

- **Before**: before the first occurrence of the $b$-action. If the $b$ never occurs, the behaviour need not hold anywhere.

- **After-First**: after the first occurrence of the $a$-action. This is called "after" in [15]. If the $a$ never occurs, the behaviour need not hold anywhere.
- **Between-First**: between any occurrence of $a$ and $b$. If there is a sequence of $a$-actions before the subsequent $b$, the behaviour needs to hold from the first of those $a$'s onward. If an $a$ is not eventually followed by a $b$, the behaviour need not hold after that $a$. This is called "between" in [15].
- **Until-First**: after an $a$ until the next $b$. If there is a sequence of $a$-actions before the next $b$, the behaviour needs to hold from the first of those $a$'s onward. If the $a$ is not followed by a $b$, the behaviour still needs to hold after that $a$. This is called "until" in [15].

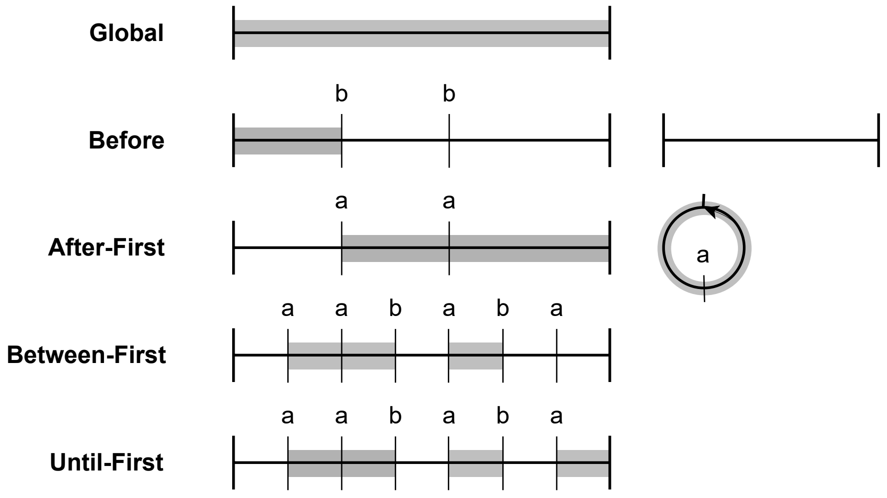


Figure 5.1: The scopes from Dywer et al. visualised.

Remenska proposed the following variants of some of these scopes, we visualise these in Figure 5.2.

- **Before-Variant**: like the before scope but if the $b$ never occurs, the behaviour needs to hold for the entire path.
- **After-Last**: the behaviour only needs to hold after the last $a$-action. This can be seen as the scope resetting every time an $a$ occurs. Matching this new interpretation of after, we also get **Between-Last** and **Until-Last**, in which if there is a sequence of $a$'s until the subsequent $b$, the behaviour only needs to hold from the last of those $a$'s.

Regarding the $\star$-last scopes, one may wonder what happens if we have an infinite path along which the $a$ occurs infinitely often. In such cases, there is no "last". Indeed, in this case the behaviour simply need not hold anywhere: the infinite $a$'s keep resetting the scope.

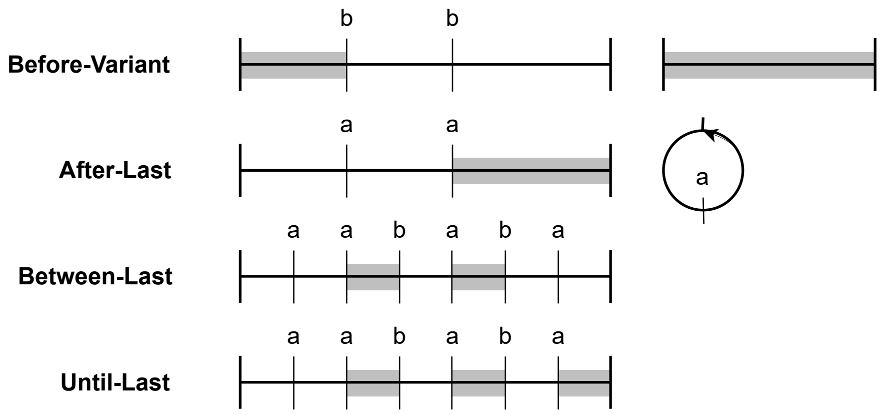


Figure 5.2: The scopes from Remenska visualised.

Given the introduction of the ⋆-first and ⋆-last versions of after, between and until, we ourselves add a third variant, visualised in Figure 5.3:

- **After-Any**: the behaviour must hold after any $a$ along the path. Correspondingly, we get **Between-Any** and **Until-Any** which say that if there is a sequence of $a$'s before the next $b$, the behaviour must hold from any of those $a$'s onward.

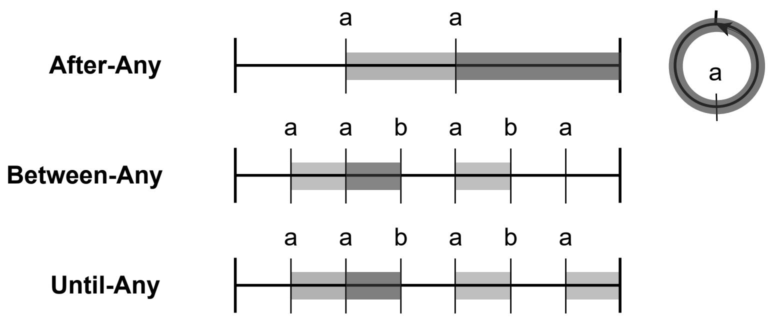


Figure 5.3: The scopes newly added here visualised.

## 5.3 Notes on the Patterns

There are a few observations we wish to make on the interpretation and usefulness of some patterns. These are mainly included for the interested reader, and are not required to use the patterns.

Firstly, we observe that the combination of behaviour and scope can lead to some confusion. We found this in particular with global existence: does it mean that the $r$-action must occur at least once on every path, or that the $r$-action must eventually occur after every state on the path? In other words, must the occurrence of $r$ be in the scope global, or must every state in the global scope satisfy that $r$ eventually occurs? This question is answered if we carefully read the description of existence: the $r$-action must occur in the scope, and the scope is the whole path. Therefore, global existence means that $r$ must occur at least once on every path. Should one wish to express that from every state on every path, $r$ eventually occurs, this an be done by combining existence after-any and global existence, by filling in $r$ for $a$. This results in the following property: $r$ must occur once on every path, and after every occurrence of $r$ there must eventually be an occurrence of $r$. Together this states that $r$ occurs infinitely often. A more direct way to express that on all paths $r$ must occur infinitely often is $\nu X.(\mu Y.([r]X \wedge [\overline{r}]\,Y \wedge \langle true \rangle tt))$, which is equivalent to the formula you get if you combine the global existence $r$ with existence of $r$ after-any $r$ formulae we present in Chapter 6. The direct formula is more elegant, but since it is also expressible without adding a new pattern we leave it as a combination of existence patterns.

Secondly, we note that not all behaviours are equally useful for us. For example, the universality behaviour is not very useful in an event-based logic such as the modal $\mu$-calculus. Global universality would mean that only the $r$-action is allowed to occur on the whole path, and there do not seem to be very many practical situations where such a specification would need to be written. It makes more sense in an state-based logic, where you may specify that some atomic proposition must be true consistently. This observation is why Remenska introduced an alternate interpretation of the universality pattern, which depends on data parameters in action labels. We do not consider action labels that contain data parameters, so we do not provide this behaviour. The always-enabled behaviour is another variant of the universality behaviour, that makes more sense in an event-based logic.

Finally, we said that the scopes presented by Dwyer et al. are all of the $\star$-first variety, but this is not quite accurate. Take the CTL formulae on their Github page, for example. Absence of $r$ after $a$ is given as $\mathbf{AG}(a \Rightarrow \mathbf{AG}(\neg r))$, which corresponds to after-any. Existence of $r$ after $a$ on the other hand, is given as $\mathbf{A}[\neg a\ \mathbf{W}\ (a\ \wedge\ \mathbf{AF}(r))]$ which corresponds to after-first. Of course, it is trivial to see that in the case of the absence behaviour, after-any and after-first are the same: if $r$ never occurs after the first $a$, it never occurs after any $a$. This is not the case for existence however: if the $r$ occurs after the first $a$ but not after the second, then existence after-first holds but existence after-any does not. Hence why we present the scopes of Dwyer et al. as the $\star$-first variant, since where it matters this approach is taken.

# Chapter 6

# Formulae for Patterns

In this chapter, we will consider the remaining property specification patterns. As we will discuss in the subsequent section, fairness is only relevant for a small subset of these. We only give formulae for those patterns where it makes sense to include fairness. For $\mu$-calculus formulae without fairness, we point to the translations by Mateescu and Remenska, as linked in the previous chapter.

In Section 6.2, we will generalise the formulae from Chapter 4 so that they can easily be adapted to other behaviours and scopes. We will give a brief overview of the modifications required for the main patterns where fairness is relevant. We do not discuss every pattern, since some are generalisations of each other (i.e., chain response generalises response). In the remaining parts of this chapter, we will discuss these modifications in more detail. In Appendix E, we compare our formulae for fair reachability of actions to Remenska's formulae for fairness.

Recall that we are still considering only fairness assumptions over actions. Hence, all our definitions in Chapter 2 that concern tasks apply to actions here. Similar to Chapter 4, we assume progress in all our arguments.

## 6.1 Relevance of Fairness

In Chapter 3, we stated that fairness assumptions only limit what infinite paths are considered valid. Consequently, regardless of what fairness assumptions are made, all finite paths are fair. Additionally, recall from Section 3.3 that weak fairness, strong fairness and fair reachability are all feasible: any finite partial path can be extended to a fair complete path. For some properties, it turns out feasible fairness assumptions have no effect because violations of this property can always be observed within finitely many steps. Take for example the absence behaviour: a violation of $r$ not being allowed will always be a path on which $r$ occurs. Such a path must have a finite prefix up to the first occurrence of $r$, this is a partial path. And since, under a feasible fairness assumption, we know for certain such a partial path can be extended fair and complete path where $r$ still occurs, we can can conclude that there must exist fair violating paths.

This same argument can be applied to any other property where violations can always be observed in finitely many steps: if there exists a violating path for the property without making a fairness assumption, then there exists a finite partial path violating the property, and thus there exists a fair complete path violating the property under a feasible fairness assumption. This is why fairness assumptions are usually only made for liveness properties ("something good eventually happens", violating paths may be infinite) and not safety properties ("something bad never happens", violations are always observable within finitely many steps).

So for many of the behaviours, regardless of scope, feasible fairness is irrelevant. We have already argued this is the case for absence. The same goes for universality of $r$: a violation comes in the form of any action other than $r$ occurring in the appropriate scope, which can be observed in a finite partial path. Similarly, always-enabled is observable in the finitely many steps it takes for $r$ to become disabled. It is less immediately obvious for precedence, yet there too fairness does not contribute. Consider that a violation of the property that $r$ must be preceded by $q$ will always be an occurrence of $r$ when $q$ has not occurred yet. This occurrence of $r$ will be observed within finitely many steps. Finally, consider bounded existence. If the action $r$ must occur at most $k$ times, then any violating path contains $r$ at least $k+1$ times so we have a finite partial path to the $k+1$'th occurrence. For at least $k$ times and exactly $k$ times, fairness does play a role.

Next, we consider what scopes are relevant with fairness. In its standard form, the before scope is not affected by fairness: in the description of the scopes in [15], it is made explicit that if some behaviour needs to be satisfied before the $b$ action occurs, and $b$ never occurs, then the property is trivially true even if the conditions of the behaviour are not satisfied. For example, existence of $r$ before $b$ is true in a system where neither $r$ nor $b$ is ever enabled. With this interpretation, the only violating paths that exist all have a finite prefix that ends with the $b$-action occurring. Remenska noted that this was not the only reasonable interpretation of the concept of "before", and proposed before-variant, where if $b$ never occurs, the behaviour must be satisfied on the whole path. Here, fairness is relevant. There is a similar situation with the between scope. Once again, [15] makes it clear that if some behaviour must be satisfied between occurrences of the $a$ and $b$ actions, then if an $a$-action occurs which is never followed by a $b$ the pattern need not be satisfied after that $a$. This means the behaviour only needs to be satisfied in a finite stretch of any path and hence fairness is irrelevant. We can here make the same variant as we did with before-variant, stating that if no $b$ occurs after an $a$ the behaviour still needs to hold. However, this exactly describes the until scope, so we simply give formulae for until and do not discuss the between scope further.

We do not cover all the behaviours for which fairness is relevant in this thesis, primarily due to time constraints. We discuss response and existence, these two are the most commonly occurring behaviours of those for which fairness is relevant. This means we do not discuss bounded existence for at least or exactly, chain-response, constrained chain, precedence-variant or response-variant. This is less bad than it may seem. Note that bounded existence is a variant of existence, which we do cover. Similarly, chain response and constrained chain (response) are variants of response. As for precedence-variant and response-variant, these are basically combinations of existence and precedence/response respectively. Hence, while we do not cover all behaviours, we do cover the most important ones. An overview of what we do and do not cover is given in Table 6.1.

| Behaviour | Relevant | Covered | Scope | Relevant | Covered |
|---|---|---|---|---|---|
| Absence | × | × | Global | ✓ | ✓ |
| Existence | ✓ | ✓ | Before | × | × |
| Existence (Most) | × | × | After-First | ✓ | ✓ |
| Existence (Least) | ✓ | × | Between-First | × | × |
| Existence (Exactly) | ✓ | × | Until-First | ✓ | ✓ |
| Universality | × | × | Before-Variant | ✓ | ✓ |
| Precedence | × | × | After-Last | ✓ | ✓ |
| Response | ✓ | ✓ | Between-Last | × | × |
| Chain-Precedence | × | × | Until-Last | ✓ | ✓ |
| Chain-Response | ✓ | × | After-Any | ✓ | ✓ |
| Always-Enabled | × | × | Between-Any | × | × |
| Precedence-Variant | ✓ | × | Until-Any | ✓ | ✓ |
| Response-Variant | ✓ | × | | | |
| Constrained-Chain | ✓ | × | | | |

Table 6.1: Overview of the different behaviours and scopes, whether (feasible) fairness is relevant for them, and whether we cover them.

*Remark.* We previously discussed that unconditional fairness is not feasible. Hence, unconditional fairness does have an impact on properties even when all violations of that property can be observed within finitely many steps. The arguments given above for why certain behaviours and scopes are not affected by fairness therefore do not apply to UFA. Consider for example Figure 3.2, the LTS we previously showed in which, if we assume progress and UFA, there are no valid paths at all. Any property will vacuously hold under UFA on this LTS, even global absence of $r$, even though $r$ is the only enabled action in the initial state. We can think of very few scenarios where this would be the intended consequence of making a fairness assumption. This is the primary reason we do not cover unconditional fairness much in this thesis, and we do not present UFA versions of every PSP pattern. For those patterns we do cover, our SFA formulae can be adapted to UFA formulae by instantiating $F$ as $Act$ and explicitly adding the possibility of ending in a deadlock, should one need UFA variants.

## 6.2 Formula Structures

In this section, we present pattern-agnostic "base" versions of the non-violate formulae from Chapter 4. We focus on the non-violate style formulae for two reasons: firstly, we have formulae in this style for all three fairness assumptions we discuss here; and secondly, because formulae in this style clearly show the shape of fair, violating paths. In Appendix E, we also give a base version of the precondition FRA formula to facilitate comparison with Remenska's formulae. We do not further discuss the precondition approach in this section.

Recall that the non-violate formulae in Chapter 4 all take the form "there does not exist a fair path on which a $q$ is not followed by an $r$". When we want to make formulae for other patterns in this same style, we need to define exactly what makes a path a violating path for that pattern. We can then specify that there is no fair path that has those characteristics. From the global response formulae we already know how to include that a violating path must have a specific prefix: it comes at the very start of the formula in the place where global response has $\mathit{true}^\star \cdot q$. Additionally, we see in the global response formulae how to say some action(s) may not occur anymore after the prefix: this is where global response excludes the $r$-action from some box and diamond operators. Examining the patterns we have decided to cover, it turns out these two things are enough to cover many patterns. For the remainder, we need a third modification: in the case of the before-variant and until scopes, we need to include that if the scope ends before the behaviour has been satisfied, we have found a violating path. Since scopes are delimited by actions, a scope can end when the action that ends the scope is enabled.

Changing these three parts of the formulae from Chapter 4 allows us to create formulae for all the patterns we cover. Instead of presenting those, rather repetitive, formulae separately, we present base versions that include placeholder variables and then provide an overview of how these variables must be filled in to represent different patterns. We use the following placeholder variables:

- $\delta_1$ is a regular formula which gives a prefix every violating path should have. This might, for example, be $\mathit{true}^\star$ if a property needs to hold at every point in a path. If a property only needs to hold from the initial state, this will be $\varepsilon$. This part of the prefix reflects the scope.

- $\delta_2$ is another regular formula, and also part of the prefix, but we separate it from $\delta_1$ because it reflects behaviour rather than scope. For instance, for all response properties we will have $\delta_2 = ?^\star \cdot q$, where the ? reflects which actions are allowed to occur between the $\delta_1$ and $q$.

- $\delta_3$ is an action formula that, if it ever becomes enabled, means a violating path exists. For example, if a particular behaviour is required to hold before $b$, and $b$ becomes enabled without the behaviour being satisfied, then a violating path can be constructed by taking this $b$-transition. This variable reflects part of the scope.

- $\delta_4$ is an action formula capturing actions that are not allowed to occur after the prefix characterised by $\delta_1 \cdot \delta_2$. This placeholder is for actions excluded based on the behaviour. For example, if the property requires $r$ to occur after the prefix then a violating path may not include the action $r$ after $\delta_1 \cdot \delta_1$.

- $\delta_5$ is similar to $\delta_4$: an action formula that represents actions that are not allowed to occur after $\delta_1 \cdot \delta_2$. We use $\delta_5$ for actions that are disallowed by the scope. For example, if we wish to specify something occurs after the last $a$, then no more $a$'s are allowed to occur after the prefix.

Using these variables, we want our non-violate formulae to express the following: there does not exist a path $\pi$ that meets all of the following requirements:

1. $\pi$ is complete and fair according to our chosen fairness assumption, and

2. $\pi$ has a prefix matching $\delta_1 \cdot \delta_2$, and

3. on $\pi$, there is no occurrence of any action in $\delta_4 \cup \delta_5$ after the prefix matching $\delta_1 \cdot \delta_2$ and before the occurrence of any action in $\delta_3$.

The word "before" should be interpreted like the before-variant scope: if there is no occurrence of $\delta_3$ after the $\delta_1 \cdot \delta_2$-prefix, there may never be an occurrence of the actions in $\delta_4 \cup \delta_5$ after that prefix. We visualise these requirements in Figure 6.1.

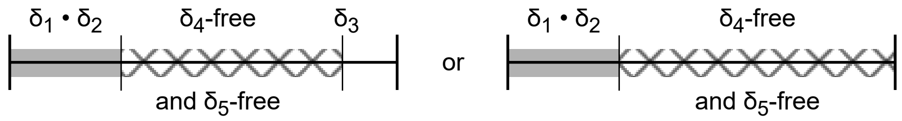


Figure 6.1: Visualisations of the shape of a violating path. The prefix matching $\delta_1 \cdot \delta_2$ is visualised in solid grey, the part that should be $\delta_4 \cup \delta_5$-free is visualised with crosshatching. The path should also be complete and fair.

The absence of a violating path according to the three requirements is captured for WFA by Formula 6.1:

$$\begin{aligned}\neg(\langle\delta_1\cdot\delta_2\rangle\nu X.(\bigwedge_{\lambda\in Act}(\langle\lambda\rangle tt \Rightarrow (\\ \mu Y.(\langle\delta_3\rangle tt \vee ([\lambda]ff \wedge X)\vee\\ \langle\lambda\setminus(\delta_4\cup\delta_5)\rangle X \vee \langle\overline{\lambda\cup\delta_4\cup\delta_5}\rangle Y)))))\end{aligned} \tag{6.1}$$

Compared to the global response formula, we have swapped $true^\star\cdot q$ for $\delta_1\cdot\delta_2$, and we have replaced the explicit references to $r$ with $\delta_4\cup\delta_5$. Finally, we added that if $\delta_3$ becomes enabled after $\delta_1\cdot\delta_2$, then a violating path has been found. This works because the formula already enforces that after $\delta_1\cdot\delta_2$, no actions in $\delta_4$ or $\delta_5$ may have occurred. So if $\delta_3$ becomes enabled, a partial path exists where after $\delta_1\cdot\delta_2$, $\delta_3$ occurs without there first being an occurrence of any of the actions in $\delta_4$ or $\delta_5$. Since WFA is feasible, this partial path can be extended to a complete fair path that meets the requirements to be a violating path.

For SFA, we use Formula 6.2:

$$\neg(\langle\delta_1\cdot\delta_2\rangle(\bigvee_{F\subseteq Act}(\mu Y.(\langle\overline{\delta_4\cup\delta_5}\rangle Y \vee \langle\delta_3\rangle tt \vee \nu X.inf(F))))) \tag{6.2}$$

Where $inf(F)$ is defined as follows. We fix an arbitrary order on the actions in $F$ such that $\alpha_1$ is the first action, $\alpha_2$ the second, etc. Let $n=|F|$.

$$\begin{aligned}inf(F) &= exec(F,n)\\ exec(F,0) &= [\overline{F}]ff \wedge X\\ exec(F,k+1) &= \mu W_{k+1}.([\overline{F}]ff\wedge\\ &\quad (\langle\overline{\delta_4\cup\delta_5}\rangle W_{k+1} \vee \langle\alpha_{k+1}\setminus(\delta_4\cup\delta_5)\rangle exec(F,k)))\end{aligned}$$

For the SFA formula, $\delta_3$ is not incorporated in the greatest fixpoint, which is how we do it for WFA. Instead, recall that the SFA formula already allows there to be a finite sequence of actions before the greatest fixpoint needs to be satisfied. We add $\delta_3$ there: if we can reach a state where $\delta_3$ is enabled within finitely many non-$\delta_4$ and non-$\delta_5$ steps, then a violating path exists. Once again, feasibility is important here: if a partial path exists where $\delta_3$ occurs after $\delta_1\cdot\delta_2$ without any actions from $\delta_4$ or $\delta_5$ occurring in-between, then this partial path can be extended into a complete and fair violating path.

We have previously remarked in Section 4.6 that there is no point in allowing the actions that should not occur to be in $F$. Hence, to slightly reduce the complexity of the formula in practice we could exclude all actions in $\delta_4$ and $\delta_5$ from being in $F$ without affecting correctness of the formula. In that case, it is important to note that $\delta_4$ and $\delta_5$ are both still subsets of $\overline{F}$.

Finally, for FRA we have Formula 6.3:

$$
\begin{aligned}
\neg(\langle \delta_1 \cdot \delta_2 \rangle \nu X.( \bigwedge_{\lambda \in Act} (\langle \mathit{true}^\star \cdot \lambda \rangle \mathit{tt} \Rightarrow ( \\
\mu Y.(\langle \delta_3 \rangle \mathit{tt} \vee ([\mathit{true}^\star \cdot \lambda]\mathit{ff} \wedge X) \vee \\
\langle \lambda \setminus (\delta_4 \cup \delta_5) \rangle X \vee \langle \overline{\lambda \cup \delta_4 \cup \delta_5} \rangle Y)))))
\end{aligned}
\tag{6.3}
$$

There is nought to be said about the FRA formula that we have not already said about the WFA formula: fairness applies to reachable actions instead of enabled actions, but otherwise the two formulae are identical.

For global response, the variables are filled in as: $\delta_1 = \varepsilon, \delta_2 = \mathit{true}^\star \cdot q, \delta_4 = r$ and $\delta_3 = \delta_5 = \mathit{false}$. Indeed, filling in these values will result in the formulae from Chapter 4. To observe this, we need to keep in mind that $\langle \varepsilon \rangle \phi = \phi$, so $\langle \varepsilon \cdot \mathit{true}^\star \cdot q \rangle = \langle \mathit{true}^\star \cdot q \rangle$. Additionally, $\delta_3 = \mathit{false}$ does not contribute to the semantics of the formulae: $\langle \mathit{false} \rangle \mathit{tt} = \bigvee_{\alpha \in \emptyset} \langle \alpha \rangle \mathit{tt}$, which is equivalent to false. Thus, adding $\langle \mathit{false} \rangle \mathit{tt}$ to a disjunction does not affect the semantics of the formula.

We present how the placeholders should be filled in for the patterns we have chosen to cover in Table 6.2. These assignments work with Formula 6.1, Formula 6.2 and Formula 6.3.

In the table we can observe that $\delta_1$, $\delta_3$ and $\delta_5$ are identical for existence and response. This is because these are only dependent on scope; they are not affected by behaviour. Even though both $\delta_2$ and $\delta_4$ are dependent on behaviour, $\delta_4$ is still the same for both existence and response as both behaviours have violating paths when $r$ does not occur after after the prefix. Consequently, existence and response behaviours only differ in their assignment to $\delta_2$, which is the $q$ action and what actions are allowed to precede the $q$.

| Behaviour | Scope | | $\delta_1$ | $\delta_2$ | $\delta_3$ | $\delta_4$ | $\delta_5$ |
|---|---|---|---|---|---|---|---|
| None | | | $\mathit{true}^\star$ | $\varepsilon$ | $\mathit{false}$ | $\mathit{false}$ | $\mathit{false}$ |
| Response | Global | | $\varepsilon$ | $\mathit{true}^\star \cdot q$ | $\mathit{false}$ | $r$ | $\mathit{false}$ |
| | Before-Var | | $\varepsilon$ | $\overline{b}^\star \cdot q$ | $b$ | $r$ | $\mathit{false}$ |
| | After | Any | $\mathit{true}^\star \cdot a$ | $\mathit{true}^\star \cdot q$ | $\mathit{false}$ | $r$ | $\mathit{false}$ |
| | | First | $\overline{a}^\star \cdot a$ | $\mathit{true}^\star \cdot q$ | $\mathit{false}$ | $r$ | $\mathit{false}$ |
| | | Last | $\mathit{true}^\star \cdot a$ | $\overline{a}^\star \cdot q$ | $\mathit{false}$ | $r$ | $a$ |
| | Until | Any | $\mathit{true}^\star \cdot a$ | $\overline{b}^\star \cdot q$ | $b$ | $r$ | $\mathit{false}$ |
| | | First | $(\overline{a}^\star \cdot a) +$ $(\mathit{true}^\star \cdot b \cdot \overline{a}^\star \cdot a)$ | $\overline{b}^\star \cdot q$ | $b$ | $r$ | $\mathit{false}$ |
| | | Last | $\mathit{true}^\star \cdot a$ | $\overline{a \cup b}^\star \cdot q$ | $b$ | $r$ | $a$ |

| Behaviour | Scope | | $\delta_1$ | $\delta_2$ | $\delta_3$ | $\delta_4$ | $\delta_5$ |
|---|---|---|---|---|---|---|---|
| Existence | Global | | $\varepsilon$ | $\varepsilon$ | *false* | $r$ | *false* |
| | Before-Var | | $\varepsilon$ | $\varepsilon$ | $b$ | $r$ | *false* |
| | After | Any | $\mathit{true}^\star \cdot a$ | $\varepsilon$ | *false* | $r$ | *false* |
| | | First | $\overline{a}^\star \cdot a$ | $\varepsilon$ | *false* | $r$ | *false* |
| | | Last | $\mathit{true}^\star \cdot a$ | $\varepsilon$ | *false* | $r$ | $a$ |
| | Until | Any | $\mathit{true}^\star \cdot a$ | $\varepsilon$ | $b$ | $r$ | *false* |
| | | First | $(\overline{a}^\star \cdot a) +$ $(\mathit{true}^\star \cdot b \cdot \overline{a}^\star \cdot a)$ | $\varepsilon$ | $b$ | $r$ | *false* |
| | | Last | $\mathit{true}^\star \cdot a$ | $\varepsilon$ | $b$ | $r$ | $a$ |

Table 6.2: The different assignments to the variables in the base formulae to represents the various properties and scopes. Before-Var is Before-Variant.

## 6.3 Correctness Claims

For each of the base formulae, we claim they exactly characterise that there does not exist a violating path that satisfies the chosen fairness assumption.

**Theorem 6.1.** *A state $s \in \mathcal{S}$ satisfies Formula 6.1 if, and only if, it does not admit a path $\pi$ meeting the following requirements:*

1. *$\pi$ is complete and satisfies weak fairness of actions, and*
2. *there is a prefix of $\pi$ that matches $\delta_1 \cdot \delta_2$, and*
3. *there is no occurrence of any action in $\delta_4$ or $\delta_5$ in $\pi$ after $\delta_1 \cdot \delta_2$ and before the first occurrence of an action in $\delta_3$.*

This theorem is proven in Appendix B.1.

**Theorem 6.2.** *A state $s \in \mathcal{S}$ satisfies formula Formula 6.2 if, and only if, it does not admit a path $\pi$ meeting the following requirements:*

1. *$\pi$ is complete and satisfies strong fairness of actions, and*
2. *there is a prefix of $\pi$ that matches $\delta_1 \cdot \delta_2$, and*
3. *there is no occurrence of any action in $\delta_4$ or $\delta_5$ in $\pi$ after $\delta_1 \cdot \delta_2$ and before the first occurrence of any action in $\delta_3$.*

The proof for this theorem is given in Appendix B.2.

**Theorem 6.3.** *A state $s \in \mathcal{S}$ satisfies Formula 6.3 if, and only if, it does not admit a path $\pi$ meeting the following requirements:*

1. *$\pi$ is complete and satisfies fair reachability of actions, and*
2. *there is a prefix of $\pi$ that matches $\delta_1 \cdot \delta_2$, and*
3. *there is no occurrence of any action in $\delta_4$ or $\delta_5$ in $\pi$ after $\delta_1 \cdot \delta_2$ and before the first occurrence of an action in $\delta_3$.*

Finally, the FRA proof is given in Appendix B.3.

## 6.4 Formula Details

In this section, we discuss the contents of Table 6.2 in more detail. We do not have full proofs for every variant, but we use this section to provide intuitive arguments as to why the choices we made in Table 6.2 are correct. Detailed comparisons with Remenska's formulae are provided in Appendix E.

We already discussed global response under all three fairness assumptions in detail in Chapter 4. We here explain the choices made for the other patterns we covered in Table 6.2. We will primarily focus on the response behaviour, since there we can cover both the impact of different scopes as well as our choice in what actions may come before $q$ in $\delta_2$. Once all the response formulae are discussed, most of the existence formulae need not be discussed anymore since they are very similar. Hence, we only make a few brief remarks on existence.

### 6.4.1 Response Before-Variant

For this combination of behaviour and scope, we need to express that every $q$ action that occurs before the first $b$ must eventually be followed by an $r$. This $r$ must also come before the first $b$. Since this is the before-variant scope, if the $b$ never occurs the response behaviour must be satisfied by the entire path.

The following is a formula expressing this property without fairness, which is due to Remenska:

$$[\overline{b}^\star \cdot q]\mu Y.(\langle \mathit{true}\rangle \mathit{tt} \wedge [b]\mathit{ff} \wedge [\overline{r}]\,Y) \tag{6.4}$$

Recall the global response without fairness formula from Chapter 4:

$$[\mathit{true}^\star \cdot q]\mu Y.(\langle \mathit{true}\rangle \mathit{tt} \wedge [\overline{r}]\,Y)$$

We can observe two differences between the two formulae:

1. Instead of considering every $q$ that occurs, Remenska only considers those that occur before any $b$ has occurred.

2. When defining the finite path until $r$ is the only enabled action, Remenska adds that $b$ may never be enabled along this path. Consider that if the $b$ becomes enabled before the $r$ has been taken, there would exist a path where the $b$ is done before the $r$ that responds to the $q$.

We can see these modifications directly reflected in Table 6.2 when looking at the global response and response before-variant rows. The first change is reflected by replacing $\mathit{true}^\star \cdot q$ with $\overline{b}^\star \cdot q$ in $\delta_2$. For the second change, we add $\delta_3 = b$.

### 6.4.2 Response After

We consider the after-any, after-first and after-last variants separately.

#### Any

For response after-any, we wish to express that after any $a$-action, any subsequent $q$ must eventually be followed by an $r$. Without fairness assumptions, this would be expressed as follows:

$$[\mathit{true}^\star \cdot a \cdot \mathit{true}^\star \cdot q]\mu Y.(\langle \mathit{true}\rangle \mathit{tt} \wedge [\overline{r}]\, Y) \tag{6.5}$$

Which expresses that whenever an $a$ is done, any subsequent occurrence of $q$ must lead to a state from which one always in finitely many steps reaches a state where only $r$ is enabled, without encountering a deadlock. The difference between Formula 6.5 and the global response without fairness formula (Formula 4.1) is that we have stuck $[\mathit{true}^\star \cdot a]$ in front to indicate it only has to hold after an $a$-action. This change can also be seen in Table 6.2, where the only difference between global response and response after-any is $\delta_1 = \mathit{true}^\star \cdot a$.

#### First

Specifying that $r$ must respond to $q$ after the first occurrence of $a$, is done in a manner similar to after-any. Instead of using $\mathit{true}^\star \cdot a$ we instead use $\overline{a}^\star \cdot a$ to specify it must be the first $a$.

It should be noted however that for the response pattern, there is no difference between the after-any scope and the after-first scope semantically. We argue this informally: after-any always subsumes after-first because the first occurrence of $a$ is *an* occurrence of $a$. In the case of the response behaviour, after-first also subsumes after-any. This is because any $q$ that comes after an arbitrary $a$ must also come after the *first* $a$ on a path, so if we know that every $q$ that comes after the first $a$ is followed by an $r$, then we can conclude this is also the case for every $q$ that comes after an arbitrary $a$.

The observation above only applies to the response behaviour. For existence, after-any is not equivalent to after-first. Consider a path on which first an $a$ occurs, then an $r$, and then an $a$ again which brings us to a deadlock state. On this path, there is an $r$ after the first $a$, but it is not the case that there is an $r$ after any $a$. Hence, separating the two scopes is valuable. The change from $\mathit{true}^\star \cdot a$ to $\overline{a}^\star \cdot a$ works for the existence behaviour as well, as is shown in Table 6.2.

**Last**

While there is no difference between after-any and after-first for the response behaviour, there is a difference with after-last. Consider a complete path consisting of the sequence of actions $aqa$. Since there is a $q$ after some $a$ which is never followed by an $r$, response after-any obviously does not hold. However, since after the last $a$ there are no more $q$-actions, response after-last does hold.

For the after-last scope, a violating path must have the following shape: there is at some point an $a$, which is followed by a $q$, after which there is no occurrence of $r$. That part is standard for response after-$\star$. The key with the after-last scope is that after this $a$, there may not be any more occurrences of $a$. If there were, it would not be the last $a$. Comparing the response after-any and response after-last variable assignments in Table 6.2, the observation above explains all differences:

- For $\delta_2$, there is no longer allowed to be an $a$ before the $q$, hence $\mathit{true}^\star \cdot q$ is replaced by $\overline{a}^\star \cdot q$[1].
- $\delta_5$ has gone from *false* to $a$, because just like no $r$ is allowed to occur after the $q$ on a violating path, the action $a$ is also not allowed to occur.

We described in Chapter 5 that the after-last scope means the behaviour need not be satisfied if there is no last $a$. Indeed, because $\delta_5 = a$, if after every $a$ and subsequent $q$, another $a$ is guaranteed to occur, we cannot construct a violating path even if $r$ never occurs.

## 6.4.3 Response Until

For response until, we wish to express that after any/the first/the last $a$ before the next $b$, any $q$-action must be followed by an $r$. The matching $r$ must also fall

[1]Technically this change is not needed: if there is an $a$ after $\delta_1$ but before $q$, then that $a$ becomes the last $a$. If the $q$ is then followed by an $a$-free and $r$-free path, we still have a violating path. So $\mathit{true}^\star \cdot q$ could still be used for $\delta_2$. However, we prefer to have the $a$ in the formula be the last $a$ on the path we are describing, so that the correspondence between formula and pattern is clear.

before the next $b$. If there is an $a$ without subsequent $b$, the response condition must still be satisfied after that $a$.

The until-$\star$ scope is a combination of the before-variant and after-$\star$ scopes. For the most part, we simply need to combine the modifications for before-variant and after-$\star$ to make the until-$\star$ formulae. In fact, we see this directly with until-any and until-last in Table 6.2.

For until-any, $\delta_1$ combines the $\varepsilon$ (i.e. nothing) from before-variant and the $\mathit{true}^\star \cdot a$ from after-any into $\mathit{true}^\star \cdot a$. Before-variant has $\overline{b}^\star \cdot q$ for $\delta_2$ and after-any has $\mathit{true}^\star \cdot q$. Not being allowed to do $b$ before $q$ is a stronger condition than being allowed to do any action before $q$, so until-any inherits $\overline{b}^\star \cdot q$. Until-any also inherits $\delta_3 = b$ from before-variant. Finally, $\delta_4 = r$ and $\delta_5 = \mathit{false}$ match both before-variant and after-any.

The values for until-last are found in a similar manner. The only thing of note is that for $\delta_2$, $\overline{b}^\star \cdot q$ and $\overline{a}^\star \cdot q$ get combined as $\overline{a \cup b}^\star \cdot q$, since here both the $a$ and the $b$ are not allowed to occur before the $q$.

Until-first requires more explanation, however. In Table 6.2, the until-first scope is the only place where we use a $+$ in the regular formulae.

Filling in the values for response until-first for the WFA formula, we get

$$\begin{aligned}
&\neg(\langle((\overline{a}^\star \cdot a) + (\mathit{true}^\star \cdot b \cdot \overline{a}^\star \cdot a)) \cdot \overline{b}^\star \cdot q\rangle \\
&\quad \nu X.(\bigwedge_{\lambda \in Act} (\langle\lambda\rangle tt \Rightarrow ( \\
&\quad \mu Y.(\langle b\rangle tt \vee ([\lambda]\mathit{ff} \wedge X) \vee \\
&\qquad \langle\lambda \setminus r\rangle X \vee \langle\overline{\lambda \cup r}\rangle Y)))))
\end{aligned} \tag{6.6}$$

Recall from Section 2.2.5 that $\langle R + Q\rangle\phi = \langle R\rangle\phi \vee \langle Q\rangle\phi$. Hence, Formula 6.6 corresponds to

$$\begin{aligned}
&\neg( \\
&\quad (\langle\overline{a}^\star \cdot a \cdot \overline{b}^\star \cdot q\rangle \nu X.(\bigwedge_{\lambda \in Act} (\langle\lambda\rangle tt \Rightarrow ( \\
&\qquad \mu Y.(\langle b\rangle tt \vee ([\lambda]\mathit{ff} \wedge X) \vee \\
&\qquad\quad \langle\lambda \setminus r\rangle X \vee \langle\overline{\lambda \cup r}\rangle Y))))) \\
&\quad \vee \\
&\quad (\langle \mathit{true}^\star \cdot b \cdot \overline{a}^\star \cdot a \cdot \overline{b}^\star \cdot q\rangle \nu X.(\bigwedge_{\lambda \in Act} (\langle\lambda\rangle tt \Rightarrow ( \\
&\qquad \mu Y.(\langle b\rangle tt \vee ([\lambda]\mathit{ff} \wedge X) \vee \\
&\qquad\quad \langle\lambda \setminus r\rangle X \vee \langle\overline{\lambda \cup r}\rangle Y))))))
\end{aligned}$$

We can see this formula as describing the absence of two types of violating paths: paths where the response behaviour is violated between the first $a$ on the path and the first subsequent $b$; and paths where the response behaviour is violated between the first $a$ *after some* $b$ *on the path* and the first subsequent $b$. The first part follows directly from combining before-variant with after-first. The reason for the second part is that the until-first scope requires that after the any $a$-until-$b$ slice of the path, the behaviour again needs to hold after the next $a$, until the subsequent $b$. Hence, after every $b$ we need to consider the next "first" $a$. This is why we add the second type of violating path. If either type of violating path occurs, the property is violated.

A more concise way of expressing both types of violating path is to use $\delta_1 = (\mathit{true}^\star \cdot b)^\star \cdot \overline{a}^\star \cdot a$. It depends on personal preference which notation one finds easier to understand. Here, we found the separated approach easier to explain.

*Remark.* Just like the after-any and after-first scopes are equivalent for the response behaviour, so are until-any and until-first. Hence, this complicated formula is not actually required, since the until-any formula can be used instead. However, once again this argument does not apply to existence behaviour, so we still give a scope-specific formula which can be used more generally with other behaviours as well.

### 6.4.4 Existence

Most of the existence formulae are not worth discussing again, since our approach for representing scopes is the same as with the response formulae. However, there are a few interesting observations to be made that are specific to the existence behaviour: "$r$ must occur in the scope".

Firstly, for the after scopes, it is interesting to note that existence after-any corresponds to global response. Existence of $r$ after any $a$ means that violations consist of paths where there is at some point an $a$ action and there subsequently is an $r$-free path that either ends in a deadlock or is infinite and fair. Replace the $a$ in this description with a $q$, and this is a violating path for global response. Indeed, in Table 6.2 replacing the $a$ in $\delta_1$ with a $q$ results in $\delta_1 \cdot \delta_2$ for global response being the same as $\delta_1 \cdot \delta_2$ for existence after-any.

Secondly, regarding existence after-last: we want to briefly note why this formula is required. We previously discussed that for response behaviour, after-any and after-first are equivalent, and that this is not the case for existence behaviour. One may wonder if existence after-any is equivalent to existence after-last, then. After all, if an $r$ occurs after the last $a$ then it also occurs after *any* $a$. However, any behaviour under the after-last scope can also be satisfied if there are infinitely many $a$'s on a path, since there is no "last" then. This is not the case for after-any. Hence, we do require a separate formula.

# Chapter 7

# Generalising Formulae

In the previous chapter, we gave several base formulae for representing different PSP patterns under fairness assumptions in the modal $\mu$-calculus. So far, we have covered three fairness assumptions: weak fairness of actions, strong fairness of actions and fair reachability of actions. In this chapter, we present a few ways to modify these formulae to cover a broader range of fairness assumptions.

In Section 7.1, we consider how to specify that only some actions need to be treated fairly. In Section 7.2, we explore how arbitrary tasks can be used in formulae. We present formulae that work with the global fairness definitions from Section 3.2, as well other task definitions. We do need to account for some of the limits of the $\mu$-calculus, which we discuss in this section.

We do not provide formal proofs for the formulae in this chapter, since we did not have the time to finish these. However, we do provide arguments for why the modifications we make to the formulae from Chapter 6 are correct.

## 7.1 Fair and Unfair Actions

In Chapter 4, we stated that the precondition weak fairness of actions formula expresses that only the $r$-action is treated fairly. So far, our interest has been in formulae that express all actions are treated fairly. After all, if you are assuming your system has some form of fair scheduling implemented then it makes sense to say all actions in the model of this system must be treated fairly. However, there may be cases where only a subset of actions should be treated fairly. For example, one may model two systems that communicate with each other, where the behaviour of each system independently is fair but their communication is not.

We here present variants of the base formulae from Chapter 6 under a fairness assumption where only a subset of actions is treated fairly. For these formulae, we divide the set $Act$ into two sets: $C$ and $N$ so that $C \cap N = \emptyset$ and $C \cup N = Act$.

The set $C$ contains those actions that we have chosen to treat fairly, $N$ those we do not treat fairly. We still assume progress on all actions, not just those in $C$.

### 7.1.1 Generalised Weak Fairness of Actions (GWFA)

Let us first consider the base weak fairness of actions formula from Section 6.2:

$$\begin{aligned}\neg(\langle \delta_1 \cdot \delta_2\rangle \nu X.(&\bigwedge_{\lambda\in Act} (\langle\lambda\rangle tt \Rightarrow (\\ &\quad \mu Y.(\langle\delta_3\rangle tt \vee ([\lambda]ff \wedge X) \vee \\ &\qquad \langle \lambda \setminus (\delta_4 \cup \delta_5)\rangle X \vee \langle \overline{\lambda \cup \delta_4 \cup \delta_5}\rangle Y)))))\end{aligned}$$

The conjunction over actions says that for all actions $\lambda$ that are enabled, there must exist a sequence of actions along which a few conditions are respected (depending on $\delta_3$, $\delta_4$ and $\delta_5$) and which leads to $\lambda$ either being disabled or occurring, and then the condition needs to hold again. Crucially, the least fixpoint requires $\lambda$ to be treated fairly regardless of how the variables are filled in. A reasonable first step towards generalising this formula is then to replace $\lambda \in Act$ with $\lambda \in C$, so that only the fairly-treated actions have this requirement. This alone is not satisfactory however, since too many traces may be accepted. When discussing the WFA formula for global response, we stated that the way the WFA formula accounts for a path being allowed to end in a deadlock state is because in those states, $\langle\lambda\rangle tt$ will be false for all $\lambda \in Act$ and so the implication is trivially true for all actions. If we only take $\lambda \in C$, however, the formula says any path that ends in a state where no fairly treated actions are enabled forms a valid counterexample. This contradicts the progress assumption: if there are actions in $N$ that are still enabled, the path is not yet complete. We must therefore explicitly incorporate that if no actions in $C$ are enabled but we are not in a deadlock, then there must be an action in $N$ that can be taken that leads to a violating, fair path. Of course, the action that is taken may not be in $\delta_4$ or $\delta_5$. We also still need to consider $\delta_3$.

These modifications together result in Formula 7.1.

$$\begin{aligned}\neg(\langle \delta_1 \cdot \delta_2\rangle \nu X.(&\\ &\bigwedge_{\lambda\in C} (\langle\lambda\rangle tt \Rightarrow (\\ &\quad \mu Y.(\langle\delta_3\rangle tt \vee ([\lambda]ff \wedge X) \vee \\ &\qquad \langle \lambda \setminus (\delta_4 \cup \delta_5)\rangle X \vee \langle \overline{\lambda \cup \delta_4 \cup \delta_5}\rangle Y))) \\ &\wedge ([C]ff \Rightarrow ([true]ff \vee \langle\delta_3\rangle tt \vee \langle\overline{\delta_4 \cup \delta_5}\rangle X))))\end{aligned} \tag{7.1}$$

One may wonder if this formula accounts for the possibility that, in order to construct the violating path, at some point an action in $N$ needs to be taken while there are actions in $C$ still enabled. While it may not be immediately obvious from reading the formula, this is indeed considered. Note that if an action $\lambda \in C$ is enabled, then it will need to be satisfied either by being taken eventually or by eventually no longer being enabled. However, as long as it is a finite sequence, it may take transitions of any action other than $\lambda$ or those in $\delta_4$ and $\delta_5$ to get to this point, including those actions in $N$. This is covered by the $\langle\overline{\lambda \cup \delta_4 \cup \delta_5}\rangle Y$-part.

If $C$ is empty, i.e. no actions are being treated fairly, this formula reduces to

$$\neg(\langle\delta_1 \cdot \delta_2\rangle \nu X.([true]ff \vee \langle\delta_3\rangle tt \vee \langle\overline{\delta_4 \cup \delta_5}\rangle X))$$

If we fill in the global response values, we get exactly Formula 4.2, the global response without fairness formula in the non-violate style. This indicates we are handling the actions in $N$ correctly.

### 7.1.2 Generalised Strong Fairness of Actions (GSFA)

We restate the SFA formulae, Formula 6.2, here:

$$\neg(\langle\delta_1 \cdot \delta_2\rangle( \bigvee_{F \subseteq Act} (\mu Y.(\langle\overline{\delta_4 \cup \delta_5}\rangle Y \vee \langle\delta_3\rangle tt \vee \nu X.inf(F)))))$$

Where $inf(F)$ is defined as follows. We fix an arbitrary order on the actions in $F$ such that $\alpha_1$ is the first action, $\alpha_2$ the second, etc. Let $n = |F|$.

$$\begin{aligned}
inf(F) &= exec(F, n)\\
exec(F, 0) &= [\overline{F}]ff \wedge X\\
exec(F, k+1) &= \mu W_{k+1}.([\overline{F}]ff \wedge\\
&\quad (\langle\overline{\delta_4 \cup \delta_5}\rangle W_{k+1} \vee \langle\alpha_{k+1} \setminus (\delta_4 \cup \delta_5)\rangle exec(F, k)))
\end{aligned}$$

Recall that the strong fairness of actions formula works by considering for every action both the possibility that in some suffix of the violating path the action is taken infinitely often (in which case it is in $F$), and that in some suffix of the violating path the action is never enabled (in which case it is in $\overline{F}$). Actions that need not be treated fairly do not fall into either category: it is alright if they are relentlessly enabled and yet taken only finitely often. So $F$ only needs to be some subset of $C$, rather than of $Act$. The actions in $F$ should then still be taken infinitely often eventually, but along this path it is no longer the case that all actions in $\overline{F}$ need to remain disabled, since the actions in $N$ are also in $\overline{F}$. Instead, it is only the actions in $C \setminus F$ that need to remain disabled. We do need to consider how to handle finite paths ending in a deadlock. In the original formula, this was covered by $F = \emptyset$. In this case, the part of the formula dependent on $F$

reduces to whether it is possible to reach a deadlock in finitely many steps, without taking actions in $\delta_4$ or $\delta_5$. However, now that $F$ is a subset of $C$ rather than $Act$, this is no longer the case. Hence, we need to explicitly include the possibility of reaching a deadlock. This could be done as a separate formula, but can also be incorporated directly into the definition of the least fixpoint with variable $Y$. We take this latter approach.

There is one final factor we need to consider, which also came up with the weak fairness formula: are actions in $N$ still allowed to occur freely? It turns out that while $exec(F, k+1)$ handles this correctly, since any action not in $\delta_4$ or $\delta_5$ is allowed to occur until $\alpha_{k+1}$ occurs, the way $exec(F, 0)$ is defined causes problems when we generalise the formula to allow non-fair actions. Similar to the issue we had with the GWFA formula, $[C \setminus F]\mathit{ff} \land X$ may be true in states that are not deadlock states, and hence do not indicate we have found a fair and complete violating path. We fix this by adding $\langle\overline{\delta_4 \cup \delta_5}\rangle$ in front of the $X$, which ensures that even when $F = \emptyset$, the formula checks for the existence of *complete* violating paths. This change has no negative impact on the semantics of the formula: since *inf* ensure the actions in $C \setminus F$ will be disabled in the states where $X$ holds, we are not accidentally allowing a state that violates this condition to be on the path. Additionally, *inf* says every action occurs infinitely often, which actions occur in-between does not matter (except they are not in $\delta_4 \cup \delta_5$), so the extra transition included in the path by placing $\langle\overline{\delta_4 \cup \delta_5}\rangle$ before $X$ is fine.

The modified formula for GSFA is given as Formula 7.2.

$$\neg(\langle\delta_1 \cdot \delta_2\rangle(\bigvee_{F \subseteq C}(\mu Y.(\langle\overline{\delta_4 \cup \delta_5}\rangle Y \lor [true]\mathit{ff} \lor \langle\delta_3\rangle tt \lor \nu X.\mathit{inf}_G(F))))) \tag{7.2}$$

With

$$\begin{aligned}
\mathit{inf}_G(F) &= \mathit{exec}_G(F, n)\\
\mathit{exec}_G(F, 0) &= [C \setminus F]\mathit{ff} \land \langle\overline{\delta_4 \cup \delta_5}\rangle X\\
\mathit{exec}_G(F, k+1) &= \mu W_{k+1}.([C \setminus F]\mathit{ff} \land\\
&\qquad (\langle\overline{\delta_4 \cup \delta_5}\rangle W_{k+1} \lor \langle\alpha_{k+1} \setminus (\delta_4 \cup \delta_5)\rangle \mathit{exec}_G(F, k)))
\end{aligned}$$

Where $\alpha_1$ is the first action in $F$, $\alpha_2$ the second, etc. until $\alpha_n$, for some arbitrary order on the actions in $F$.

Let us again consider what this formula looks like when $C = \emptyset$. We expect that if we fill in the variables for global response, we get Formula 4.2. There is only a single $F$ possible when $C = \emptyset$, namely $F = \emptyset$. We get the following formula for global response with $C = \emptyset$:

$$\neg(\langle true^\star \cdot q\rangle\mu Y.(\langle\overline{r}\rangle Y \lor [true]\mathit{ff} \lor \nu X.(\langle\overline{r}\rangle X)))$$

Syntactically, this is different from Formula 4.2: instead of $\nu X.([true]ff \vee \langle\overline{r}\rangle X)$ we have $\mu Y.(\langle\overline{r}\rangle Y \vee [true]ff \vee \nu X.(\langle\overline{r}\rangle X))$. But semantically, they are equivalent. We do not provide a formal proof of this, due to time constraints. However, informally it obvious that both express that there either exists a finite $r$-free path ending in a deadlock, or an infinite $r$-free path. MLSolver also confirms they are equivalent, at least up to an alphabet of size 8.

### 7.1.3 Generalised Fair Reachability of Actions (GFRA)

The GFRA formula is designed in much the same way as the GWFA formula. Unsurprising, given the similarities in the base formulae. We just need to ensure that now, the option to ignore the actions in $C$ and take an action in $N$ should only be available when none of the actions in $C$ are reachable, rather than whenever none of them are enabled. The result is Formula 7.3.

$$\begin{aligned}
&\neg(\langle\delta_1 \cdot \delta_2\rangle \nu X.( \\
&\quad\bigwedge_{\lambda\in C}(\langle true^\star \cdot \lambda\rangle tt \Rightarrow ( \\
&\qquad \mu Y.(\langle\delta_3\rangle tt \vee ([true^\star \cdot \lambda]ff \wedge X) \vee \\
&\qquad\quad \langle\lambda \setminus (\delta_4 \cup \delta_5)\rangle X \vee \langle\overline{\lambda \cup \delta_4 \cup \delta_5}\rangle Y))) \\
&\quad \wedge ([true^\star \cdot C]ff \Rightarrow ([true]ff \vee \langle\delta_3\rangle tt \vee \langle\overline{\delta_4 \cup \delta_5}\rangle X))))
\end{aligned} \tag{7.3}$$

### 7.1.4 Combining All Generalised Formulae

Out of interest, we also present a formula that combines all of the formulae previously presented in this chapter. The idea is to have a formula that states different actions fall under different fairness assumptions. We do not account for one actions falling under multiple fairness assumptions; there is a strict hierarchy between these assumptions, see Lemma 3.6, so if an action falls under multiple assumptions it is sufficient to only consider it affected by the strongest of those assumptions.

Let $S$ be the set of actions that should be treated strongly fairly, $W$ the set of actions that should be treated weakly fairly, $R$ the set of fair reachability actions, and finally $N$ for actions that need not be treated fairly at all. These sets should be pairwise disjoint, and we require that the union of all of them equals $Act$.

The basis for the combined formula is the GSFA formula. We still need to consider all possible subsets $S'$ of $S$ to determine which strongly fair actions are enabled finitely often and which are taken infinitely often.

Next, consider that for all actions in $W$, they must either be infinitely often taken, or infinitely often disabled. After all, if they are only finitely often disabled on an infinite path, then there is some suffix of that path on which they are perpetually enabled and hence should occur infinitely often. A similar observation can be made about the actions in $R$: they either occur infinitely often or are unreachable infinitely often. We can use these observations to handle $W$ and $R$ similarly to how $S$ is handled, running down all the actions in the sets. Except in this case, they are satisfied if we can, in finitely many steps, either take the action or observe it is disabled or unreachable respectively. Of course, the elements of $S$ that are not in $S'$ need to remain disabled during these sections.

$$\begin{aligned}\neg(\langle\delta_1 \cdot \delta_2\rangle( \bigvee_{S'\subseteq S} (\mu Y.(\langle\overline{\delta_4 \cup \delta_5}\rangle Y \vee [\mathit{true}]\mathit{ff}\vee \\ \langle\delta_3\rangle \mathit{tt} \vee \nu X.\mathit{inf}_C(S', W, R)))))\end{aligned} \tag{7.4}$$

Let the elements in $S'$ be $\alpha_1, \alpha_2, \ldots, \alpha_s$, the elements in $W$ $\beta_1, \beta_2, \ldots, \beta_w$, and the elements in $R$ $\gamma_1, \gamma_2, \ldots, \gamma_r$. We define the following functions:

$$\begin{aligned}
\mathit{inf}_C(S', W, R) &= \mathit{strong}(S', W, R, s)\\
\mathit{strong}(S', W, R, k+1) &= \mu A_{k+1}.([S \setminus S']\mathit{ff}\wedge\\
&\quad(\langle\overline{\delta_4 \cup \delta_5}\rangle A_{k+1}\vee\\
&\qquad\langle\alpha_{k+1} \setminus (\delta_4 \cup \delta_5)\rangle \mathit{strong}(S', W, R, k)))\\
\mathit{strong}(S', W, R, 0) &= [S \setminus S']\mathit{ff} \wedge \mathit{weak}(S', W, R, w)\\
\mathit{weak}(S', W, R, k+1) &= \mu B_{k+1}.([S \setminus S']\mathit{ff}\wedge\\
&\quad(\langle\overline{\delta_4 \cup \delta_5}\rangle B_{k+1}\vee\\
&\qquad([\beta_{k+1}]\mathit{ff} \wedge \mathit{weak}(S', W, R, k))\vee\\
&\qquad\langle\beta_{k+1} \setminus (\delta_4 \cup \delta_5)\rangle \mathit{weak}(S', W, R, k)))\\
\mathit{weak}(S', W, R, 0) &= [S \setminus S']\mathit{ff} \wedge \mathit{reach}(S', W, R, r)\\
\mathit{reach}(S', W, R, k+1) &= \mu C_{k+1}.([S \setminus S']\mathit{ff}\wedge\\
&\quad(\langle\overline{\delta_4 \cup \delta_5}\rangle C_{k+1}\vee\\
&\qquad([\mathit{true}^\star \cdot \gamma_{k+1}]\mathit{ff} \wedge \mathit{reach}(S', W, R, k))\vee\\
&\qquad\langle\gamma_{k+1} \setminus (\delta_4 \cup \delta_5)\rangle \mathit{reach}(S', W, R, k)))\\
\mathit{reach}(S', W, R, 0) &= [S \setminus S']\mathit{ff} \wedge \langle\overline{\delta_4 \cup \delta_5}\rangle X
\end{aligned}$$

These definitions are somewhat overwhelming to read through, but they boil down to considering every action that is not in $N$ in turn, and ensuring that the action is treated as appropriate for the fairness assumption that applies to that action.

We have not addressed unconditional fairness since Chapter 4. It may be interesting to note here that should there be a set of actions $U$ that one wishes to be treated unconditionally fair, these actions can simply be added to $S'$. This enforces that these actions must occur infinitely often in a violating path.

## 7.2 Tasks

The formulae presented thus far have been for fairness assumptions based on actions. In fact, we have not considered sets of transitions at all in our formulae, only action labels. This is for a very simple reason: with the modal $\mu$-calculus as defined in Section 2.2 the only information of an LTS we can reference is the action labels of transitions. There is no state information in this version of the modal $\mu$-calculus, although variants of the $\mu$-calculus that include atomic propositions similar to LTL or CTL do exist.

There certainly is no information available about individual transitions. That such information can never be known follows from the modal $\mu$-calculus not being able to distinguish bisimilar LTSs. We have not discussed notions of LTS equivalence in this thesis, so we will not go into the specifics of this argument. It suffices to say that there is a notion of equivalence between labelled transition systems known as bisimulation equivalence and that it is a known fact that two LTSs satisfy exactly the same modal $\mu$-calculus formulae if, and only if, they are bisimulation equivalent [9]. Consider the two LTSs in Figure 7.1. These two are bisimulation equivalent and we therefore know that it is impossible to construct a modal $\mu$-calculus formula that is satisfied by one but not the other. However, if we could reference transitions directly, distinguishing the two would be trivial. For example, $t_1$ can occur infinitely often in the left system, but there is no transition that can occur infinitely often in the right system.

Actions are the only thing we can reference, and so it may seem that all other task-based definitions are impossible to represent in the modal $\mu$-calculus. To some extent, this is true. Information about transitions, components and instructions is not directly available in the modal $\mu$-calculus and so we do not have a solution to this problem that can work for arbitrary transition systems. Yet, if we have the ability to construct or modify the LTS we are analysing ourselves, then there are often methods for encoding such information in the action labels. In effect, we are translating the other task definitions back to the action-based setting. We might annotate the action label of each transition with which component(s) are responsible for that transition, for example. We could also add information about which instruction(s) gave rise to a transition in the same way. If there are finitely many transitions in the system, we may even give them all a unique label so that

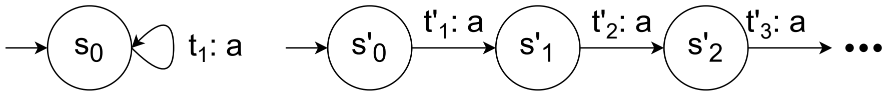


Figure 7.1: The two LTSs in this image are bisimulation equivalent. The one on the right represents an infinite sequence of transitions labelled with the $a$-action.

we can distinguish individual transitions.

Of course, all these methods only work if we have access to this information and the ability to modify the action labels in the LTS. Additionally, we have assumed *Act* is finite and several of our proofs depend on this assumption, so this approach does not work if our label modification would result in an infinite set of actions.

In cases where the relabelling trick is possible, we only need to turn each task, which is a set of transitions, into a set of actions that these transitions are labelled with. We can then analyse fairness with respect to these new tasks over actions. For the remainder of this section, when we discuss tasks we are talking about sets of actions rather than sets of transitions.

The formulae over actions that have been presented in previous chapters can be turned into formulae over sets of actions without much difficulty. Recall from Section 2.2 that we already allow for the box and diamond modalities to be applied to sets of actions, rather than individual actions only. Applying this modification directly, the base weak fairness for arbitrary set of tasks $\mathcal{T}$ becomes

$$\begin{aligned}\neg(\langle \delta_1 \cdot \delta_2\rangle \nu X.(&\bigwedge_{t\in\mathcal{T}}(\langle t\rangle tt \Rightarrow (\\ &\mu Y.(\langle \delta_3\rangle tt \vee ([t]f\!f \wedge X) \vee \\ &\qquad \langle t \setminus (\delta_4 \cup \delta_5)\rangle X \vee \langle \overline{t \cup \delta_4 \cup \delta_5}\rangle Y)))))\end{aligned}$$

We have not changed how the variables $\delta_1$ to $\delta_5$ are incorporated, since we are only modifying the fairness assumption, not the property that is being represented. For global response, we still have $\delta_4 = r$ and the other assignments from Table 6.2.

Note that $\langle t \setminus (\delta_4 \cup \delta_5)\rangle X$ excludes specifically the actions in $\delta_4$ and $\delta_5$ from being taken. It is not the case that a task that happens to include actions from $\delta_4$ or $\delta_5$ is not allowed to occur in its entirety, other actions in such a task may still occur. Here we are again making a distinction between the parts of the formula that reflect the fairness assumption, and the parts that reflect the expressed pattern.

There are a couple of factors we need to consider carefully before we can declare the above formula correct. Specifically, what if an action is in several tasks? And what if an action is in no tasks?

It turns out that the formula works just fine if an action is in several tasks. It merely means each of those tasks is enabled whenever that action is enabled. Each of those tasks will need to be satisfied, either by no longer being enabled within finitely many steps, or by having one of the actions in the task (but not $\delta_4$ or $\delta_5$) occur within finitely many steps. It is therefore correct that the right-hand side of the implication needs to be true for all tasks an enabled action is in.

Things are more complicated if there is an action that is in no task. First, it may be worthwhile to consider if this can even occur. The global fairness definitions presented in Section 3.2 all guarantee every transition is in some task. At least, this is the case if *instr* and *comp* never map a transition to the empty set, which according to the definitions in [22] should never occur. Hence for all these global fairness definitions, every transition – and therefore every action that is ever enabled in the system – will be in some task.

In the case of local fairness assumptions on the other hand, we could encounter a scenario where some action is not in any task. In those cases, we run into the same problem we encountered several times in Section 7.1: the progress assumption says a finite violating path is only valid if it ends in a deadlock state, but without further changes this formula would considers paths valid that end in states where the actions that are not in any task are still enabled. Luckily, we have solved this problem already in Section 7.1. Let $\mathcal{T}$ be the set of tasks, and let $C$ be the set of actions that appears in some task in $\mathcal{T}$, then Formula 7.5 is the formula for weak fairness with tasks.

$$\begin{aligned}
&\neg(\langle \delta_1 \cdot \delta_2 \rangle \nu X.( \\
&\quad \bigwedge_{t \in \mathcal{T}} (\langle t \rangle tt \Rightarrow ( \\
&\qquad \mu Y.(\langle \delta_3 \rangle tt \vee ([t]ff \wedge X) \vee \\
&\qquad\quad \langle t \setminus (\delta_4 \cup \delta_5) \rangle X \vee \langle \overline{t \cup \delta_4 \cup \delta_5} \rangle Y))) \\
&\quad \wedge ([C]ff \Rightarrow ([true]ff \vee \langle \delta_3 \rangle tt \vee \langle \overline{\delta_4 \cup \delta_5} \rangle X))))
\end{aligned} \tag{7.5}$$

These same modifications can be made for all formulae presented in this thesis: quantify over tasks instead of actions, and if there are actions that are not in any task then the generalised version should be adapted rather than the base version.

Since the changes are minimal, do not show the the strong fairness of tasks and fair reachability of tasks formulae here, instead they are presented in Appendix F.

# Chapter 8

# Fairness Formulae in mCRL2

In this thesis, we have presented modal $\mu$-calculus formulae which can be used to express a number of properties under a variety of fairness assumptions. In this chapter, we discuss one method for using these formulae for model checking in practice. Calculating whether a model satisfies a given formula is not a task anyone wishes to do by hand, so we will need tool support. This requires our formulae to be in a shape our chosen tool can use.

There exist several tools for model checking with modal $\mu$-calculus formulae. We will here focus on the mCRL2 toolset, introduced in Section 2.3.1, which is the primary tool used at the TU/e and has been used in a number of practical model checking projects [16, 27, 37, 42]. One great benefit of the mCRL2 toolset is that it admits the entire modal $\mu$-calculus, and even incorporates extensions with data parameters for the fixpoint operators. For comparison, the CADP toolset primarily supports the regular alternation-free subset of the modal $\mu$-calculus. This is not enough to represent our formulae: all three of the base formulae in Section 6.2 contain a least fixpoint nested inside a greatest fixpoint in such a way that the formal variable bound by the greatest fixpoint occurs within the least fixpoint. This violates alternation-freedom [35]. The CADP toolset does have some support for specific fairness properties with alternation depth two, through an operator for describing infinite loops [36]. Using this operator, it may be possible to translate some or all of our formulae to a form that can be understood by CADP. We did not further explore this possibility, since the mCRL2 toolset allows us to use our formulae directly with very little modification.

This chapter will explain how to modify an mCRL2 model in such a way that the formulae presented throughout this thesis can be used. We will also present how the formulae themselves should be written in the mCRL2 toolset. Finally, we will give a brief case study of verifying starvation freedom under various fairness assumptions for Dekker's mutual exclusion algorithm. We will pay particular attention on the difficulty of using the SFA formula in practice, and some ways of mitigating the problem. Throughout this chapter, we will focus primarily on the formulae from Chapter 4 and Chapter 6, i.e. fairness of actions. However, we will also briefly discuss how arbitrary tasks can be used, using the formulae from Section 7.2.

## 8.1 The Model

The mCRL2 toolset [11] uses the mCRL2 modelling language [24], a process algebra, to describe models. These can then be interpreted as labelled transition systems. We here only explain the language as far as needed to understand the specific modifications required to allow our formulae to be used. For more information and tutorials please refer to the cited works as well as the tool's website[1].

For the most part, a model does not need to have a special structure or design to work with our formulae. The only issue we run into is that all our formulae quantify over *Act*. Actions, optionally with parameters, form the primitive elements of an mCRL2 model and they are specifically defined as part of a model. The issue for us is that the language has no support for quantifying over these actions. We could circumvent this by manually unwinding the quantifications in the formulae, but that would mean changing our formulae whenever we change the model. Not to mention, it would be rather tedious work. Instead we use a trick, inspired by [7], which allows us to keep the quantification in the formulae.

The idea is that while we cannot quantify over actions directly, we can quantify over parameters of actions. We therefore turn action labels into parameters. To this end, we define an action $l$, for "label", which gets one parameter of a newly defined data type *Label*. The specifics of this data type depend on the model in question, but there should be a unique label for every action that appears in the LTS described by the model. For this discussion, let $\alpha'$ be the label associated with an arbitrary action $\alpha$. Everywhere in the model where an action $\alpha$ is used, we replace it with the multi-action $\alpha \mid l(\alpha')$. A multi-action represents several actions that occur simultaneously. We then hide the action $\alpha$ using the mCRL2 keyword *hide* to tell the model that when $\alpha$ occurs, it should not be shown in the generated LTS. Once we have done this for every action in the model, we get an LTS that is the exact same as the LTS we got before we made our changes, but for every action $\alpha$ every occurrence has been replaced with $l(\alpha')$. These changes only need to be made to actions that are actually visible in the final LTS. For large models, where checking the LTS manually is difficult, the mCRL2 tool ltsinfo with command – –action–label can be used to list all the actions that occur in the LTS[2].

[1]The mCRL2 website can be found at `https://www.mcrl2.org/web/index.html`.

[2]For reasons outside the scope of this thesis, the action *tau* will always be included in the list of actions. This can be ignored.

We demonstrate the addition of labels with an example.

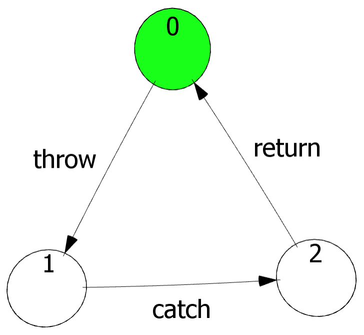
**Example 8.1.** Say we have a simple mCRL2 model of a person playing catch with their dog. The person throws the ball, then waits for their dog to return it. The dog waits until it sees the ball being thrown, catches it and then returns it to their owner. Then they both repeat their actions. In mCRL2, we could model this as follows:

```
act
    throw_s, throw_r, throw;
    return_s, return_r, return;
    catch;
proc
    Person = throw_s . return_r . Person;
    Dog = throw_r . catch . return_s . Dog;
init
    allow({
        throw, return, catch},
    comm({
        throw_s | throw_r -> throw,
        return_s | return_r -> return},
    Person || Dog ));
```

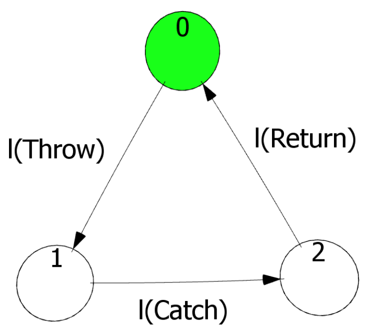
The actions in the model are listed under *act*. We model two recursive processes: the person and their dog, both under the keyword *proc*. We only use the operator ., which represents the sequential composition of actions. Under *init*, we list which actions are allowed to occur and the communication rules for how the synchronisation between the processes should work. Communication happens between the person and their dog: when the person throws the ball (*throw_s*, for "send") the dog has to see it (*throw_r*, for "receive"). Those two actions together are represented by the single action *throw* in the final LTS. A similar rule is introduced for the action *return*. Finally, we state that we want the model to show us the behaviour of *Person* and *Dog* in parallel. See Figure 8.1a for the LTS generated by mCRL2 based on this model.

Adding the labels to this model results in:

```
sort
    Label = struct Throw | Return | Catch;
act
    throw_s, throw_r, throw;
    return_s, return_r, return;
    catch;
    l: Label;
proc
    Person = throw_s|l(Throw) . return_r . Person;
    Dog = throw_r . catch|l(Catch) . return_s|l(Return) . Dog;
init
    hide({
        throw, return, catch},
    allow({
        throw|l, return|l, catch|l},
    comm({
        throw_s | throw_r -> throw,
        return_s | return_r -> return},
Person || Dog )));
```



(a) The LTS for the basic model.



(b) The same LTS, with labels.

Figure 8.1: The LTSs generated for the two catch models.

We add the *Label* data type as a structured sort with three constructors. In this case, the constructors do not get any parameters. We add the action $l$, and define it with a single parameter of type *Label*. The action $l$ is added to all three actions that are visible in the LTS. The left-hand side of communication rules can only be a single action, not a multi-action, so we cannot add $l(\mathit{Throw})$ and $l(\mathit{Return})$ there. Instead, we add these labels to the corresponding $\_s$-action. Finally, we included in the initialisation that the old actions should be hidden, and that the *throw*, *return* and *catch* actions are only allowed to occur together with the the $l$ action. See Figure 8.1b for the resulting LTS.

In addition to needing to quantify over actions for all formulae, the strong fairness formula also requires us to have some fixed order on the actions in $F$. There are a number of ways we could include an ordering; the one we have chosen to use is to create a mapping from natural numbers to the labels that end up in the final model. This means we create a global ordering on all the actions that occur in the LTS, rather than a local ordering on $F$. This leads to a slight modification of the strong fairness formula, which we will detail in the next section. For now, we only focus on how to modify the model to include this ordering.

We use the keyword *map* to add a mapping *order* from natural numbers to the data type *Label*. In order to keep our $\mu$-calculus formulae generic, we also introduce the constant $N$ for the total number of action labels that we map to. We always start the mapping at $\mathit{order}(0)$, and end it at $\mathit{order}(N-1)$. In mCRL2, mappings are specified using a system of equations under the keyword *eqn*. These allow for a lot of flexibility, including using variables in the equations and having conditional equations. In this case, we do not need such features and can simply state for every natural number to which label it is mapped explicitly.

**Example 8.2.** Expanding on Example 8.1, we add an ordering on actions to the second model:

```
map
    order: Nat -> Label;
    N: Nat;
eqn
    order(0) = Throw;
    order(1) = Return;
    order(2) = Catch;
    N = 3;
```

The order of the labels is arbitrary, all that matters is that it is fixed.

These are all the changes that need to be made to a standard mCRL2 model to allow us to use the formulae with action-based fairness.

If we want to have formulae with arbitrary tasks, we will need to define those tasks in the model as well. We need to add a mapping *task* from natural numbers to sets of labels, as well as a constant $T$ for the number of tasks. If there are any labels that do appear in the LTS but not in any of the tasks, then we need the formulae that take this into account as described at the end of Section 7.2. In that case, we also need a constant set of labels $C$ which contains every label that also appears in a task. We have found in practice that when defining $C$ and the tasks, it is best to define these by explicitly listing all the labels instead of using set comprehension. Set comprehension works, but seems to make verification slower.

**Example 8.3.** Say we want a task containing *Throw* and one containing *Return*, but *Catch* does not appear in any tasks. Once again expanding on Example 8.1, we add the following to the model:

```
map
    task: Nat -> Set(Label);
    T: Nat;
    C: Set(Label);
eqn
    task(0) = {Throw};
    task(1) = {Return};
    T = 2;
    C = {Throw, Return};
```

## 8.2 The Formulae

The modal $\mu$-calculus used in the mCRL2 toolset is very similar to the one we present in Section 2.2. There is, however, an additional feature that we will make use of: the ability to add parameters to the formal variables used in fixpoint operators. We will use this feature to write down the strong fairness formula compactly. Besides that, we can represent the formulae pretty much exactly as they have been presented so far, save for some changes in syntax. Instead of $\cdot$ we use ., in place of $\nu$ and $\mu$ we use *nu* and *mu*, $\bigwedge$ is replaced with *forall*, etc. For the specifics of the $\mu$-calculus syntax used by mCRL2, we once again refer to the tool's website[3]. We here merely show how the base formulae presented in Section 6.2 are represented in the tool. We use $\delta'_i$ as variable $\delta_i$ where every action $\alpha$ has been replaced with $l(\alpha')$.

For weak fairness of actions, we use:

```
!(<δ'₁.δ'₂>nu X.(forall a:Label. (<l(a)>true => (
  mu Y.(
        <δ'₃>true
     || ([l(a)]false && X)
     || <l(a) && !(δ'₄ || δ'₅)>X
     || <!(l(a) || δ'₄ || δ'₅)>Y
  )))))
```

The strong fairness of actions formula requires a bit more introduction. Firstly, we move the quantification over sets of actions to the outside of the formula. Note that $\neg(\langle\delta_1 \cdot \delta_2\rangle(\bigvee_{F\subseteq Act}(\ldots)$ is equivalent to $\bigwedge_{F\subseteq Act}(\neg(\langle\delta_1 \cdot \delta_2\rangle \ldots))$. This is not due to any limitations on the part of mCRL2, since the *exists* keyword can be used to represent a $\bigvee$. Rather, this will make it easier for us to separate this formula into many smaller formulae, which we will use in the case study section.

Secondly, there is the matter of how to incorporate the global order on action labels into the SFA formula. As previously mentioned, mCRL2 allows parameters to be included with fixpoint operators. We can use this to represent $W_0$ to $W_n$ from the SFA formula with the single formal variable $W$, which gets a natural number *num* as a parameter . We initialise *num* with 0, and then increment it whenever the *order*(*num*) action is taken. Once *num* hits $N$, we require $X$ to hold again, which corresponds to starting $W$ again with $num = 0$. This means we are moving up through this list of actions rather than down as we did in Formula 6.2. This does not affect the correctness of the formula, since the effect is the same as if we had inverted the arbitrary order we have chosen. This way happens to be easier to write down. The fixpoint operator with formal variable $W$ corresponds to *inf* from Formula 6.2. Since we are dealing with a global order on actions rather than one specific to $F$, if we simply run *num* from 0 to $N$ repeatedly we will encounter situations where *order*(*num*) is not in $F$. We need to explicitly include in our formula that in those cases, the associated action need not occur and *num* should immediately be incremented.

Finally, we must consider how $F$ is represented. The most straightforward approach is to quantify over all possible sets of *Label*. This indeed works, but only if the data type *Label* is finite. In Example 8.1, this is the case: *Label* has three constructors and those constructors do not have any parameters. Hence, quantifying over all possible sets of elements of the type *Label* means quantifying

[3]The specific page on the $\mu$-calculus has been archived at `https://web.archive.org/save/https://www.mcrl2.org/web/user_manual/language_reference/mucalc.html#`

over just $2^3$ options. However, if *Label* is infinite this approach will not work. If we try, mCRL2 will throw an error. Using finite sets instead of normal sets does not seem to resolve the issue either, at least not with the June 2022 release of mCRL2. We can, however, quantify over lists, which we use to solve this problem: we quantify over all Boolean lists of length exactly $N$. This effectively gives us bitmap representations of the sets we actually want: the ones over just those actions that actually occur in the LTS. We can check whether $order(num)$ is included in $F$ by checking if $F$ at $num$ equals *true*.

These modifications gives us the following mCRL2 version of the SFA formula:

```
forall F: List(Bool). (val(#F == N) => (
  !(<$\delta_1'$.$\delta_2'$>mu Y.(<!($\delta_4'$ || $\delta_5'$)>Y || <$\delta_3'$>true || nu X.(
    mu W(num: Nat = 0).(
          (val(num == N) => (
            (forall i:Nat.(val(i < N && !(F.i)) =>
              [l(order(i))]false)) && X))
       && (val(num < N && F.num) => (
                (forall i:Nat.(val(i < N && !(F.i)) =>
                  [l(order(i))]false))
              && (<!($\delta_4'$||$\delta_5'$)>W(num)
                  || <l(order(num)) && !($\delta_4'$||$\delta_5'$)>W(num+1))))
       && (val(num < N && !(F.num)) => (
            W(num + 1)))
  ))))))
```

For fair reachability of actions, we could just use the WFA formula again with $true^\star$ inserted in a few places. However, in practice mCRL2 has trouble determining for which labels $\alpha'$ the condition $\langle true^\star \cdot l(\alpha')\rangle tt$ is satisfied when *Label* is an infinite data type. Determining which labels satisfy $\langle l(\alpha')\rangle tt$ is not a problem even when *Label* is infinite, hence why the WFA formula does not have this issue. To solve this problem, we use the *order* mapping again to ensure we are quantifying over a finite set, namely the natural numbers up to but not including $N$. The resulting FRA formula is:

```
!(<$\delta_1'$.$\delta_2'$>nu X.(forall i: Nat. (val(i < N)
  && <true*.l(order(i))>true => (
    mu Y.(
          <$\delta_3'$>true
       || ([true*.l(order(i))]false && X)
       || <l(order(i)) && !($\delta_4'$ || $\delta_5'$)>X
       || <!(l(order(i)) || $\delta_4'$ || $\delta_5'$)>Y
  )))))
```

Each of these formulae above need to be slightly modified if we want to reference tasks explicitly. In those cases, quantification happens over tasks instead of action labels, and we need to add the special case for when all the tasks are disabled but there are still actions enabled. The main complication is that we cannot directly replace $l(a)$ or $l(\mathit{order}(i))$ in the above formulae with tasks instead of labels. After all, the action $l$ has as parameter a label, not a set of labels. We must therefore use the *exists* (in case of diamond) and *forall* (in case of box) keywords to quantify over all labels in a task. Since they do not differ much from the formulae we have already presented, we do not include the task variants here. Instead, they are shown in Appendix G.

## 8.3 Case Study

In Section 4.6 we mentioned that the strong fairness of actions formula is significantly less efficient than the weak fairness and fair reachability formulae. In this section, we illustrate this observation with a case study of Dekker's mutual exclusion algorithm. This section serves the secondary purpose of demonstrating how the steps described in the rest of this chapter can be applied in practice to analyse a property under multiple fairness assumptions.

In [23], Groote and Keiren present a tutorial on how to model distributed software with mCRL2. As an example, they model and analyse Dekker's algorithm. They present a counterexample showing that Dekker's algorithm does not satisfy starvation freedom[4] without fairness assumption. The authors determine the counterexample to starvation freedom is unfair. They do not state explicitly which fairness assumption they use, but based on their descriptions of why this path is unfair, weak fairness of components seems to fit. Instead of adding a generic fairness assumption to their formula, the authors modify the formula to exclude the specific violating path they observed. They subsequently get a new counterexample, which they judge to still be unfair. Instead of further altering the starvation freedom formula they move on to analysing Peterson's algorithm instead.

We here build on their work to analyse starvation freedom for Dekker's algorithm with WFA, FRA and SFA. We focus on the action-based fairness assumptions since we want to compare the efficiency of these three. Since the original paper gives arguments based on a form of fairness of components, we also briefly address WFC, FRC and SFC. Fortunately for us, the model of Dekker's algorithm used by Groote and Keiren is included with the mCRL2 distribution[5], so we use

[4]In [23], starvation freedom is referred to as eventual access. These two terms refer to the same property.

[5]The model of Dekker's algorithm can be found at `https://github.com/mCRL2org/mCRL2/blob/master/examples/academic/mutex_models/Dekker/Dekker_spec.mcrl2`. Archived: `https://web.archive.org/web/20230901054435/https://raw.githubusercontent.com/mCRL2org/mCRL2/master/examples/academic/mutex_models/Dekker/Dekker_spec.mcrl2`.

it as the base of our model. We do need to make a few modifications, in addition to the changes outlined in Section 8.1.

Firstly, the actions in the original model contain the minimum amount of information required. For example, when thread $i$ reads value $t$ in the register *turn*, this is represented with the action $get\_turn(t)$. That thread $i$ is the one reading is not included in the action. When Groote and Keiren observe that the counterexample they get for starvation freedom is unfair, they say it is unfair because process 1 repeatedly reads $flag[0]$ and *turn* without process 0 getting a chance to read *turn*, even though process 0 is always able to do so. The action labels for process 1 reading the value 1 in *turn* and process 0 doing the same are the same action, however. Consequently, while the action $get\_turn(1)$ is indeed perpetually enabled after a point on this path, it also infinitely often occurs, so weak or strong fairness of actions would not deem this path unfair. This is why it seems more likely that Groote and Keiren intended a form of fairness of components. We are analysing fairness of actions and so to distinguish these two events we add the id of the responsible process to each action. Note that for the analysis of fairness of components, we also need represent this information in the action labels, else we could not assign the actions by different processes to different tasks.

Secondly, we replace the *wish*, *enter* and *leave* actions from the model with the *noncrit* and *crit* actions. Starvation freedom can also be expressed with these actions, and having one action less makes analysing SFA much faster. Since the complexity of the SFA formula is exponential in the number of actions that could be in $F$, even reducing the number of actions by just one makes a big difference.

We also add the modifications described in Section 8.1 to create the final version of the model we analyse, see Appendix H.1. We ensure a one-to-one correspondecne between the LTS of the model without labels and the one with labels, similar to Figure 8.1, by ensuring the *Label* data type has the same parameters for every label as the matching actions have. We use ltsinfo to check exactly which combination of parameters actually occurs in the model, so that we can add only those actions to the *order* mapping. We end up needing to include 18 actions. The order we choose is arbitrary, although we ensure that $order(0)$ is *Crit(0)* and $order(1)$ is *Crit(1)*, for reasons that will soon become apparent.

We construct the formulae for starvation freedom under WFA, FRA and SFA, using the templates from Section 8.2. The starvation freedom property corresponds to the global response pattern, with the added factor that we need to check it twice, once for each process. For starvation freedom of process $i$, the global response pattern is instantiated with $q = noncrit(i)$ and $r = crit(i)$. Using this information, we can fill in the formulae using Table 6.2. We do not include each of the formulae here, since there is little new information. We do show the SFA formula, because it has one small extra change: we already observed in Section 4.6 that we know that $r$ can never be in $F$ when using the global response property. We add this to the SFA formula as well, to reduce the complexity. To do this, we use that *order*($i$) corresponds to *Crit*($i$), so we simply need to include !($F.i$) as part of the conditions of the lists we consider.

```
forall i: Nat. (val(i < 2) => (
  forall F: List(Bool). (val(#F == N && !(F.i)) => (
    !(<true*.l(Noncrit(i))>mu Y.(<!l(Crit(i))>Y || nu X.(
      mu W(num: Nat = 0).(
            (val(num == N) => (
              (forall j:Nat.(val(j < N && !(F.j)) =>
                [l(order(j))]false)) && X))
         && (val(num < N && F.num) => (
                 (forall j:Nat.(val(j < N && !(F.j)) =>
                   [l(order(j))]false))
               && (<!l(Crit(i))>W(num)
                   ||<l(order(num))&&!l(Crit(i))>W(num+1))
              ))
         && (val(num < N && !(F.num)) => (
              W(num+1)))
  ))))))))
```

We do the verification by first turning the mCRL2 model into a linear process specification (LPS) using mcrl2lps, then into an LTS using lps2lts, which we combine with the appropriate formula into a parameterised Boolean equation system (PBES) using lts2pbes. Finally, we using pbessolve to solve the PBES. We do this for starvation freedom under WFA, FRA and SFA.

The WFA formula takes less than a second to check, and reports starvation freedom is satisfied. If our primary goal was to draw conclusions about Dekker's algorithm, rather than experimenting with our formulae, we could stop here. Weak fairness is the weakest of the fairness types we consider, so we can conclude directly that FRA and SFA will also be sufficient. We are interested in the formulae, however, so we still check the other two.

The FRA formula takes slightly longer to calculate than the WFA formula, but still under a second. It reports starvation freedom is satisfied, as expected.

The SFA formula, however, poses some difficulties. On our computer, after about an hour and a half and with 3 million BES equations generated, mCRL2 reports it has run out of memory. This is partially due to the computer we used being mid-range, but it does illustrate a broader issue: this formula is far too big and expensive to calculate as soon as the number of actions goes into the double digits. Having 18 actions is not uncommon for a model, in fact many practical models have many more. Being unable to run the calculation with a model at this scale is therefore an issue. There is a way of circumventing this problem, however. We use that the calculation for a single $i$ and a single $F$ is not expensive. The problem with the SFA formula is that this single formula has to represent all possible choices of $i$ and $F$. Even using the trick of excluding the $r = \mathit{Crit}(i)$ action from $F$, there are still $2 \cdot 2^{17} = 262144$ combinations of $i$ and $F$ that need to be checked. We can deal with this by separating out those parts. This is why we moved the quantification over $\mathit{Act}$ to the outside of the formula in Section 8.2. All we need is a script that automatically generates the $\mu$-calculus formula for a specific $i$ and $F$: this formula then represents that there is no path where $\mathit{Noncrit}(i)$ is not followed by $\mathit{Crit}(i)$ while the actions in $F$ occur infinitely often and those outside $F$ are enabled finitely often. The script then checks whether the model satisfies that formula and saves the result. An added benefit is that if any of the small formulae are not satisfied, then a fair violating path exists and we do not need to keep searching. With limited scripting skills, we wrote a batch script to do this which we include in Appendix H.3. Using this script, the computation is possible but still takes a very long time, on our computer about 48 hours. As expected, it reports no violating paths exist.

This demonstrates that the SFA formula is not practical to use for most models. A more targeted approach is better, for example first assuming WFA. If the property holds under WFA, it also holds under SFA. If the property is violated under WFA, the mCRL2-produced counterexample can give insight into which actions need to be treated strongly fairly. Then the formulae from Chapter 7 can be used to hone in on those actions specifically, which reduces the number of options that need to be considered for $F$.

Alternatively, a form of fairness other than fairness of actions can be used. The issue of the formula is purely that we need to consider every possible way $F$ can be chosen. If we use a form of fairness that results in fewer tasks, then there are far fewer choices of $F$ and the formula becomes much more reasonable to check. When we analyse SFC, for instance, there are only two tasks and so the formula can be checked in a matter of seconds. In fact, in our experiment with starvation freedom under WFC, FRC and SFC it was FRC that was the slowest to verify, taking about 5 seconds. We are uncertain why FRC seems to become slower, possibly it is somewhat expensive to check for reachability of some or all of the labels in a task. As was to be expected, starvation freedom is satisfied under all three assumptions.

The full model, the formulae and the script used in this case study are included in Appendix H, as well as on `https://github.com/MSpronck/FairnessInMucalc`.

# Chapter 9

# Conclusion

In this thesis, the greater-scope problem we wished to address was the difficulty of designing correct modal $\mu$-calculus formulae in practice. We narrowed our focus down to a specific type of formulae: those that describe properties that can be captured by the property specification patterns, evaluated under a fairness assumption. We have designed these modal $\mu$-calculus formulae, and have provided proofs of the semantics of the base formulae that all others are based on, so that users can be confident they express what we claim we express. We have also demonstrated how our formulae can be used in practical model checking projects.

In this chapter, we recap our research questions and provide a summary of the answers we have given to these questions in this thesis. Since many of our results are formulae and proofs, we will not restate these in full, but instead reference where they can be found when appropriate.

While we have largely achieved our goals, time limitations have forced us to prioritise certain parts of the project in favour of others. For example, while correctness proofs are given for the global response formulae and base formulae, no such proofs are provided for the different instantiations of the base formulae for the remaining patterns. We also do not have proofs that the way we generalise the fairness assumptions to cover tasks is fully correct, although we do provide arguments why we believe this method to be correct. This is an aspect of our thesis that could be expanded on in future work. In Section 9.2, the final section of this thesis, we include some additional suggestions for future work.

## 9.1 Research Questions

Our first research question was RQ1: *Which fairness assumptions exist in the literature, and of those which are the most interesting and relevant for us to cover?* Most of this question is addressed in Chapter 3. Based on several papers and books, we determined that for event-based logics such as the modal $\mu$-calculus the assumptions of weak fairness, strong fairness and unconditional fairness are the most commonly discussed. Hence, we decided to cover these at least. Additionally, previous work by Remenska [41] suggests $\mu$-calculus formulae for a type of fairness we named fair reachability, also known as hyperfairness or $\infty$-fairness in the literature. Since there is precedent for this type of fairness being used in modal $\mu$-calculus formulae, we cover it as well. We give definitions for our chosen fairness assumptions in Section 3.1. A brief overview of some of the fairness assumptions from the literature we chose not to cover is included in Section 3.4.

We defined our fairness assumptions over tasks and restated the global task definitions from [22]. For the majority of this thesis, we focused on fairness of actions. We briefly argued why in Section 3.5 and expanded on this argument in Section 7.2: the syntax of the modal $\mu$-calculus lends itself best to expressing properties over actions. In fact, without lifting other aspects of a model, such as individual transitions, component information and instruction information, to the action-domain, we cannot express fairness assumptions on these at all. We discussed how to do this lifting to the action labels in Section 7.2, since we believe it useful to cover task definitions other than fairness of actions as well.

Our second research question was RQ2: *How can our chosen fairness assumptions be integrated into modal $\mu$-calculus formulae following the global response pattern?* Chapter 4 is dedicated to answering this question specifically for fairness of actions. We discussed a few different approaches for adding fairness assumptions to modal $\mu$-calculus formulae we observed in the literature. We called these approaches *model-specific*, *precondition* and *non-violate*. We dismissed model-specific as an approach because it, as the name suggests, is an approach that depends on the specific model being checked. Since our goal was to present generic formulae, this was not the right approach for us. We discussed formulae in both the precondition and non-violate styles, although after Chapter 4 we mostly disregarded the precondition approach, due to the non-violate approach being much more flexible

We ultimately presented the following formulae for global response under fairness: Formula 4.4 for weak fairness of action in the non-violate style; Formula 4.5 and Formula 4.6 for fair reachability of actions in the precondition and non-violate styles respectively; Formula 4.9 for unconditional fairness of actions (plus some equivalent variations); and Formula 4.11 for strong fairness of actions. We proved the correctness for the weak and strong fairness formulae, as well as the fair reachability formula. The unconditional fairness formula is so straightforward we restricted ourselves to an informal argument. While we are mostly satisfied with the formulae presented here, the strong fairness formula is quite inefficient, being exponential in the number of actions in the model. We argued why this is not an entirely unexpected issue with strong fairness in Section 4.6, but could not definitively prove no more efficient formula exists.

The only part of RQ2 that is not answered in Chapter 4 is how to generalise these formulae to other choices of tasks. This is indirectly addressed in Section 7.2, where we argued how to do this generalisation for the pattern-agnostic formulae, which can be adapted to cover global response.

This brings us to RQ3: *How can our chosen fairness assumptions be integrated into modal $\mu$-calculus formulae following the property specification patterns?* To answer this question, we set out to provide formulae for all the different patterns of PSP. Instead of providing these all separately, we presented base formulae that include placeholder variables: Formula 6.1 for weak fairness of actions, Formula 6.2 for strong fairness of actions and Formula 6.3 for fair reachability of actions. We then showed in Table 6.2 how the variables should be filled in to express response and existence under the following scopes: global, before-variant, after-any, after-first, after-last, until-any, until-first, and until-last.

This does not cover all the behaviours and scopes that are part of PSP, or unconditional fairness at all. As argued in Section 6.1, there are several scopes and behaviours that are not affected by feasible fairness assumptions. If a formula in such a pattern is violated by a model, no fairness assumption can eliminate all violating paths. This is because for these patterns, the violation is observable within finitely many steps, and we can extend an arbitrary partial path to a fair complete path for any feasible fairness assumption. Since weak fairness, strong fairness and fair reachability are feasible, which we showed in Section 3.3, there was no point in us giving special fairness formulae for such patterns with these fairness assumptions. For these, we instead pointed to the fairness-free translations by Mateescu and Remenska. While formulae would still be required for the infeasible unconditional fairness assumption, we considered there to be few practical reasons why one would want to make an infeasible fairness assumption, a sentiment that is echoed in the literature. Hence, we chose to disregard unconditional fairness.

For the base formulae, we prove their semantics are what we claim they are. While no proofs are provided for the formulae with the placeholders filled-in, save for global response which was proven earlier, the reader can fill in the variables in the semantics to conclude what the filled-in formulae represent.

Similar to RQ2, RQ3 is only fully answered after Section 7.2, where we explain how these formulae can be modified to cover more than just fairness of actions.

Our final research question was RQ4: *What is required for our designed formulae to be used in the model checking tool mCRL2?* This question is answered in Chapter 8, where we explained how our formulae can be represented and used in mCRL2. Due to our formulae quantifying over actions, and referring to orders on actions, models are required to be modified a bit for the verification to be possible, but this is all possible within the mCRL2 language itself. We gave templates for the formulae in mCRL2, and demonstrated the application of these formulae with a case study on Dekker's mutual exclusion algorithm.

## 9.2 Future Work

There is still plenty of work to be done on this topic. For one, there are still other fairness assumptions that can be considered, such as those mentioned in Section 3.4. In particular, it may be interesting discuss state-based fairness assumptions for the modal $\mu$-calculus with atomic propositions.

Another obvious area of expansion is considering other properties. We focused the property specification patterns because it gave us a stronger foundation than considering every possible property one may write simultaneously, and surveys such as those in [15] and [41] indicate that the patterns cover most of the properties that are used in practice. They do not cover everything, however, and there it is still an open question if and how fairness assumptions can be added to arbitrary properties.

Additionally, we did not actually cover all behaviours from PSP: the chains and bounded existence still need to be discussed, although we suspect our formulae can be quite easily generalised to cover these behaviours as well. Another extensions of our formulae that is likely not too difficult is to account for blocking actions. Under alternative progress assumptions, paths are also complete if they end in states where only blocking actions are enabled. The justness formula from [7], on which our weak fairness and fair reachability formulae are based, includes this.

A question that arose during this project was if the complexity of a $\mu$-calculus formula incorporating the strong fairness assumption is necessarily exponential in the number of tasks. We could not find definitive proof that no better formula is possible. A further exploration of this topic would be interesting. Even if the modal $\mu$-calculus indeed requires exponential formulae for strong fairness, it may be interesting to consider if more efficient formulae are possible in some extension of the $\mu$-calculus such as the polyadic $\mu$-calculus [32] or the hybrid $\mu$-calculus [28].

The future work that we ourselves are the most interested in is formalising our proofs using a proof assistant. At the start of this project, we planned on putting all our proofs in Isabelle/HOL. We did start this process, including putting the modal $\mu$-calculus syntax and semantics into Isabelle files. However, lack of experience with the tool and language made the process very slow, so we did not have enough time to formalise any of our proofs. We would still like to do this; any written proof, no matter how carefully constructed, carries the risk of oversights and errors. While we are quite convinced of the correctness of the formulae presented, confirming the proofs with a proof assistant will guarantee their correctness.

# Bibliography

[1] Apt, K.R., Francez, N., Katz, S.: Appraising fairness in languages for distributed programming. Distributed Computing **2**, 226–241 (1988)

[2] Apt, K.R., Olderog, E.R.: Proof rules and transformations dealing with fairness. Science of Computer Programming **3**(1), 65–100 (1983)

[3] Arts, T., Benac Earle, C., Derrick, J.: Development of a verified Erlang program for resource locking. International Journal on Software Tools for Technology Transfer **5**, 205–220 (2004)

[4] Baeten, J.C.M., Weijland, W.P.: Process Algebra, chap. 4 - Communication, p. 91–118. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press (1990). https://doi.org/10.1017/CBO9780511624193.005

[5] Bellini, P., Nesi, P., Rogai, D.: Expressing and organizing real-time specification patterns via temporal logics. Journal of Systems and Software **82**(2), 183–196 (2009)

[6] Best, E.: Fairness and conspiracies. Information Processing Letters **18**(4), 215–220 (1984)

[7] Bouwman, M., Luttik, B., Willemse, T.A.C.: Off-the-shelf automated analysis of liveness properties for just paths. Acta Informatica **57**(3-5), 551–590 (2020)

[8] Bradfield, J., Stirling, C.: Modal logics and mu-calculi: an introduction. In: Handbook of process algebra, pp. 293–330. Elsevier (2001)

[9] Bradfield, J., Stirling, C.: Modal mu-calculi. Studies in logic and practical reasoning **3**, 721–756 (2007)

[10] Bradfield, J., Walukiewicz, I.: The mu-calculus and model checking. In: Handbook of Model Checking, pp. 871–919. Springer (2018)

[11] Bunte, O., Groote, J.F., Keiren, J.J.A., Laveaux, M., Neele, T., de Vink, E.P., Wesselink, W., Wijs, A., Willemse, T.A.C.: The mCRL2 toolset for analysing concurrent systems. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 21–39. Springer (2019)

[12] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An OpenSource tool for symbolic model checking. In: Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings 14. pp. 359–364. Springer (2002)

[13] Cranen, S., Groote, J.F., Reniers, M.: A linear translation from CTL$^\star$ to the first-order modal $\mu$-calculus. Theoretical Computer Science **412**(28), 3129–3139 (2011)

[14] Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Communications of the ACM **18**(8), 453–457 (1975)

[15] Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the 21st international conference on Software engineering. pp. 411–420 (1999)

[16] van Eekelen, M., ten Hoedt, S., Schreurs, R., Usenko, Y.S.: Analysis of a session-layer protocol in mCRL2. In: Leue, S., Merino, P. (eds.) Formal Methods for Industrial Critical Systems. pp. 182–199. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)

[17] Emerson, E.A., Lei, C.L.: Efficient model checking in fragments of the propositional mu-calculus. In: IEEE Symposium on Logic in Computer Science. pp. 267–278. IEEE Computer Society Press (1986)

[18] Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. Journal of Computer and System Sciences **18**(2), 194–211 (1979). https://doi.org/https://doi.org/10.1016/0022-0000(79)90046-1, `https://www.sciencedirect.com/science/article/pii/0022000079900461`

[19] Francez, N.: Fairness. Springer Science & Business Media (1986)

[20] Friedmann, O., Lange, M.: A solver for modal fixpoint logics. Electronic Notes in Theoretical Computer Science **262**, 99–111 (2010)

[21] Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: a toolbox for the construction and analysis of distributed processes. International Journal on Software Tools for Technology Transfer **15**(2), 89–107 (2013)

[22] van Glabbeek, R.J., Höfner, P.: Progress, Justness, and Fairness. ACM Computing Surveys (CSUR) **52**(4), 1–38 (2019)

[23] Groote, J.F., Keiren, J.J.: Tutorial: designing distributed software in mcrl2. In: 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. pp. 226–243. Springer (2021)

[24] Groote, J.F., Mousavi, M.R.: Modeling and Analysis of Communicating Systems. MIT Press Ltd. (2014)

[25] Grumberg, O., Francez, N., Katz, S.: A complete rule for equifair termination. Journal of Computer and System Sciences **33**(3), 313–332 (1986)

[26] Grunske, L.: Specification patterns for probabilistic quality properties. In: Proceedings of the 30th international conference on Software engineering. pp. 31–40 (2008)

[27] Hwong, Y.L., Keiren, J.J.A., Kusters, V.J.J., Leemans, S., Willemse, T.A.C.: Formalising and analysing the control software of the compact muon solenoid experiment at the large hadron collider. Science of Computer Programming **78**(12), 2435–2452 (2013). https://doi.org/https://doi.org/10.1016/j.scico.2012.11.009, `https://www.sciencedirect.com/science/article/pii/S0167642312002365`

[28] Kernberger, D., Lange, M.: The fully hybrid mu-calculus. In: 24th International Symposium on Temporal Representation and Reasoning (TIME 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)

[29] Kozen, D.: Results on the propositional $\mu$-calculus. Theoretical computer science **27**(3), 333–354 (1983)

[30] Kwiatkowska, M.Z.: Survey of fairness notions. Information and Software Technology **31**(7), 371–386 (1989)

[31] Lamport, L.: Fairness and hyperfairness. Distributed Computing **13**(4), 239–245 (2000)

[32] Lange, M.: The arity hierarchy in the polyadic $\mu$-calculus. EPTCS 191 p. 105 (2015)

[33] Lehmann, D., Pnueli, A., Stavi, J.: Impartiality, justice and fairness: The ethics of concurrent termination. In: Automata, Languages and Programming: Eighth Colloquium Acre (Akko), Israel July 13–17, 1981 8. pp. 264–277. Springer (1981)

[34] Manna, Z., Pnueli, A.: Temporal verification of reactive systems: safety. Springer Science & Business Media (2012)

[35] Mateescu, R., Sighireanu, M.: Efficient on-the-fly model-checking for regular alternation-free mu-calculus. Science of Computer Programming **46**(3), 255–281 (2003)

[36] Mateescu, R., Thivolle, D.: A model checking language for concurrent value-passing systems. In: FM 2008: Formal Methods: 15th International Symposium on Formal Methods, Turku, Finland, May 26-30, 2008 Proceedings 15. pp. 148–164. Springer (2008)

[37] Mathijssen, A., Pretorius, A.J.: Verified design of an automated parking garage. In: Brim, L., Haverkort, B., Leucker, M., van de Pol, J. (eds.) Formal Methods: Applications and Technology. pp. 165–180. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)

[38] Niwiński, D.: On fixed-point clones. In: International Colloquium on Automata, Languages, and Programming. pp. 464–473. Springer (1986)

[39] Pnueli, A.: On the extremely fair treatment of probabilistic algorithms. In: Proceedings of the fifteenth annual ACM symposium on Theory of computing. pp. 278–290 (1983)

[40] Queille, J.P., Sifakis, J.: Fairness and related properties in transition systems—a temporal logic to deal with fairness. Acta informatica **19**, 195–220 (1983)

[41] Remenska, D.: Bringing Model Checking Closer To Practical Software Engineering. Ph.D. thesis, Vrije U., Amsterdam (2016)

[42] Remenska, D., Willemse, T.A.C., Verstoep, K., Templon, J., Bal, H.: Using model checking to analyze the system behavior of the LHC production grid. Future Generation Computer Systems **29**(8), 2239–2251 (2013). https://doi.org/https://doi.org/10.1016/j.future.2013.06.004, `https://www.sciencedirect.com/science/article/pii/S0167739X13001180`

[43] Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. Journal of the ACM (JACM) **32**(3), 733–749 (1985)

[44] Spronck, M.S.C., Luttik, B.: Process-algebraic models of multi-writer multi-reader non-atomic registers. In: 34th International Conference on Concurrency Theory (CONCUR 2023). vol. 279 (2023), to appear

[45] Völzer, H.: On conspiracies and hyperfairness in distributed computing. In: Distributed Computing: 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005. Proceedings 19. pp. 33–47. Springer (2005)

# Appendix A

# Miscellaneous Proofs

This appendix contains the proofs for those lemmas, propositions and theorems where the proof took up too much space or was not relevant enough to be included in the main text. The proofs for the base formulae from Chapter 6 are given in Appendix B instead.

In the proofs in this appendix, as well as in Appendix B, we do use some new notation. Specifically, we on a few occasions need to do calculations on semantics for $\langle\alpha\rangle\phi$ or $[\alpha]\phi$ where $\alpha$ is an action formula. We want to use the $s \xrightarrow{\alpha} s'$ notation in those cases when there is a transition labelled with one of the actions in $\alpha$ between states $s$ and $s'$, but this notation has only been defined for actions, not action formulae. We therefore introduce the notation $s \xrightarrow{\alpha} s'$ to represent $\exists_{a\in\alpha}.s \xrightarrow{a} s'$ .

For all proofs, we fix the model $M = (\mathcal{S}, s_{init}, \mathit{Act}, \mathit{Trans})$

## A.1 Proof of Theorem 4.3

Here we prove how the commonly used formula-shape $\mu Y.(\phi \vee \langle\alpha\rangle Y)$ can be interpreted. This is used in many of our proofs. We prove a few lemmas first.

**Lemma A.1.** *For any environment $\epsilon$ and modal $\mu$-calculus formulae $\phi$ and $\psi$, where $\phi$ does not depend on $Y$, it is the case that $[\![\phi]\!]_\epsilon \subseteq [\![\mu Y.(\phi \vee \psi)]\!]_\epsilon$.*

*Proof.* This follows from the semantics presented in Section 2.2.

$$\begin{aligned}
[\![\mu Y.(\phi \vee \psi)]\!]_\epsilon &= \bigcap\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq [\![\phi \vee \psi]\!]_{\epsilon[Y:=\mathcal{S}']}\} \\
&= \bigcap\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq [\![\phi]\!]_{\epsilon[Y:=\mathcal{S}']} \cup [\![\psi]\!]_{\epsilon[Y:=\mathcal{S}']}\}
\end{aligned}$$

We stated that $\phi$ does not depend on $Y$, therefore $[\![\phi]\!]_{\epsilon[Y:=\mathcal{S}']} = [\![\phi]\!]_\epsilon$. Note the semantics of $\mu Y.(\phi \vee \psi)$ are the intersection of all $\mathcal{S}'$ such that $[\![\phi \vee \psi]\!]_{\epsilon[Y:=\mathcal{S}']} \subseteq \mathcal{S}'$. Since $[\![\phi]\!]_\epsilon \subseteq [\![\phi]\!]_\epsilon \cup [\![\psi]\!]_{\epsilon[Y:=\mathcal{S}']} = [\![\phi \vee \psi]\!]_{\epsilon[Y:=\mathcal{S}']}$, we know that $[\![\phi]\!]_\epsilon \subseteq \mathcal{S}'$ for all such $\mathcal{S}'$. Hence, we can conclude that $[\![\phi]\!]_\epsilon$ is a subset of $[\![\mu Y.(\phi \vee \psi)]\!]_\epsilon$. □

**Lemma A.2.** *If $s' \in [\![\mu Y.(\phi \vee \langle\alpha\rangle\, Y)]\!]_\epsilon$ and $s$ admits an $\alpha$-transition to $s'$, then $s \in [\![\mu Y.(\phi \vee \langle\alpha\rangle\, Y)]\!]_\epsilon$, for all environments $\epsilon$, states $s, s' \in \mathcal{S}$, modal $\mu$-calculus formulae $\phi$ and action formulae $\alpha$.*

*Proof.* We can calculate the semantics of $\mu Y.(\phi \vee \langle\alpha\rangle\, Y)$ as follows:

$$\begin{aligned}
&[\![\mu Y.(\phi \vee \langle\alpha\rangle\, Y)]\!]_\epsilon \\
&\quad = \bigcap\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq [\![\phi \vee \langle\alpha\rangle Y]\!]_{\epsilon[Y:=\mathcal{S}']}\} \\
&\quad = \bigcap\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq [\![\phi]\!]_{\epsilon[Y:=\mathcal{S}']} \cup [\![\langle\alpha\rangle Y]\!]_{\epsilon[Y:=\mathcal{S}']}\} \\
&\quad = \bigcap\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq [\![\phi]\!]_{\epsilon[Y:=\mathcal{S}']} \cup \{s \in \mathcal{S} \mid \exists_{s' \in \mathcal{S}}. s \xrightarrow{\alpha} s' \wedge s' \in [\![Y]\!]_{\epsilon[Y:=\mathcal{S}']}\}\} \\
&\quad = \bigcap\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \supseteq [\![\phi]\!]_{\epsilon[Y:=\mathcal{S}']} \cup \{s \in \mathcal{S} \mid \exists_{s' \in \mathcal{S}}. s \xrightarrow{\alpha} s' \wedge s' \in \mathcal{S}'\}\}
\end{aligned}$$

Note that since $s' \in [\![\mu Y.(\phi \vee \langle\alpha\rangle\, Y)]\!]_\epsilon$, we know that for all $\mathcal{S}' \subseteq \mathcal{S}$ that satisfy $\mathcal{S}' \supseteq [\![\phi]\!]_{\epsilon[Y:=\mathcal{S}']} \cup \{s \in \mathcal{S} \mid \exists_{s' \in \mathcal{S}}. s \xrightarrow{\alpha} s' \wedge s' \in \mathcal{S}'\}$, $s' \in \mathcal{S}'$. If $s' \in \mathcal{S}'$, then since $s$ has an $\alpha$-transition to $s'$, $s \in \{s \in \mathcal{S} \mid \exists_{s' \in \mathcal{S}}. s \xrightarrow{\alpha} s' \wedge s' \in \mathcal{S}'\}$ and hence if $s'$ is in $\mathcal{S}'$, so is $s$. From this we conclude that $s \in [\![\mu Y.(\phi \vee \langle\alpha\rangle\, Y)]\!]_\epsilon$. □

For this next theorem and proof, we need to use an alternate characterisation of the semantics of least fixpoints than the one we presented in Section 2.2. We do not use this characterisation outside of this proof, hence why it is given here rather than in the preliminaries. We also only give the definitions that we require for our proofs. We base ourselves on [8, 9]. Let $Y$ be an arbitrary formal variable, $\phi$ be an arbitrary modal $\mu$-calculus formula and $\epsilon$ an arbitrary environment. Let $T_\phi$ be the transformer associated with $\phi$, defined as

$$T_\phi(\mathcal{F}) = \{s \in \mathcal{S} \mid s \in [\![\phi]\!]_{\epsilon[Y:=\mathcal{F}]}\}$$

And define

$$\begin{aligned}
T^0_\phi(\mathcal{F}) &= \mathcal{F} \\
T^{i+1}_\phi(\mathcal{F}) &= T_\phi(T^i_\phi(\mathcal{F}))
\end{aligned}$$

Then we can calculate the semantics of $\mu Y.(\phi)$ under $\epsilon$ as:

$$[\![\mu Y.(\phi)]\!]_\epsilon = \bigcup_{0 \leq i \leq |\mathcal{S}|} T^i_\phi(\emptyset)$$

Note that this definition only works for finite systems, since it uses $|\mathcal{S}|$. We call $T^i_\phi(\emptyset)$ the $i$'th approximation of $\phi$.

**Lemma A.3.** *For all environments $\epsilon$, states $s \in \mathcal{S}$, modal $\mu$-calculus formulae $\phi$ that do not depend on $Y$, action formulae $\alpha$, and natural numbers $0 \leq i \leq |\mathcal{S}|$, it holds that: $s$ is in the $i$'th approximation of $\mu Y.(\phi \vee \langle\alpha\rangle Y)$ under $\epsilon$ if, and only if, $s$ admits a finite, possibly partial, path of length at most $i-1$ on which only actions in $\alpha$ occur and which ends in a state in $[\![\phi]\!]_\epsilon$.*

*Proof.* Let $T$ be the transformer of $\mu Y.(\phi \vee \langle\alpha\rangle Y)$. We prove that $s$ is in $T^i(\emptyset)$ if, and only if, $s$ admits a finite, possibly partial, path $\pi$ of length at most $i-1$ on which only actions in $\alpha$ occur and which ends in a state $s' \in [\![\phi]\!]_\epsilon$. We do this by induction on $i$.

For the first *base*, take $i = 0$. Note that $s$ is in the $i$'th approximation if $s \in T^0(\emptyset)$. However, $T^0(\emptyset) = \emptyset$, so $s$ cannot be in the 0'th approximation. Indeed, we cannot have a path of length at most $-1$. So in both directions of the bi-implication, the left side of the implication does not hold.

For the second *base*, take $i = 1$.

- First, assume $s$ is in the first approximation. Then $s \in T^1(\emptyset) = \{s \in \mathcal{S} \mid s \in [\![\phi \vee \langle\alpha\rangle Y]\!]_{\epsilon[Y:=\emptyset]}\}$. Hence, $s \in [\![\phi \vee \langle\alpha\rangle Y]\!]_{\epsilon[Y:=\emptyset]}$. Since $\phi$ does not depend on $Y$, this reduces to $s \in [\![\phi]\!]_\epsilon \vee s \in \{s \in \mathcal{S} \mid \exists_{s' \in \mathcal{S}}.s \xrightarrow{\alpha} s' \wedge s' \in \emptyset\}$. It is not possible for a state $s'$ to exist that is in $\emptyset$, hence $s \in [\![\phi]\!]_\epsilon$. We conclude that $s$ admits a finite, possibly partial, path of length at most 0 on which only actions in $\alpha$ occur and which ends in a state in $[\![\phi]\!]_\epsilon$; a witness to this is the path consisting of only $s$.

- Second, assume $s$ admits a path of length at most 0 on which only actions on $\alpha$ occur and which ends in a state in $[\![\phi]\!]_\epsilon$. The only path starting in $s$ of length at most 0 is the path consisting of only $s$. Hence, $s \in [\![\phi]\!]_\epsilon$. Since $\phi$ does not depend on $Y$, we also have $s \in [\![\phi]\!]_{\epsilon[Y:=\emptyset]}$. Hence we also have $s \in [\![\phi]\!]_\epsilon \vee s \in \{s \in \mathcal{S} \mid \exists_{s' \in \mathcal{S}}.s \xrightarrow{\alpha} s' \wedge s' \in \emptyset\}$. We finally conclude $s \in \{s \in \mathcal{S} \mid s \in [\![\phi \vee \langle\alpha\rangle Y]\!]_{\epsilon[Y:=\emptyset]}\} = T^1(\emptyset)$ and hence $s$ is in the first approximation.

The *induction hypothesis* we use is that a state $s'$ is in the $k$'th approximation of $\mu Y.(\phi \vee \langle\alpha\rangle Y)$ if, and only if, $s'$ admits a finite, possibly partial, path of length at most $k-1$ on which only actions in $\alpha$ occur and which ends in a state in $[\![\phi]\!]_\epsilon$. This is for all $k \geq 1$.

For the *step* case, we prove the claim for $k+1$. Let $S$ be the set of states that admit finite, possibly partial, paths of length at most $k-1$ on which only actions in $\alpha$ occur and which end in a state in $[\![\phi]\!]_\epsilon$. By the induction hypothesis, $S = T^k(\emptyset)$. Since this lemma is a bi-implication, we prove both directions separately.

- We assume $s \in T^{k+1}(\emptyset)$. We need to prove $s$ admits a path $\pi$ that is of length at most $k+1-1 = k$, on which only actions in $\alpha$ occur and which ends in a state in $[\![\phi]\!]_\epsilon$. We have $s \in T^{k+1}(\emptyset) = T(T^k(\emptyset)) = T(S)$ Hence, $s \in \{s \in \mathcal{S} \mid s \in [\![\phi \vee \langle\alpha\rangle Y]\!]_{\epsilon[Y:=S]}\}$. This reduces to $s \in [\![\phi]\!]_\epsilon \vee s \in \{s \in \mathcal{S} \mid \exists_{s' \in \mathcal{S}}. s \xrightarrow{\alpha} s' \wedge s' \in S\}$. We do a case distinction on whether $s \in [\![\phi]\!]_\epsilon$.

- If $s \in [\![\phi]\!]_\epsilon$, then the path $\pi$ consisting of only $s$ is a path of length 0 on which only actions in $\alpha$ occur and which ends in a state in $[\![\phi]\!]_\epsilon$, namely $s$ itself. Since we assumed $k \geq 1$, we know $0 \leq k$, hence $s$ admits a path meeting the requirements of length at most $k$.
- If $s \notin [\![\phi]\!]_\epsilon$, then $s \in \{s \in \mathcal{S} \mid \exists_{s' \in \mathcal{S}}. s \xrightarrow{\alpha} s' \wedge s' \in S\}$. Hence, there exists a state $s'$ such that that there exists an $\alpha$-transition $t$ from $s$ to $s'$ and $s'$ is in $S$. Since $s' \in S$, we know $s'$ admits a path $\pi'$ of length at most $k-1$, on which only actions in $\alpha$ occur and which ends in a state satisfying $[\![\phi]\!]_\epsilon$. Let $\pi = st\pi'$. Since $\pi'$ has length at most $k-1$ and we added one transition, $\pi$ has length at most $k$. Additionally, $t$ is an $\alpha$-transition, as are all transitions in $\pi'$, so all transitions in $\pi$ are labelled with actions in $\alpha$. Finally, since $\pi'$ ends in a state satisfying $[\![\phi]\!]_\epsilon$, so does $\pi$. Hence, $\pi$ is a witness that $s$ admits a path meeting all requirements.

In both cases $s$ admits such a path $\pi$ of length at most $k$.

- We assume $s$ admits a path $\pi$ of length at most $k$ such that all transitions on $\pi$ are labelled with actions in $\alpha$ and $\pi$ ends in a state satisfying $[\![\phi]\!]_\epsilon$. We prove $s \in T^{k+1}(\emptyset) = T(T^k(\emptyset)) = T(S) = \{s \in \mathcal{S} \mid s \in [\![\phi \vee \langle\alpha\rangle Y]\!]_{\epsilon[Y:=S]}\}$. We do a case distinction on whether the length of $\pi$ is zero.

  - If the length of $\pi$ is zero, then $\pi = s$ and $s \in [\![\phi]\!]_\epsilon$. Since $\phi$ does not depend on $Y$, we also have $s \in [\![\phi]\!]_{\epsilon[Y:=S]}$ and hence also $s \in [\![\phi \vee \langle\alpha\rangle Y]\!]_{\epsilon[Y:=S]}$. Thus, $s \in T(S) = T^{k+1}(\emptyset)$.
  - If the length of $\pi$ is greater than zero, then there is at least one transition in $\pi$. Let $t$ be the first transition of $\pi$. Since there are only $\alpha$-transitions in $\pi$, $t$ is an $\alpha$-transition. Let $s'$ be the target of $t$, and let $\pi'$ be the suffix of $\pi$ starting in $s'$. Then since the length of $\pi$ is at most $k$, the length of $\pi'$ is at most $k-1$. Hence, $\pi'$ witnesses that $s'$ admits a path of length at most $k-1$ on which only $\alpha$-transitions occur and which ends in a state satisfying $[\![\phi]\!]_\epsilon$. Hence, $s \in S$. So $s$ admits an $\alpha$-transition, namely $t$, to a state in $S$, namely $s'$. Therefore $s \in [\![\langle\alpha\rangle Y]\!]_{\epsilon[Y:=S]}$ and hence also $s \in [\![\phi \vee \langle\alpha\rangle Y]\!]_{\epsilon[Y:=S]}$. We conclude that $s \in T(S) = T^{k+1}(\emptyset)$.

  In both cases we demonstrate that $s$ is in the $k+1$'th approximation.

We have proven both sides of the bi-implication that $s$ is in the $k+1$'th approximation of $\mu Y.(\phi \vee \langle\alpha\rangle Y)$ if, and only if, $s$ admits a finite, possibly partial, path of length at most $k$ on which only $\alpha$ actions occur and which ends in a state satisfying $[\![\phi]\!]_\epsilon$. This proves the step case.

By induction, we have proven the claim for all $0 \leq i \leq |\mathcal{S}|$. $\square$

Finally, we restate Theorem 4.3:

**Theorem A.4.** *For all environments $\epsilon$, states $s \in \mathcal{S}$, modal $\mu$-calculus formulae $\phi$ that do not depend on $Y$, and action formulae $\alpha$, it holds that: $s$ is in $[\![\mu Y.(\phi \vee \langle\alpha\rangle Y)]\!]_\epsilon$ if, and only if, $s$ admits a finite, possibly partial, path on which only actions in $\alpha$ occur and which ends in a state in $[\![\phi]\!]_\epsilon$.*

*Proof.* There are two sides to this proof.

- First, we assume $s \in [\![\mu Y.(\phi \vee \langle\alpha\rangle Y)]\!]_\epsilon$ and prove that $s$ admits a finite, possibly partial, path on which only actions in $\alpha$ occur and which ends in a state satisfying $\phi$. The transformer for this least fixpoint is $T(\mathcal{F}) = \{s \in \mathcal{S} \mid s \in [\![\phi \vee \langle\alpha\rangle Y]\!]_{\epsilon[Y:=\mathcal{F}]}\}$. By the alternative semantics definition, $s \in [\![\mu Y.(\phi \vee \langle\alpha\rangle Y)]\!]_{\epsilon[X:=\mathcal{F}]}$ implies $s \in \bigcup_{0\leq i\leq|\mathcal{S}|} T^i(\emptyset)$, where $T$ is the transformer matching $\mu Y.(\phi \vee \langle\alpha\rangle Y)$. Hence, there exist one or more natural numbers $i$ such that $s \in T^i(\emptyset)$. Let $i$ be the smallest such number. Then by Lemma A.3, $s$ admits a finite, possibly partial, path of length at most $i-1$ on which only actions in $\alpha$ occur and which ends in a state in $[\![\phi]\!]_\epsilon$. This path is a witness for $s$ admitting such a path of arbitrary length.

- Secondly, we assume $s$ admits a finite, possibly partial, path on which only actions in $\alpha$ occur and which ends in a state satisfying $\phi$ and prove that $s \in [\![\mu Y.(\phi \vee \langle\alpha\rangle Y)]\!]_\epsilon$. Let $\pi$ be the finite path admitted by $s$ on which only actions in $\alpha$ occur and which ends in a state satisfying $\phi$. We do induction on the number of transitions in $\pi$.

  For the *base*, we assume that $\phi$ contains zero transitions and hence is made up of only the state $s$. Since $\pi$ ends in a state satisfying $\phi$, we conclude that $s$ satisfies $\phi$. The semantics of $\mu Y.(\phi \vee \langle\alpha\rangle Y)$ are a superset of the semantics of $\phi$, hence $s \in [\![\mu Y.(\phi \vee \langle\alpha\rangle Y)]\!]_\epsilon$, see Lemma A.1.

  For the *step*, we assume that $\pi$ contains $k+1$ transitions. Our *induction hypothesis* is that for every state $s'$ that admits a possibly partial path consisting of $k$ transitions in which only actions in $\alpha$ occur and which ends in a state satisfying $\phi$, $s' \in [\![\mu Y.(\phi \vee \langle\alpha\rangle Y)]\!]_\epsilon$. Consider the states $s_0 = s$ to $s_{k+1}$ on $\pi$. The state $s_1$ admits a path consisting of $k$ transitions in which only actions in $\alpha$ occur and which ends in a state satisfying $\phi$, namely the suffix of $\pi$ starting in $s_1$. Hence by the induction hypothesis, $s_1 \in [\![\mu Y.(\phi \vee \langle\alpha\rangle Y)]\!]_\epsilon$.

We also know, since $\pi$ is a path of only $\alpha$-actions and $s_1$ is the state directly after $s$ on $\pi$, that $s$ admits an $\alpha$-transition to $s_1$. Therefore, we conclude using Lemma A.2 that $s \in [\![\mu Y.(\phi \vee \langle \alpha \rangle Y)]\!]_\epsilon$.

We conclude by induction that, if $s$ admits a finite path on which only $\alpha$ actions occur and which ends in a state satisfying $\phi$, then we have $s \in [\![\mu Y.(\phi \vee \langle \alpha \rangle Y)]\!]_\epsilon$.

We have proven both sides of the bi-implication, and thus the lemma. □

## A.2 Proof of Proposition 4.4

We restate the proposition here.

**Proposition A.5.** *Let $\pi$ be a WFA path , then $s_0 t_1 s_1 ... t_n \pi$ is a WFA path.*

*Proof.* Let $\pi' = s_0 t_1 s_1 ... t_n \pi$ be a path where $\pi$ is a WFA path. Consider an arbitrary suffix of $\pi'$, $\pi''$ and an arbitrary action $\lambda$ that is enabled perpetually in $\pi''$. We prove that $\lambda$ occurs in $\pi''$. Since $\pi$ is a suffix of $\pi'$, $\pi''$ will either contain $\pi$ in full or $\pi''$ will also be a suffix of $\pi$.

- If $\pi''$ contains $\pi$, then $\lambda$ is also perpetually enabled in $\pi$. Hence, $\lambda$ is perpetually enabled in any suffix of $\pi$. Since $\pi$ is a WFA path, this means $\lambda$ occurs in any suffix of $\pi$, meaning $\lambda$ occurs in $\pi$ itself and since $\pi$ is part of $\pi''$, $\lambda$ also occurs in $\pi''$.
- If $\pi''$ is also a suffix of $\pi$, then since $\pi$ satisfies WFA and $\lambda$ is perpetually enabled in $\pi''$, we can directly conclude $\lambda$ occurs in $\pi''$.

We have shown that in both cases, $\lambda$ occurs in $\pi''$. We can conclude that for every suffix of $\pi'$, any action that is enabled perpetually in that suffix must occur in that suffix. Hence, $\pi'$ satisfies WFA. □

## A.3 Proof of Proposition 4.5

We restate the proposition:

**Proposition A.6.** *Let $\pi = s_0 t_1 s_1 ...$ be a finite or infinite path. If $\pi$ satisfies WFA then also any suffix of $\pi$ satisfies WFA.*

*Proof.* Let $\pi$ be a WFA path. Let $\pi'$ be a suffix of $\pi$. Trivially, any suffix of $\pi'$ is also a suffix of $\pi$. Hence, for any suffix of $\pi'$ it holds that any action that is perpetually enabled on this suffix occurs in this suffix. We conclude $\pi'$ satisfies WFA. □

## A.4 Proof of Lemma 3.13

Recall we had made the assumption that $\mathcal{T}$ is always finite.

**Lemma A.7.** *Fair reachability is feasible.*

*Proof.* To prove this, we need to show that any finite partial path $\pi_0$ can be extended to a complete path $\pi$ that satisfies fair reachability for an arbitrary choice of tasks. We will construct $\pi$ in steps, let $\pi_i$ with $i \geq 0$ be the paths constructed in each step. Let $s_i$ be the last state of $\pi_i$. For this construction we use a queue $Q$, which is initialised with exactly one copy of every task in the chosen set of tasks $\mathcal{T}$ in some arbitrary order. At each step $i$, we do the following:

Take the head of $Q$, and assign it to variable $T$. If $T$ is not reachable from $s_i$, remove $T$ from $Q$. Repeat until either $Q$ is empty or $T$ is reachable from $s_i$.

- If $Q$ is empty, then $s_i$ is a state from which no tasks are reachable. If $s_i$ is a deadlock state, then $\pi_i$ is a complete finite path and therefore trivially satisfies fair reachability. If $s_i$ is not a deadlock state, then all transitions reachable from $s_i$ are labelled in actions that are not in any task in $\mathcal{T}$. We can extend $s_i$ with arbitrary transitions to construct $\pi$: either a deadlock state is eventually reached or we build an infinite path.

- If we find a $T$ that is reachable from $s_i$, then we know that there exists some $t \in T$ such that there exists a path $\pi'_i$ starting in $s_i$ which ends in a state $s'_i$ such $t$ is enabled in $s'_i$. Let $s''_i = \mathit{target}(t)$. We take $\pi_{i+1}$ to be $\pi_i$ appended with $\pi'_i t s''_i$. Thus, $\pi_{i+1}$ is $\pi_i$ extended with a sequence of steps ending with executing an action in $T$. We now add $T$ to the end of queue $Q$ and continue the construction. Note that $s''_i$ will be $s_{i+1}$.

We sketch three different scenarios:

1. We reach a point where $Q$ is empty and we either are in a deadlock state or can reach a deadlock state in finitely many steps.

2. We reach a point where $Q$ is empty and then construct an infinite path by taking arbitrary transitions.

3. $Q$ is never empty.

We prove that in each of the cases, the constructed path $\pi$ satisfies fair reachability and is complete.

The first case is trivial, any finite path is fair under each of our fairness assumptions and we have constructed a complete and finite path.

For both the second and third case, it is important to note that if a task $T$ is not reachable from $s_i$ for any $i \geq 0$, then it is also not reachable from any state in $\pi$ after $s_i$. This is the case because the suffix of $\pi$ starting at $s_i$ is a path admitted by $s_i$, if $T$ were reachable from a state on this suffix it would also be reachable from $s_i$. Hence, whenever we remove a task from $Q$ during the construction because we observed it was not reachable, this task will never again be reachable

This means that if we reach a state where $Q$ is empty, even if there are still transitions enabled that are not in any task, we can arbitrarily take transitions without ever seeing a task become enabled again. This means no tasks in $\mathcal{T}$ are perpetually reachable in any suffix of $\pi$ according to the second construction scenario. Since for none of the tasks in $\mathcal{T}$ there is any suffix of $\pi$ in which that task in perpetually reachable, fair reachability is trivially satisfied on $\pi$.

Lastly, there is the third scenario. In this case, we also construct an infinite path. After all, if a deadlock was ever reached then no tasks would be reachable and $Q$ would be empty. Note that we still remove tasks from $Q$ when they become unreachable. We also temporarily remove tasks that are still reachable, but then they are added back in after the path has been extended. Hence, $Q$ eventually contains exactly the set of task that is perpetually reachable on $\pi$. We previously pointed out that, due to the fact we are working with modal $\mu$-calculus formulae and we have assumed *Act* is always finite, we can be sure that $\mathcal{T}$ is finite. Hence, $Q$ will be finite. This means that, once $Q$ has stabilised to exactly those tasks that are perpetually reachable on $\pi$, we will infinitely often select each task in $Q$ to be $T$. Whenever we select a task as $T$, we ensure a transition in that task occurs as part of $\pi$. Hence, every task that is perpetually reachable on $\pi$ will also occur infinitely often. We conclude $\pi$ satisfies fair reachability. □

## A.5 Proof of Theorem 4.12

Similar to the proof of the WFA global response formula, this proof relies on *Act* being finite, and we assume progress. This proof is extremely similar to the WFA proof.

We start with a few relevant propositions.

**Proposition A.8.** *Let $\pi$ be an FRA path, then $s_0 t_1 s_1 \dots t_n \pi$ is an FRA path.*

*Proof.* Let $\pi' = s_0 t_1 s_1 \dots t_n \pi$ be a path, where $\pi$ is an FRA path. Consider an arbitrary suffix $\pi''$ of $\pi'$, and an arbitrary action $\lambda$ that is perpetually reachable in $\pi''$. To prove $\pi'$ is FRA, we must prove that $\lambda$ occurs in $\pi''$. Since $\pi$ is a suffix of $\pi'$, $\pi''$ wil either contain $\pi$ in full or $\pi''$ will also be a suffix of $\pi$.

- If $\pi''$ contains $\pi$, then $\lambda$ is also perpetually reachable in $\pi$. Hence, $\lambda$ is perpetually reachable on any suffix of $\pi$. Because $\pi$ is FRA, we know $\lambda$ must occur in every suffix of $\pi$, meaning $\lambda$ occurs in $\pi$ itself and hence occurs in $\pi''$.

- If $\pi''$ is also a suffix of $\pi$, then since $\pi$ is FRA and $\pi''$ is a suffix of $\pi$ on which $\lambda$ is perpetually reachable, $\lambda$ must occur in $\pi''$.

In both cases, $\lambda$ occurs in $\pi''$. From this we conclude that for every suffix of $\pi$, any action that is perpetually reachable in that suffix must occur in that suffix. Hence, $\pi$ satisfies FRA. □

**Proposition A.9.** *Let $\pi = s_0 t_1 s_1 \ldots$ be a finite or infinite path. If $\pi$ satisfies FRA then also any suffix of $\pi$ satisfies FRA.*

*Proof.* Let $\pi'$ be a suffix of the FRA path $\pi$. Any suffix of $\pi'$ will also be a suffix of $\pi$, hence any action that is perpetually reachable on a suffix of $\pi'$ is also perpetually reachable on a suffix of $\pi$. Since $\pi$ is FRA, any action that is perpetually reachable on a suffix of $\pi$ must occur on that suffix. It follows that any action that is perpetually reachable on a suffix of $\pi'$ must occur on that suffix. Hence, $\pi'$ satisfies FRA. □

We now move on to proving the semantics of Formula 4.6. For this, we start with breaking the formula up into parts.

$$\begin{aligned}
&\mathit{violate}_{\mathit{FRA}} && = \langle \mathit{true}^\star \cdot q\rangle \mathit{invariant}_{\mathit{FRA}}\\
&\mathit{invariant}_{\mathit{FRA}} && = \nu X.(\bigwedge_{\lambda \in \mathit{Act}} (\langle \mathit{true}^\star \cdot \lambda\rangle \mathit{tt} \Rightarrow \mathit{satisfy}_{\mathit{FRA}}(\lambda)))\\
&\mathit{satisfy}_{\mathit{FRA}}(\lambda) && = \mu Y.(([\mathit{true}^\star \cdot \lambda]\mathit{ff} \wedge X) \vee \langle \lambda \setminus r\rangle X \vee \langle \overline{\lambda \cup r}\rangle Y)
\end{aligned}$$

We first prove that $\mathit{satisfy}_{\mathit{FRA}}(\lambda)$ characterises all states that admit an $r$-free path where $\lambda$ occurs or becomes unreachable, and that end in a state that is in the set of states represented by $X$.

**Lemma A.10.** *Let $\epsilon$ be an environment, $s$ a state in $\mathcal{S}$, $\lambda$ an action in $\mathit{Act}$ and $\mathcal{F}$ a subset of $\mathcal{S}$. It holds that $s \in [\![\mathit{satisfy}_{\mathit{FRA}}(\lambda)]\!]_{\epsilon[X:=\mathcal{F}]}$ if, and only if, $s$ admits a finite, possibly partial, path $\pi$ that is $r$-free, ends in a state in $\mathcal{F}$ and either*

1. *$\pi$ is $\lambda$-free and ends in a state where $\lambda$ is unreachable, or*
2. *the last transition of $\pi$ is labelled with $\lambda$ and $\pi$ is otherwise $\lambda$-free.*

*Proof.* First, we apply Theorem 4.3 to determine that $s \in [\![\mu Y.(([\mathit{true}^\star \cdot \lambda]\mathit{ff} \wedge X) \vee \langle \lambda \setminus r\rangle X \vee \langle \overline{\lambda \cup r}\rangle Y)]\!]\epsilon[X := \mathcal{F}]$ if, and only if, $s$ admits a finite, possibly partial path $\pi'$ that ends in a state $s'$ such that only actions in $\overline{\lambda \cup r}$ occur in $\pi'$

and $s' \in [\![([true^\star \cdot \lambda]\mathit{ff} \wedge X) \vee \langle \lambda \setminus r \rangle X]\!]_{\epsilon[X:=\mathcal{F}]}$. Since only actions in $\overline{\lambda \cup r}$ occur in $\pi'$, $\pi'$ is both $\lambda$-free and $r$-free. From $s' \in [\![([true^\star \cdot \lambda]\mathit{ff} \wedge X) \vee \langle \lambda \setminus r \rangle X]\!]_{\epsilon[X:=\mathcal{F}]}$, we know that $s'$ is either a state where $\lambda$ is unreachable and $s' \in \mathcal{F}$, or $s'$ admits a $\lambda$-transition $t$ to a state $s''$ in $\mathcal{F}$ and, $\lambda \neq r$. In the former case, $\pi'$ is a finite, $r$-free path that ends in a state in $\mathcal{F}$, is $\lambda$-free and ends in a state where $\lambda$ is unreachable. In the latter case, $\pi'$ extended with $t$ and $s''$ is a finite, $r$-free path ending in a state in $\mathcal{F}$ such that the last transition is labelled with $\lambda$ and the path is otherwise $\lambda$-free. Hence, the semantics of $\mathit{satisfy}_{FRA}$ are indeed those states that admit a path meeting the requirements stated in the lemma. $\square$

Subsequently, we prove that $\mathit{invariant}_{FRA}$ exactly characterises those states that admit an $r$-free, FRA path. We call the set of all such states $S_{r,FRA}$. For this, we first prove the following lemma:

**Lemma A.11.** *The set of states admitting $r$-free, FRA paths, $S_{r,FRA}$, is a fixed point of the following transformer $T_{FRA}$:*

$$T_{FRA}(\mathcal{F}) = \bigcap_{\lambda \in Act} \{s \in \mathcal{S} \mid s \in [\![\langle true^\star \cdot \lambda \rangle tt]\!]_{\epsilon[X:=\mathcal{F}]} \Rightarrow s \in [\![\mathit{satisfy}_{FRA}(\lambda)]\!]_{\epsilon[X:=\mathcal{F}]}\}$$

*For arbitrary environment $\epsilon$.*

*Proof.* We prove $S_{r,FRA}$ is a fixed point of $T_{FRA}$ by showing that $T_{FRA}(S_{r,FRA}) = S_{r,FRA}$. We prove this through mutual set inclusion of $S_{r,FRA}$ and $T_{FRA}(S_{r,FRA})$.

- Let $s$ be an arbitrary state in $S_{r,FRA}$. We prove $s \in T_{FRA}(S_{r,FRA})$. Since $s \in S_{r,FRA}$, we know $s$ admits an $r$-free, FRA path. Let $\pi$ be such a path starting in $s$. We need to prove $s \in T_{FRA}(\mathcal{F})$ by showing that for all $\lambda \in Act$, if $\lambda$ is reachable from $s$ then $s \in [\![\mathit{satisfy}_{FRA}(\lambda)]\!]_{\epsilon[X:=S_{r,FRA}]}$. Since the condition in the transformer is an implication, $s$ is trivially in the set associated with any action not reachable from $s$. Let $\lambda$ be an arbitrary action that is reachable from $s$. We must show that $s \in [\![\mathit{satisfy}_{FRA}(\lambda)]\!]_{\epsilon[X:=S_{r,FRA}]}$ for this $\lambda$. We do a case distinction on whether $\lambda$ is perpetually reachable on $\pi$.
  - If $\lambda$ is perpetually reachable on $\pi$, then by definition of fair reachability of actions $\lambda$ must occur in $\pi$. Let $s'$ be the first state in $\pi$ reached by a $\lambda$-transition. The path from $s$ to $s'$, $\pi'$ is finite and $r$-free because $\pi$ is $r$-free. Additionally, through our choice of $s'$ we know that none but the last transition on this path are $\lambda$-transitions. To prove $s \in [\![\mathit{satisfy}_{FRA}]\!]_{\epsilon[X:=S_{r,FRA}]}$, it only remains to prove that $s' \in S_{r,FRA}$. We can then apply Lemma A.10.

    By Proposition A.9, the suffix $\pi''$ of $\pi$ starting in $s'$ satisfies FRA, and since $\pi$ is $r$-free so is $\pi''$. Hence $s'$ admits an $r$-free, FRA path and therefore $s' \in S_{r,FRA}$. We conclude $s \in [\![\mathit{satisfy}_{FRA}(\lambda)]\!]_{\epsilon[X:=S_{r,FRA}]}$.

  - If $\lambda$ is not perpetually reachable on $\pi$, then in finitely many steps of $\pi$ a state $s'$ is reached where $\lambda$ is unreachable. None of these steps are $r$-transitions, since $\pi$ is $r$-free. We do a case distinction on whether there is an occurrence of $\lambda$ on $\pi$ before $s'$.
    * If there is such a transition, then we can apply the same argument for $s$ being in $[\![\mathit{satisfy}_{FRA}(\lambda)]\!]_{\epsilon[X:=S_{r,FRA}]}$ as we did in the case that $\lambda$ is perpetually reachable. After all, we only used $\lambda$ being perpetually reachable to conclude it must occur.
    * If there is no occurrence of $\lambda$ before $s'$, then then let $\pi'$ be the prefix of $\pi$ ending in $s'$. Not only is $\pi'$ $r$-free, it is also $\lambda$-free. Additionally, $\lambda$ is unreachable from $s'$. To be able to apply Lemma A.10 to conclude $s \in [\![\mathit{satisfy}_{FRA}(\lambda)]\!]_{\epsilon[X:=S_{r,FRA}]}$, we only need to prove $s' \in S_{r,FRA}$.

      By Proposition A.9, the suffix $\pi''$ of $\pi$ starting in $s'$ satisfies FRA, and since $\pi$ is $r$-free so is $\pi''$. Hence $s'$ admits an $r$-free, FRA path and therefore $s' \in S_{r,FRA}$. Hence, $s \in [\![\mathit{satisfy}_{FRA}(\lambda)]\!]_{\epsilon[X:=S_{r,FRA}]}$.

  We conclude that $s \in [\![\mathit{satisfy}_{FRA}(\lambda)]\!]_{\epsilon[X:=S_{r,FRA}]}$ and hence also that for all actions reachable from $s$, $s$ is in the associated set. We conclude that $s \in T_{FRA}(S_{r,FRA})$.

- Let $s$ be an arbitrary state in $T_{FRA}(S_{r,FRA})$. We prove $s \in S_{r,FRA}$. From $s \in T_{FRA}(S_{r,FRA})$ we know that for all actions $\lambda$ that are reachable from $s$, $s \in [\![\mathit{satisfy}_{FRA}(\lambda)]\!]_{\epsilon[X:=S_{r,FRA}]}$. If there are no actions reachable from $s$ then $s$ is a deadlock state. And if $s$ is a deadlock state, then it admits the path consisting of only itself. This path is trivially $r$-free and, since it is finite, FRA. Hence, we have a witness for $s \in S_{r,FRA}$.

  If $s$ is not a deadlock, let $\lambda$ be an arbitrary action that is reachable from $s$. Then $s \in [\![\mathit{satisfy}_{FRA}(\lambda)]\!]_{\epsilon[X:=S_{r,FRA}]}$. By Lemma A.10, there must be a finite $r$-free path $\pi$ from $s$ to some $s'$ such that $s' \in S_{r,FRA}$. Since $s' \in S_{r,FRA}$ we know $s'$ admits some $r$-free, FRA path $\pi'$. By Proposition A.8, $\pi$ extended with $\pi'$ satisfies FRA and since both $\pi$ and $\pi'$ are $r$-free, so is their combination. Hence $s$ admits an $r$-free, FRA path and therefore $s \in S_{r,FRA}$.

We conclude that the set of all $r$-free, FRA paths $S_{r,FRA}$ is a fixed point of $T_{FRA}$. $\square$

Note that the semantics of $\mathit{invariant}_{FRA}$ is the greatest fixpoint of the transformer $T_{FRA}$. Hence, we need to prove this greatest fixed point equals $S_{r,FRA}$.

**Lemma A.12.** *The set of all $r$-free, FRA paths, $S_{r,FRA}$, is the greatest fixed point of $T_{FRA}$ as defined in Lemma A.11.*

*Proof.* We show that for any $\mathcal{F}$ satisfying $T_{FRA}(\mathcal{F}) = \mathcal{F}$, we have that $\mathcal{F} \subseteq S_{r,FRA}$. Let $\mathcal{F}$ be an arbitrary subset of $\mathcal{S}$ that is a fixed point for $T_{FRA}$. Let $s \in \mathcal{F}$. We prove $s \in S_{r,FRA}$. We do this by constructing an $r$-free, FRA path $\pi$ from $s$. Observe that since $s \in \mathcal{F}$ and $\mathcal{F} = T_{FRA}(\mathcal{F})$, we also have $s \in T_{FRA}(\mathcal{F})$. If there are no reachable actions from $s$, then trivially it admits an $r$-free, WFA path in the path consisting only of $s$, because $s$ is a deadlock. We therefore assume that there are reachable actions from $s$. Let $L$ be the set of all reachable actions in $s$.

Consider an arbitrary but fixed order $<$ on the actions in $L$ and let $\lambda$ be the least of these actions. From $s \in T_{FRA}(\mathcal{F})$ we conclude that there exists some finite, $r$-free path $\pi'$ from $s$ to a state $s'$ such that $s' \in \mathcal{F}$ and either $\lambda$ is not reachable from $s'$ or $\lambda$ is the last action in $\pi'$. We will let $\pi'$ be the start of our constructed path $\pi$.

Denote the set of actions reachable from $s'$ by $L'$. Consider that $L'$ must be a subset of $L$: any action that is reachable from $s'$ would also be reachable from $s$. We next choose the least $\lambda' \in \{\lambda'' \in L' \mid \lambda < \lambda''\}$, so the smallest action that is larger than $\lambda$ and is reachable from both $s$ and $s'$. Since $s' \in \mathcal{F}$ and $\mathcal{F} = T_{FRA}(\mathcal{F})$ we can apply the same construction to find a finite $r$-free path $\pi''$ from $s'$ to $s''$ such that either $\lambda'$ is not reachable $s''$ or $\lambda'$ is the last action in $\pi''$. We add $\pi''$ to our constructed path $\pi$.

We repeat this construction until there are no more reachable actions that are larger than the actions we have already added segments for. This construction will terminate, since there are finitely many actions. Once the construction stops in a state $s_{\mathit{final}}$, we continue extending the path constructed so far with the same construction method, now letting $L$ be the set of actions reachable $s_{\mathit{final}}$. This leads to either an infinite path, or a finite path where there are no actions reachable from the final state.

A path constructed in this manner is trivially $r$-free, since we only ever add path segments that are $r$-free. It is less trivial to see that the resulting path satisfies FRA. Let $\pi_{con}$ be the final path we have constructed. If it is finite, it is trivially FRA. Hence, we henceforth assume $\pi_{con}$ is infinite. Consider a suffix $\pi'_{con}$ of $\pi_{con}$ where an action $\alpha$ is perpetually reachable. Since $\pi'_{con}$ is infinite and our construction procedure is finite until we refresh the set of actions $L$ that need to be satisfied, we know that starting in the first state of $\pi'_{con}$ there will be finitely many steps until we reach a state in which the set of actions $L$ is refreshed. Since $\alpha$ is perpetually reachable in $\pi'_{con}$, $\alpha$ will be part of the new set $L$. During construction of the next stretch of the path, we at some point add a path segment in which $\alpha$ either becomes unreachable or occurs. If $\alpha$ could become unreachable, it would not be perpetually reachable, hence we are sure we have added a path segment in which $\alpha$ occurs. Hence, $\alpha$ occurs in $\pi'_{con}$.

We conclude that, since every action that is perpetually reachable in some suffix of $\pi_{con}$ is guaranteed to occur in that suffix, our constructed path satisfies FRA. We have therefore proven we can construct a path from $s$ which is both $r$-free and satisfies FRA. Hence, $s \in S_{r,FRA}$ and thus $S_{r,FRA}$ is the greatest fixed point of $T_{FRA}$. □

**Lemma A.13.** *For all environments $\epsilon$ and states $s \in \mathcal{S}$, we have that $s \in [\![invariant_{FRA}]\!]_\epsilon$ if, and only if, $s$ admits an $r$-free, FRA path.*

*Proof.* The semantics of $invariant_{FRA}$ is exactly the greatest fixed point of $T_{FRA}$. As we have shown in Lemma A.12 that the greatest fixed point of $T_{FRA}$ is $S_{r,FRA}$, we can conclude that $invariant_{FRA}$ exactly captures those states that admit $r$-free, FRA paths. □

Next, we need to prove that $violate_{WFA}$ indeed characterises the existence of an FRA path that violates global response.

**Lemma A.14.** *For all environments $\epsilon$ and states $s \in \mathcal{S}$, we have that $s \in [\![violate_{FRA}]\!]_\epsilon$ if, and only if, $s$ admits an FRA path that violates global response.*

*Proof.* This follows directly from the definition of $violate_{FRA}$ and Lemma A.13. By the semantics of $violate_{FRA}$, the states $s \in [\![violate_{FRA}]\!]_\epsilon$ are exactly those states that admit a finite, possibly partial, path $\pi$ where the final transition is labelled with a $q$-action, and the subsequent state is in $[\![invariant_{FRA}]\!]_\epsilon$, hence admits a path $\pi'$ that is $r$-free and satisfies WFA. By Proposition A.8, $\pi\pi'$ satisfies FRA. So we have an FRA path violating global response. □

We can now restate and prove Theorem 4.12:

**Theorem A.15.** *A state satisfies Formula 4.6 if, and only if, it satisfies global response under fair reachability of actions.*

*Proof.* The formula Formula 4.6 is the negation of $violate_{FRA}$. By Lemma A.14, a state satisfies $violate_{FRA}$ if, and only if, it admits an FRA path that violates global response. Hence, a state satisfies Formula 4.6 if, and only if, it does not admit such a path. □

## A.6 Proof of Theorem 4.15

The proof for the SFA global response formula differs from the WFA and FRA proofs somewhat, because the formula has such a different structure. However, we still need to prove the same propositions before we can get to the formula itself.

**Proposition A.16.** *Let $\pi$ be an SFA path, then $s_0 t_1 s_1 \ldots t_n \pi$ is an SFA path.*

*Proof.* Let $\pi' = s_0t_1s_1 \dots t_n\pi$ be a path, where $\pi$ is an SFA path. Let $\pi''$ be an arbitrary suffix of $\pi'$ and let $\lambda$ be an action that is relentlessly enabled on $\pi''$. We prove that $\lambda$ occurs in $\pi''$. We do a case distinction on whether $\pi''$ is also a suffix of $\pi$, or $\pi$ is a suffix of $\pi''$.

- If $\pi''$ is a suffix of $\pi$, then since $\pi$ is strongly fair of actions and $\lambda$ is relentlessly enabled on $\pi''$, we know $\lambda$ occurs in $\pi''$.
- If $\pi$ is a suffix of $\pi''$, then since $\lambda$ is relentlessly enabled on $\pi''$ it is also relentlessly enabled on $\pi$. All of $\pi$ is a suffix of $\pi$, and $\pi$ is SFA, so since $\lambda$ is relentlessly enabled on $\pi$ it must occur in $\pi$. And since $\pi$ is a suffix of $\pi''$, $\lambda$ occurs in $\pi''$.

We conclude that $\lambda$ occurs in $\pi''$. Hence, for every suffix of $\pi$, every action that is relentlessly enabled in that suffix occurs in that suffix. Hence, $\pi'$ is strongly fair of actions. □

**Proposition A.17.** *Let $\pi = s_0t_1s_1 \dots$ be a finite or infinite path. if $\pi$ satisfies SFA then also any suffix of $\pi$ satisfies SFA.*

*Proof.* Let $\pi'$ be a suffix of the SFA path $\pi$. Any suffix of $\pi'$ will also be a suffix of $\pi$. Hence, any action $\lambda$ that is relentlessly enabled on a suffix $\pi''$ of $\pi'$ is also relentlessly enabled on that same suffix $\pi''$ of $\pi$. Since $\pi$ is SFA, we know that $\lambda$ must occur in $\pi''$. Hence, any action that is relentlessly enabled on a suffix of $\pi'$ occurs in that suffix, so $\pi'$ is SFA. □

To prove Formula 4.11 indeed expresses that there does not exist an SFA path violating global response, we once again divide the formula up into several smaller parts.

$$\begin{aligned}
&\mathit{violate}_{\mathit{SFA}} && = \langle \mathit{true}^\star \cdot q\rangle\, \mathit{disjunct}_{\mathit{SFA}} \\
&\mathit{disjunct}_{\mathit{SFA}} && = \bigvee_{F \subseteq Act} \mathit{finprefix}_{\mathit{SFA}}(F) \\
&\mathit{finprefix}_{\mathit{SFA}}(F) && = \mu Y.(\langle \overline{r}\rangle\, Y \vee \mathit{satisfy}_{\mathit{SFA}}(F)) \\
&\mathit{satisfy}_{\mathit{SFA}}(F) && = \nu X.(\mathit{inf}_{\mathit{SFA}}(F))
\end{aligned}$$

We also restate the definitions of *inf* and *exec* here, for completeness. Recall that we fix an arbitrary order on the actions in $F$ such that $\alpha_1$ is the first action, $\alpha_2$ the second, etc. and $|F| = n$.

$$\begin{aligned}
&\mathit{inf}_{\mathit{SFA}}(F) && = \mathit{exec}_{\mathit{SFA}}(F, n) \\
&\mathit{exec}_{\mathit{SFA}}(F, 0) && = [\overline{F}]\mathit{ff} \wedge X \\
&\mathit{exec}_{\mathit{SFA}}(F, k+1) && = \mu W_{k+1}.([\overline{F}]\mathit{ff} \wedge \\
& && \qquad (\langle \overline{r}\rangle W_{k+1} \vee \langle \alpha_{k+1} \setminus r\rangle\, \mathit{exec}_{\mathit{SFA}}(F, k)))
\end{aligned}$$

We would like to directly prove the semantics of $\mathit{satisfy}_{\mathit{SFA}}$, but this requires first proving the semantics of the $\mathit{inf}_{\mathit{SFA}}$ function.

**Lemma A.18.** *Let $F$ be an arbitrary set of actions of size $n$, on which an arbitrary order has been fixed so that $\alpha_1$ is the least action, $\alpha_2$ the second least, etc. For an arbitrary environment $\epsilon$, set $\mathcal{F} \subseteq \mathcal{S}$ and state $s \in \mathcal{S}$, $s \in [\![\mathit{inf}_{\mathit{SFA}}(F)]\!]_{\epsilon[X:=\mathcal{F}]}$ if, and only if, $s$ admits a finite, possibly partial, path $\pi$ that is $r$-free and on which all actions in $\overline{F}$ are perpetually disabled. Additionally, there is a subsequence of the transitions of $\pi$ on which every action in $F$ occurs exactly once, in reverse order. Finally, the final state of $\pi$ must be in $\mathcal{F}$, and the final transition of $\pi$ is labelled with $\alpha_1$ if $\alpha_1$ exists.*

The proof for this lemma is highly involved due to the nuances in the way $\mathit{inf}_{\mathit{SFA}}$ is defined. A more general version of this lemma is given as Corollary B.9, which is a corollary to Lemma B.8. In this one case, we do not prove the more specific case here and instead point directly to the generalised case that is given later, simply due to the complexity of the proof.

We use Lemma A.18 to prove the semantics of $\mathit{satisfy}_{\mathit{SFA}}$. For this proof, we define the set of states that admit an $r$-free path on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled to be $S_{r,F,\mathit{SFA}}$.

**Lemma A.19.** *Let $F$ be an arbitrary subset of $\mathit{Act}$ of size $n$, on which there exists an arbitrary ordering such that $\alpha_1$ is the least action, $\alpha_2$ the second-least, etc. and let $\epsilon$ be an arbitrary environment. The set $S_{r,F,\mathit{SFA}}$ is a fixed point of the transformer $T_{F,\mathit{SFA}}$ where*

$$T_{F,\mathit{SFA}}(\mathcal{F}) = \{s \in \mathcal{S} \mid s \in [\![\mathit{inf}_{\mathit{SFA}}(F)]\!]_{\epsilon[X:=\mathcal{F}]}$$

*Proof.* We prove $S_{r,F,\mathit{SFA}}$ is a fixed point of this transformer my proving $S_{r,F,\mathit{SFA}} = T_{F,\mathit{SFA}}(S_{r,F,\mathit{SFA}})$ through mutual set inclusion.

- Let $s$ be an arbitrary state in $S_{r,F,\mathit{SFA}}$, we prove $s \in T_{F,\mathit{SFA}}(S_{r,F,\mathit{SFA}})$. For this, we need to prove that $s \in [\![\mathit{inf}_{\mathit{SFA}}]\!]_{\epsilon[X:=S_{r,F,\mathit{SFA}}]}$. Since $s \in S_{r,F,\mathit{SFA}}$, we know $s$ admits a path $\pi$ that is $r$-free on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled.

  If $n = 0$, then $F = \emptyset$ and $\pi$ is guaranteed to be a path of length 0 containing only $s$ itself, since all actions in $\mathit{Act}$ must be perpetually disabled. In this case, $\pi$ witnesses that $s$ admits a path that is $r$-free, on which all actions in $\overline{F}$ are perpetually disabled and which ends in a state satisfying $S_{r,F,\mathit{SFA}}$. Since $F = \emptyset$, it is also the case that on $\pi$ any condition on all actions in $F$ is vacuously satisfied. By Lemma A.18, $s \in [\![\mathit{inf}_{\mathit{SFA}}]\!]_{\epsilon[X:=S_{r,F,\mathit{SFA}}]}$.

  If $n > 0$, then the proof is a bit more complex. However, $\pi$ can still be used to construct a witness to $s \in [\![\mathit{inf}_{\mathit{SFA}}]\!]_{\epsilon[X:=S_{r,F,\mathit{SFA}}]}$ by Lemma A.18. Consider that since all actions in $F$ occur infinitely often in $\pi$, $\pi$ is infinitely long and there are infinitely many ways to find a subsequence of the transitions in $\pi$ such that all actions in $F$ occur exactly once in the subsequence and do so in reverse order. Take an arbitrary subsequence $l$ that meets these requirements. The last transition in $l$ is a transition $t$ in $\pi$ that is labelled with $\alpha_1$. Let $\pi'$ be the prefix of $\pi$ of which $t$ is the last transition. Then $\pi'$ is $r$-free and every action in $\overline{F}$ is perpetually disabled on $\pi'$, both because $\pi'$ is a prefix of $\pi$. Since $l$ is also a subsequence of $\pi'$, $\pi'$ contains such a subsequence, and since $t$ is the last transition of $\pi'$ the final transition of $\pi'$ is labelled with $\alpha_1$.

  To be able to conclude $s \in [\![inf_{SFA}]\!]_{\epsilon[X:=S_{r,F,SFA}]}$ with $\pi'$ as a witness, we only need to establish that the final state of $\pi'$, $s'$ is in $S_{r,F,SFA}$. The suffix of $\pi$ starting in $s'$, $\pi''$, is a witness to this fact. Hence, we can conclude $s \in [\![inf_{SFA}]\!]_{\epsilon[X:=S_{r,F,SFA}]}$ and thus $s \in T_{F,SFA}(S_{r,F,SFA})$.

- Let $s$ be an arbitrary state in $T_{F,SFA}(S_{r,F,SFA})$. We prove $s \in S_{r,F,SFA}$. From $s \in T_{F,SFA}(S_{r,F,SFA})$ we conclude $s \in [\![inf(F)]\!]_{\epsilon[X:=S_{r,F,SFA}]}$. From Lemma A.18, we can therefore conclude that $s$ admits a path $\pi$ such that $\pi$ is $r$-free, all actions in $\overline{F}$ are perpetually disabled on $\pi$ and the final state of $\pi$, $s'$ is in $S_{r,F,SFA}$. There are other things we know about $\pi$ as well, but these are the only facts we care about.

  Since $s' \in S_{r,F,SFA}$, $s'$ admits a path $\pi'$ that is $r$-free, on which all actions in $F$ occur infinitely often and on which the actions in $\overline{F}$ are perpetually disabled. We use $\pi\pi'$ as a witness for $s \in S_{r,F,SFA}$.

We conclude that $S_{r,F,SFA} = T_{F,SFA}(S_{r,F,SFA})$. □

Since the semantics of $satisfy_{SFA}(F)$ are the greatest fixed point of $T_{F,SFA}$, we still need to prove that $S_{r,F,SFA}$ is the greatest fixed point of this transformer.

**Lemma A.20.** *Let $F$ be an arbitrary subset of Act of size $n$, and $\epsilon$ be an arbitrary environment. Let there be an arbitrary ordering on $F$ and let $\alpha_1$ be the least action of $F$, followed by $\alpha_2$, etc. until $\alpha_n$. The set $S_{r,F,SFA}$ is the greatest fixed point of the transformer $T_{F,BSFA}$, defined in Lemma A.19.*

*Proof.* We prove this by showing that any state in an arbitrary fixed point of $T_{F,SFA}$ is also in $S_{r,F,SFA}$. Let $\mathcal{F}$ be an arbitrary fixed point of $T_{F,BSFA}$ and let $s$ be an arbitrary state in $\mathcal{F}$. We proceed to prove $s \in S_{r,F,BSFA}$. To this end, we must demonstrate that $s$ admits a path $\pi$ that is $r$-free, and on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled.

Since $s \in \mathcal{F}$ and, because $\mathcal{F}$ is a fixed point of $T_{F,SFA}$, $\mathcal{F} = T_{F,BSFA}(\mathcal{F})$, we know that $s \in T_{F,BSFA}(\mathcal{F})$. By definition of $T_{F,SFA}$, we conclude that $s \in$

$[\![ \mathit{inf}_{\mathit{SFA}}(F) ]\!]_{\epsilon[X:=\mathcal{F}]}$. By applying Lemma A.18, we conclude that $s$ must admit a finite, possibly partial, path $\pi'$ such that $\pi'$ is $r$-free, all actions in $\overline{F}$ are perpetually disabled on $\pi'$, all actions in $F$ occur in $\pi'$ at least once, and the final state of $\pi'$ must be in $\mathcal{F}$. There are other things we could conclude about $\pi'$ from Lemma A.18, but we will not need them.

We start our construction of $\pi$ with $\pi'$.

If $F$ is the empty set, then $\pi'$ will consist of only the state $s$ since all states on $\pi'$ are deadlock states because all actions in $\overline{F}$ are perpetually disabled. Consequently, $\pi'$ cannot contain transitions and is only the state $s$, which is a deadlock state. In this case, $\pi = \pi'$ is a witness of $s \in S_{r,F,BSFA}$, since this path is $r$-free by virtue of not having any transitions, and all actions in $F = \emptyset$ occur infinitely often because there are no such actions. Since $s$ is a deadlock state, it is also the case that on all states in $\pi$, all actions in $\overline{F}$ are perpetually disabled.

Henceforth, we assume $F \neq \emptyset$. In this case, $\pi'$ contains at least $n$ transitions since all actions in $F$ occur at least once in $\pi'$. Let $s'$ be the final state of $\pi'$. By definition of $\pi'$, $s' \in \mathcal{F}$. Since $\mathcal{F}$ is a fixed point of the transformer, we also have $s' \in T_{F,SFA,}(\mathcal{F})$ and hence, by definition of the transformer, $s' \in [\![ \mathit{inf}_{\mathit{SFA}}(F) ]\!]_{\epsilon[X:=\mathcal{F}]}$. We can apply Lemma A.18 again to conclude $s'$ admits a path $\pi''$ satisfying all the conditions that must be satisfied by $\pi'$. Append $\pi''$ to the path $\pi$ we have constructed so far.

This construction can be repeated infinitely often: every time we add a partial path to the path $\pi$ we have constructed so far, the final state is in $\mathcal{F}$ and so there is always a next partial path to append. This way, we construct the infinite path $\pi$.

To use $\pi$ as a witness for $s \in S_{r,F,SFA}$, we need to prove $\pi$ is $r$-free, all actions in $\overline{F}$ are perpetually disabled on it, and all actions in $F$ occur infinitely often on it. The first two points follow directly from our construction: every path segment we add meets those two conditions. Additionally, $\pi$ is constructed from infinitely many segments, and on each of those segments all actions in $F$ occurs at least once. Hence, every action in $F$ occurs infinitely often in $\pi$. This proves $s \in S_{BSFA,F}$. $\square$

As noted previously, the semantics of *satisfySFA*$(F)$ are exactly the greatest fixed point of $T_{F,SFA}$. Hence:

**Corollary A.21.** . *For all environments $\epsilon$, sets $F \subseteq \mathit{Act}$ and states $s \in \mathcal{S}$, $s \in [\![ \mathit{satisfy}_{\mathit{SFA}}(F) ]\!]_\epsilon$ if, and only if, $s$ admits an $r$-free path $\pi$ on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled.*

Note that since $F \cup \overline{F} = \mathit{Act}$ and $F \cap \overline{F} = \emptyset$, a path on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled is a path on which any action that is enabled infinitely often must be part of $F$ and therefore also occurs infinitely often. Hence, it is an SFA path.

**Corollary A.22.** *For all environments $\epsilon$, sets $F \subseteq Act$ and states $s \in \mathcal{S}$, $s \in [\![\mathit{satisfy}_{BSFA}(F)]\!]_\epsilon$ if, and only if, $s$ admits an* ***SFA*** *and $\delta_{dis}$-free path $\pi$ on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled.*

We can now finish the remainder of the proof.

**Lemma A.23.** *Let $\epsilon$ be an environment, $s$ a states in $\mathcal{S}$ and $F$ a subset of $Act$. Then $s \in [\![\mathit{finprefix}_{SFA}(F)]\!]_\epsilon$ if, and only if, $s$ admits an $r$-free, finite path ending in a state $s'$ such that $s' \in [\![\mathit{satisfy}_{SFA}(F)]\!]_\epsilon$.*

*Proof.* This follows directly from Theorem 4.3. We use $\phi = \mathit{satisfy}_{BSFA}(F)$ and $\alpha = \overline{r}$. □

**Lemma A.24.** *Let $\epsilon$ be an environment and $s$ a states in $\mathcal{S}$. Then we have $s \in [\![\mathit{disjunct}_{SFA}]\!]_\epsilon$ if, and only if, it admits an $r$-free, SFA path.*

*Proof.* In Corollary A.22, we established that a state satisfies $\mathit{satisfy}_{SFA}(F)$ if, and only if, it admits an $r$-free, SFA path on which the actions in $F$ occur infinitely often and the actions in $Act \setminus F$ are never enabled.

The formula $\mathit{finprefix}_{SFA}(F)$ extends $\mathit{satisfy}_{SFA}(F)$ with a finite-length, $r$-free prefix. By Proposition A.16, we know that an addition of a prefix to an SFA path results in an SFA path. Hence, a state satisfies $\mathit{finprefix}_{SFA}(F)$ if, and only if, it admits an $r$-free, SFA path on which the actions in $F$ occur infinitely often and the actions in $Act \setminus F$ are not relentlessly enabled. The latter set of actions may be enabled during the finite prefix, but are not enabled afterwards.

Consider that any path contains actions that are relentlessly enabled and actions that are not relentlessly enabled. In an SFA path, all relentlessly enabled actions occur infinitely often. Hence, any SFA path can be seen as dividing $Act$ into two sets: a set of actions that occur infinitely often and a set of actions that are not relentlessly enabled. In $\mathit{disjunct}_{SFA}$, all possible ways of dividing $Act$ into these two sets are considered.

We show both directions of this theorem.

- If a state $s$ admits an $r$-free, SFA path $\pi$, then this path induces the set $F$ of actions that are infinitely often taken, and its complement which are actions that are not relentlessly enabled. Then $s$ will satisfy $\mathit{finprefix}_{SFA}(F)$ and hence it will satisfy $\mathit{disjunct}_{SFA}$.

- If a state $s$ satisfies $\mathit{disjunct}_{SFA}$, then there is some $F \subseteq Act$ such that $s$ satisfies $\mathit{finprefix}_{SFA}(F)$. Hence, $s$ admits an $r$-free, SFA path. Specifically, a path along which the actions in $F$ are taken infinitely often and the actions in $Act \setminus F$ are not relentlessly enabled.

Hence, $\mathit{disjunct}_{SFA}$ exactly characterises those states that admit $r$-free, SFA paths. $\square$

**Lemma A.25.** *Let $\epsilon$ be an environment and $s \in \mathcal{S}$. Then $s \in [\![\mathit{violate}_{SFA}]\!]_\epsilon$ if, and only if, it admits an SFA path violating global response.*

*Proof.* By construction, a state $s$ satisfies $\mathit{violate}_{SFA}$ if, and only if, it admits a finite path ending with a $q$-action, to a state $s'$ which satisfies $\mathit{disjunct}_{SFA}$. From Lemma A.24, we know $s'$ therefore admits and $r$-free, SFA path. Hence, $s$ satisfies $\mathit{violate}_{SFA}$ if, and only if, it admits a path $\pi$ that contains a $q$-action, such that the suffix of $\pi$ after that $q$ is an $r$-free, SFA path. By Proposition A.17, $\pi$ is an SFA path. If $s$ admits an SFA path on which a $q$ is not followed by an $r$, then it admits a path violating global response under the assumption of strong fairness of actions. $\square$

We can now restate and prove Theorem 4.15.

**Theorem A.26.** *A state satisfies Formula 4.11 if, and only if, it satisfies global response under strong fairness of actions.*

*Proof.* The formula Formula 4.11 is the negation of $\mathit{violate}_{SFA}$. By Lemma A.25, a state satisfies $\mathit{violate}_{SRA}$ if, and only if, it admits an SFA path that violates global response. Hence, a state satisfies Formula 4.11 if, and only if, it does not admit such a path. $\square$

# Appendix B

# Proofs of Base Formulae

This appendix specifically contains the proofs for the claims made in Section 6.3. These proofs are similar to the proofs we give for the global response formulae, but contain more detailed arguments.

For all proofs, fix the model $M = (\mathcal{S}, s_{init}, Act, Trans)$ Additionally, since for the purposes of the proofs we do not care about what parts of the formulae come from behaviour and what parts from scope, we simplify some of the placeholder variables. In these proofs, we rename $\delta_1 \cdot \delta_2$ to $\delta_{pre}$, $\delta_3$ to $\delta_{en}$ and $\delta_4 \cup \delta_5$ to $\delta_{dis}$.

## B.1 Proof of Theorem 6.1

This proof concerns Formula 6.1.

$$\begin{aligned}
&\neg(\langle \delta_{1} \cdot \delta_{2} \rangle \nu X.( \bigwedge_{\lambda \in Act} (\langle \lambda \rangle tt \Rightarrow ( \\
&\qquad \mu Y.(\langle \delta_{3} \rangle tt \vee ([\lambda] ff \wedge X) \vee \\
&\qquad\qquad \langle \lambda \setminus (\delta_{4} \cup \delta_{5}) \rangle X \vee \langle \overline{\lambda \cup \delta_{4} \cup \delta_{5}} \rangle Y)))))
\end{aligned}$$

After renaming the placeholder variables, this formula becomes:

$$\begin{aligned}
&\neg(\langle \delta_{pre} \rangle \nu X.( \bigwedge_{\lambda \in Act} (\langle \lambda \rangle tt \Rightarrow ( \\
&\qquad \mu Y.(\langle \delta_{en} \rangle tt \vee ([\lambda] ff \wedge X) \vee \langle \lambda \setminus \delta_{dis} \rangle X \vee \langle \overline{\lambda \cup \delta_{dis}} \rangle Y)))))
\end{aligned}$$

We largely follow the same proof approach as we do for the global response formulae. We begin by splitting the formula into multiple parts

$$\begin{aligned}
&\mathit{violate}_{BWFA} &&= \langle \delta_{pre} \rangle \mathit{invariant}_{BWFA} \\
&\mathit{invariant}_{BWFA} &&= \nu X.( \bigwedge_{\lambda \in Act} (\langle \lambda \rangle tt \Rightarrow \mathit{satisfy}_{BWFA}(\lambda))) \\
&\mathit{satisfy}_{BWFA}(\lambda) &&= \mu Y.(\langle \delta_{en} \rangle tt \vee ([\lambda]\mathit{ff} \wedge X) \vee \langle \lambda \setminus \delta_{dis} \rangle X \vee \langle \overline{\lambda \cup \delta_{dis}} \rangle Y)
\end{aligned}$$

We prove that Formula 6.1 captures exactly those states that do not admit WFA paths that meet the following conditions: the path begins with $\delta_{pre}$, after which it is either entirely $\delta_{dis}$-free or there is a $\delta_{dis}$-free sequence which ends in a state where $\delta_{en}$ is enabled, after which arbitrary actions may occur. This is a more informal way of describing the conditions of violating paths we have given in Section 6.2.

We do this in steps. First, we prove that $\mathit{satisfy}_{BWFA}(\lambda)$ characterises all states that allow the $\lambda$ action to be treated fairly or for $\delta_{en}$ to become enabled, without taking any action in $\delta_{dis}$. More formally:

**Lemma B.1.** *For all environments $\epsilon$, states $s \in \mathcal{S}$, actions $\lambda \in Act$ and sets $\mathcal{F} \subseteq \mathcal{S}$, we have that a state $s \in [\![\mathit{satisfy}_{BWFA}(\lambda)]\!]_{\epsilon[X:=\mathcal{F}]}$ if, and only if, $s$ admits a finite, possibly partial, path $\pi$ meeting the following conditions:*

1. *$\pi$ is $\delta_{dis}$-free, and*
2. *at least one of the following conditions is satisfied:*
   (a) *the path ends in a state in which $\delta_{en}$ is enabled and $\pi$ is $\lambda$-free, or*
   (b) *$\pi$ ends in a state satisfying $\mathcal{F}$ in which $\lambda$ is disabled and $\pi$ is $\lambda$-free, or*
   (c) *the last transition in $\pi$ is a $\lambda$-transition, this is the only $\lambda$-transition on $\pi$, and the last state satisfies $\mathcal{F}$.*

*Proof.* For this proof, we make use of Theorem 4.3, with $\phi = \langle \delta_{en} \rangle tt \vee ([\lambda]\mathit{ff} \wedge X) \vee \langle \lambda \setminus \delta_{dis} \rangle X$ and $\alpha = \overline{\lambda \cup \delta_{dis}}$. From the theorem, we conclude

$$s \in [\![\mu Y.(\langle \delta_{en} \rangle tt \vee ([\lambda]\mathit{ff} \wedge X) \vee \langle \lambda \setminus \delta_{dis} \rangle X \vee \langle \overline{\lambda \cup \delta_{dis}} \rangle Y)]\!]_{\epsilon[X:=\mathcal{F}]}$$

if, and only if, $s$ admits a finite, possibly partial path $\pi'$ meeting the following conditions:

3. on $\pi'$, only actions in $\overline{\lambda \cup \delta_{dis}}$ occur, and

4. $\pi'$ ends in a state $s'$ in $[\![\langle\delta_{en}\rangle tt \vee ([\lambda]\mathit{ff} \wedge X) \vee \langle\lambda \setminus \delta_{dis}\rangle X]\!]_{\epsilon[X:=\mathcal{F}]}$, which is equivalent to $s' \in [\![\langle\delta_{en}\rangle tt]\!]_{\epsilon[X:=\mathcal{F}]} \vee s' \in [\![[\lambda]\mathit{ff} \wedge X]\!]_{\epsilon[X:=\mathcal{F}]} \vee s' \in [\![\langle\lambda \setminus \delta_{dis}\rangle X]\!]_{\epsilon[X:=\mathcal{F}]}$.

We now prove that $s$ admits such a path $\pi'$ if, and only if, it admits a path $\pi$ that satisfies 1 and 2.

First, we assume $s$ admits a path $\pi'$ satisfying 3 and 4 and prove it then also admits a path $\pi$ satisfying 1 and 2. From 4, we get that $s'$ is in at least one of three sets. How we construct $\pi$ depends on which of the three sets $s'$ is in.

- If $s'$ is in $[\![\langle\delta_{en}\rangle tt]\!]_{\epsilon[X:=\mathcal{F}]}$ then $\pi = \pi'$. In this case, since $\pi'$ is free from $\delta_{dis}$-actions and $\lambda$, due to 3, so is $\pi$. Additionally, since $\delta_{en}$ is enabled in the final state of $\pi'$, it is also enabled in the final state of $\pi$. Therefore, $\pi$ meets the conditions 1 and 2a.

- If $s'$ is in $[\![[\lambda]\mathit{ff} \wedge X]\!]_{\epsilon[X:=\mathcal{F}]}$, then we can conclude $\lambda$ is disabled in $s'$ and $s' \in \mathcal{F}$. We take $\pi = \pi'$. Since $\pi'$ is free from $\delta_{dis}$-actions and $\lambda$, so is $\pi$. Additionally, since the final state of $\pi'$ satisfies $\mathcal{F}$ and $\lambda$ is disabled in the final state of $\pi'$, the same holds for $\pi$. Hence, $\pi$ meets the conditions 1 and 2b.

- If $s'$ is in $[\![\langle\lambda \setminus \delta_{dis}\rangle X]\!]_{\epsilon[X:=\mathcal{F}]}$ then $s'$ admits a transition $t$ to a state $s''$ such that $\mathit{action}(t) = \lambda$ and $\mathit{action}(t) \notin \delta_{dis}$ and $s'' \in \mathcal{F}$. We let $\pi = \pi' t s''$. Since $\pi'$ is free from $\delta_{dis}$-actions, and $\mathit{action}(t) \notin \delta_{dis}$, $\pi$ is $\delta_{dis}$-free. Additionally, the last transition of $\pi$ is labelled with $\lambda$ and its last state satisfies $\mathcal{F}$. Finally, since $\pi'$ is $\lambda$-free, $t$ is the only $\lambda$-transition in $\pi$. We conclude that $\pi$ meets the conditions 1 and 2c.

In all three cases, we can construct a path $\pi$ that meets conditions 1 and at least one of the options from 2. We conclude $s$ admits such a path.

The other way around, we assume $s$ admits a path $\pi$ satisfying 1 and 2 and prove it also admits a path $\pi'$ satisfying 3 and 4. We know $\pi$ is $\delta_{dis}$-free from 1, we do a case distinction on which condition from 2 holds:

- If 2a holds on $\pi$, then $\pi$ is also $\lambda$-free and $\delta_{en}$ is enabled in the last state of $\pi$, $s'$. We take $\pi' = \pi$. Since $\pi$ is $\delta_{dis}$-free and $\lambda$-free, so is $\pi'$ and 3 is satisfied. Since $\delta_{en}$ is enabled in $s'$, $s' \in [\![\langle\delta_{en}\rangle tt]\!]_{\epsilon[X:=\mathcal{F}]}$ and hence $\pi'$ satisfies 4.

- If 2b holds on $\pi$, then $\pi$ is also $\lambda$-free, and in the last state $s'$ of $\pi$, $\lambda$ is disabled and $s' \in \mathcal{F}$. This means $s' \in [\![[\lambda]\mathit{ff} \wedge X]\!]_{\epsilon[X:=\mathcal{F}]}$. We take $\pi' = \pi$, and because $\pi$ satisfies 3 and 4, so does $\pi'$.

- If 2c holds on $\pi$, then the last transition of $\pi$, $t$ is a $\lambda$-transition. Because $\pi$ is $\delta_{dis}$-free, we know $\lambda \notin \delta_{dis}$. Additionally, we know from 2c that the last state of $\pi$, $s'$, satisfies $\mathcal{F}$. Let $s'' = \mathit{source}(t)$, then we know $s'' \in [\![\langle \delta \setminus \delta_{dis} \rangle X]\!]_{\epsilon[X:=\mathcal{F}]}$. The prefix of $\pi$ ending in $s''$ is then our candidate for $\pi'$. Since $t$ is the only $\lambda$-transition on $\pi$, $\pi'$ is both $\delta_{dis}$ and $\lambda$-free, meaning it satisfies 3. Additionally, since $s''$ is the last state of $\pi'$, it also satisfies 4.

We conclude that if $s$ admits a path satisfying 1 and 2, it also admits a path satisfying 3 and 4.

We conclude that, since $s \in [\![\mathit{satisfy}_{BWFA}(\lambda)]\!]_{\epsilon[X:=\mathcal{F}]} \Leftrightarrow s$ admits a finite, possibly partial, path satisfying 3 and 4 $\Leftrightarrow s$ admits a finite, possibly partial, path satisfying 1 and 2, the lemma holds. □

The meaning of *satisfy*$_{BWFA}$ when expressed like this, may be somewhat esoteric. It makes more sense when combined with *invariant*$_{BWFA}$. Let $S_{BWFA}$ be the set of states that admits complete, WFA paths that are either entirely $\delta_{dis}$-free or contain a prefix which is $\delta_{dis}$-free and ends in a state where $\delta_{en}$ is enabled. We prove that *invariant*$_{BWFA}$ captures exactly the set $S_{BWFA}$.

To do this, we first prove $S_{BWFA}$ is a fixed point for the transformer belonging to the semantics of *invariant*$_{BWFA}$.

**Lemma B.2.** *The set $S_{BWFA}$ is a fixed point of the transformer $T_{BWFA}$, where*

$$T_{BWFA}(\mathcal{F}) = \bigcap_{\lambda \in Act} \{s \in \mathcal{S} \mid s \in [\![\langle \lambda \rangle tt]\!]_{\epsilon[X:=\mathcal{F}]} \Rightarrow s \in [\![\mathit{satisfy}_{BWFA}(\lambda)]\!]_{\epsilon[X:=\mathcal{F}]}\}$$

*For an arbitrary environment $\epsilon$.*

*Proof.* $S_{BWFA}$ being a fixed point of $T_{BWFA}$ means that $T_{BWFA}(S_{BWFA}) = S_{BWFA}$. We prove this through mutual set inclusion.

- Let $s$ be an arbitrary state in $\mathcal{S}$. We assume $s \in S_{BWFA}$ and prove $s \in T_{BWFA}(S_{BWFA})$. To prove $s \in T_{BWFA}(S_{BWFA})$ we need to prove that, for all $\lambda \in Act$, if $\lambda$ is enabled in $s$ then $s \in [\![\mathit{satisfy}_{BWFA}(\lambda)]\!]_{\epsilon[X:=S_{BWFA}]}$. Let $\lambda$ be an arbitrary action in $Act$. If $\lambda$ is not enabled in $s$, the implication trivially holds. Hence, we assume $\lambda$ is enabled in $s$. We need to prove that $s \in [\![\mathit{satisfy}_{BWFA}(\lambda)]\!]_{\epsilon[X:=S_{BWFA}]}$. Using Lemma B.1, we know this is the case if $s$ admits a finite, possibly partial, path $\pi$ such that

  1. $\pi$ is $\delta_{dis}$-free, and
  2. at least one of the following conditions is satisfied:
     (a) the path ends in a state in which $\delta_{en}$ is enabled and $\pi$ is $\lambda$-free, or
     (b) $\pi$ ends in a state satisfying $S_{BWFA}$ in which $\lambda$ is disabled and $\pi$ is $\lambda$-free, or

(c) the last transition in $\pi$ is a $\lambda$-transition, this is the only $\lambda$-transition on $\pi$, and the last state satisfies $S_{BWFA}$.

We therefore only need to prove such a path $\pi$ indeed exists. From $s \in S_{BWFA}$ we know that $s$ admits a complete, WFA path $\pi_{WFA}$ that is either entirely $\delta_{dis}$-free or has a prefix $\pi_{pre}$ that is $\delta_{dis}$-free and ends in a state $s_{pre}$ such that $\delta_{en}$ is enabled in $s_{pre}$. We do a case distinction on whether $\lambda$ is perpetually enabled on $\pi_{WFA}$ or not.

- If $\lambda$ is perpetually enabled, then since $\pi_{WFA}$ satisfies weak fairness of actions it must be the case that $\lambda$ occurs on $\pi_{WFA}$. Let $t$ be the first transition that is part of $\pi_{WFA}$ which is labelled with $\lambda$. We do a case distinction on whether any state in $\pi_{WFA}$ before the execution of $t$ there is a state $s_{en}$ on which $\delta_{en}$ is enabled.
  - If such a state $s_{en}$ exists, then $\pi_{WFA}$ has a prefix $\pi$ such that $s_{en}$ is the last state of $\pi$ and, because $s_{en}$ by assumption comes before the first transition labelled with $\lambda$, $\pi$ is $\lambda$-free. Since $\pi_{WFA}$ is $\delta_{dis}$-free, so is $\pi$. We can therefore say that $\pi$ satisfies 1 and 2a.
  - If no such state $s_{en}$ exists, we instead construct a path that satisfies 2c. Recall that $t$ is the first $\lambda$-transition on $\pi_{WFA}$, let $\pi$ be the prefix of $\pi_{WFA}$ where $t$ is the last transition. Since $\pi_{WFA}$ is $\delta_{dis}$-free, so is $\pi$ and 1 is satisfied. We know that $t$ is the last transition of $\pi$, and since it is the first $\lambda$-transition on $\pi_{WFA}$ we also know that $\pi$ is $\lambda$-free with the exception of $t$. To establish $\pi$ satisfies 2c, it only rests to show that $target(t) \in S_{BWFA}$.
    To do this, we need to establish that $target(t)$ admits a complete, WFA path that is either entirely $\delta_{dis}$-free or contains a $\delta_{dis}$-free prefix that ends in a state where $\delta_{en}$ is enabled. The suffix of $\pi_{WFA}$ starting in $target(t)$ is such a path: from Proposition 4.5 we know that a suffix of a WFA path is WFA, and since no state $s_{en}$ exists before $t$ on $\pi_{WFA}$, we are either dealing with a fully $\delta_{dis}$-free path, or we are still in the $\delta_{dis}$-free prefix. Either way, the suffix of $\pi_{WFA}$ starting in $target(t)$ is a witness to the fact that $target(t) \in S_{BWFA}$ and hence we can conclude that 2c is satisfied by the prefix of $\pi_{WFA}$ ending in $target(t)$.
- If $\lambda$ is not perpetually enabled on $\pi_{WFA}$ then there are states on $\pi_{WFA}$ in which $\lambda$ is not enabled. Let $s_{not}$ be the first of those states. We do a case distinction on whether $\lambda$ occurs on $\pi_{WFA}$ before $s_{not}$ is reached.
  - If $\lambda$ occurs before $s_{not}$ is reached, then we can repeat the argument for the case where $\lambda$ is perpetually enabled on $\pi_{WFA}$. In this

argument, the only thing we use $\lambda$ being perpetually enabled for is to conclude $\lambda$ must occur. Since in this case $\lambda$ occurs anyway, even though it is not perpetually enabled, we can apply the same argument to conclude we can construct a path $\pi$ satisfying 1 and 2.

∗ If $\lambda$ does not occur before $s_{not}$ in $\pi_{WFA}$, we know the prefix of $\pi_{WFA}$ before $s_{not}$ is $\lambda$-free. We need to make one more case distinction on whether there is a state $s_{en}$ before $s_{not}$ on which $\delta_{en}$ is enabled.

  · If such a state $s_{en}$ exists, then $\pi_{WFA}$ has a prefix $\pi$ such that $s_{en}$ is the last state of $\pi$ and, because we assumed that there is no occurrence of $\lambda$ before $s_{not}$ and therefore also not before $s_{en}$, $\pi$ is $\lambda$-free. Since $\pi_{WFA}$ is $\delta_{dis}$-free, so is $\pi$. Hence, $\pi$ satisfies 1 and 2a.

  · If no such state $s_{en}$ exists, we construct a path that satisfies 1 and 2b. For this, let $\pi$ be the prefix of $\pi_{WFA}$ ending in $s_{not}$. Since $\pi_{WFA}$ is $\delta_{dis}$-free, so is $\pi$. Hence, $\pi$ satisfies 1. Additionally, we have assumed $\lambda$ is disabled in $s_{not}$, and we have established the prefix of $\pi_{WFA}$ before $s_{not}$ is $\lambda$-free. To establish $\pi$ satisfies 2b, we only need to show that $s_{not}$ is in $S_{BWFA}$.
  This requires us to show that $s_{not}$ admits a complete, WFA path which is either entirely $\delta_{dis}$-free, or contains a $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled. We know $s_{not}$ admits such a path because it admits the suffix of $\pi_{WFA}$ starting in $s_{not}$. Using Proposition 4.5, we know that the suffix of a WFA path is WFA. Additionally, by assumption $\delta_{en}$ has not been enabled yet when we get to $s_{not}$, so either $\pi_{WFA}$ is entirely $\delta_{dis}$-free, in which case its suffix is as well, or when we get to $s_{not}$ we are still in the $\delta_{dis}$-free prefix, in which case its suffix still has such a prefix. Either way, the suffix of $\pi_{WFA}$ starting in $s_{not}$ is a witness to the fact that $s_{not} \in S_{BWFA}$ and hence we can conclude that 2c is satisfied by the prefix of $\pi_{WFA}$ ending in $s_{not}$.

We have shown in every case that $s$ admits a path $\pi$ that satisfies 1 and 2. Using Lemma B.1, we conclude that $s \in [\![\mathit{satisfy}_{BWFA}(\lambda)]\!]_{\epsilon[X:=S_{BWFA}]}$. Hence, we have shown that for every arbitrary action $\lambda$, if $\lambda$ is enabled in $s$ then $s \in [\![\mathit{satisfy}_{BWFA}(\lambda)]\!]_{\epsilon[X:=S_{BWFA}]}$, from which we conclude that $s \in T_{BWFA}(S_{BWFA})$.

- Let $s$ be an arbitrary state in $\mathcal{S}$. We assume $s \in T_{BWFA}(S_{BWFA})$ and prove $s \in S_{BWFA}$. From our assumption, we know that for every action $\lambda \in \mathit{Act}$ that is enabled in $s$, $s \in [\![\mathit{satisfy}_{BWFA}(\lambda)]\!]_{\epsilon[X:=S_{BWFA}]}$. We use this to prove that $s \in S_{BWFA}$. To do this, we need to construct a path $\pi$ starting in $s$ such that $\pi$ is a complete, WFA path that is either entirely $\delta_{dis}$-free or has a prefix that is $\delta_{dis}$-free which ends in a state where $\delta_{en}$ is enabled. First, we do a case distinction on whether $s$ is a deadlock state.

– If $s$ is a deadlock state, then $s$ admits the complete path $\pi = s$. This path is trivially WFA since it is finite, and it is trivially $\delta_{dis}$-free since it contains no transitions at all. Hence, this $\pi$ is a witness for $s$ being in $S_{BWFA}$.
– If $s$ is not a deadlock state, there must be at least one action enabled in $s$. Let $\lambda$ be an arbitrary action that is enabled in $s$. Since $\lambda$ is enabled in $s$, we know that $s \in [\![ \mathit{satisfy}_{BWFA}(\lambda) ]\!]_{\epsilon[X:=S_{BWFA}]}$. Using Lemma B.1, this is sufficient to conclude that $s$ admits a finite, possibly partial, path $\pi'$ such that
  1. $\pi'$ is $\delta_{dis}$-free, and
  2. at least one of the following conditions is satisfied:
     (a) the path ends in a state in which $\delta_{en}$ is enabled and $\pi'$ is $\lambda$-free, or
     (b) $\pi'$ ends in a state satisfying $S_{BWFA}$ in which $\lambda$ is disabled and $\pi'$ is $\lambda$-free, or
     (c) the last transition in $\pi'$ is a $\lambda$-transition, this is the only $\lambda$-transition on $\pi'$, and the last state satisfies $S_{BWFA}$.

  We use $\pi'$ to construct $\pi$. For this, we need to do a case distinction on which of the options in 2 holds for $\pi'$.

  ∗ If the path $\pi'$ satisfies 2a, then $\pi'$ is a finite path ending in a state on which $\delta_{en}$ is enabled. Either $\pi'$ is a complete path, in which case it is trivially WFA because it is finite, or it is a partial path. If it is partial, then because WFA is feasible, it can be extended to a complete WFA path. We therefore know there exists a complete, WFA path $\pi$ of which $\pi'$ is a prefix. Since $\pi'$ is $\delta_{dis}$-free, due to 1, and ends in a state where $\delta_{en}$ is enabled, we know that $\pi$ contains a $\delta_{dis}$-free prefix which ends in a state where $\delta_{en}$ is enabled. So $\pi$ is a witness for $s$ being in $S_{BWFA}$.
  ∗ If the path $\pi'$ satisfies 2b or 2c, then $\pi'$ ends in a state satisfying $S_{BWFA}$. We do not need any other information from these two cases, so we address them both simultaneously. Let $s'$ be the final state of $\pi'$. Since $s' \in S_{BWFA}$, we know $s'$ admits a path $\pi''$ that is complete, WFA and either entirely $\delta_{dis}$-free or has a $\delta_{dis}$-free prefix which ends in a state where $\delta_{en}$ is enabled. We let $\pi = \pi'\pi''$. From

Proposition 4.4, we know that since $\pi''$ is WFA and $\pi'$ is finite, $\pi'\pi''$ is WFA. Additionally, since $\pi'$ is $\delta_{dis}$-free, due to 1, we can prepend $\pi'$ to $\pi''$ while preserving that the resulting path is either entirely $\delta_{dis}$-free or contains a $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled. Hence, $\pi = \pi'\pi''$ is a witness for $s$ being in $S_{BWFA}$.

In all three cases, we can construct a $\pi$ that witnesses $s \in S_{BWFA}$.

We conclude that if $s \in T_{BWFA}(S_{BWFA})$ then $s \in S_{BWFA}$.

Through mutual set inclusion, we have proven $T_{BWFA}(S_{BWFA}) = S_{BWFA}$, hence $S_{BWFA}$ is a fixed point of the transformer $T_{BWFA}$. □

To prove that $S_{BWFA} = [\![\mathit{invariant}_{BWFA}]\!]_\epsilon$, we still need to show that $S_{BWFA}$ is the *greatest* fixed point of the transformer $T_{BWFA}$.

**Lemma B.3.** *The set $S_{BWFA}$ is the greatest fixed point of $T_{BWFA}$ as defined in Lemma B.2.*

*Proof.* We show that for any $\mathcal{F} \subseteq \mathcal{S}$ satisfying $T_{BWFA}(\mathcal{F}) = \mathcal{F}$, it is the case that $\mathcal{F} \subseteq S_{BWFA}$. Let $\mathcal{F}$ be an arbitrary subset of $\mathcal{S}$ that is a fixed point for $T_{BWFA}$. Let $s$ be an arbitrary state in $\mathcal{F}$. We prove that $s \in S_{BWFA}$ by constructing a complete WFA path $\pi$ starting in $s_0 = s$ that is either entirely $\delta_{dis}$-free or contains a $\delta_{dis}$-free prefix which ends in a state where $\delta_{en}$ is enabled.

Observe that since $s \in \mathcal{F}$ and $\mathcal{F} = T_{BWFA}(\mathcal{F})$, we also have $s \in T_{BWFA}(\mathcal{F})$. From this we know that for all actions $\lambda$ that are enabled in $s$, it is the case that $s \in [\![\mathit{satisfy}_{BWFA}(\lambda)]\!]_{\epsilon[X:=\mathcal{F}]}$. Let $L_0$ be the set of actions that are enabled in $s$. If $L_0$ is empty, then $s$ is a deadlock state. In that case, the path consisting of only the state $s$ serves as a witness for $s$ being in $S_{BWFA}$: it is a complete WFA path which is entirely $\delta_{dis}$-free. We henceforth assume that $L_0$ is not empty.

Consider an arbitrary but fixed order $<$ on $L_0$, and let $\lambda_0$ be the least action in $L_0$ by this order. Since $\lambda_0$ is an action that is enabled in $s$, we know that $s \in [\![\mathit{satisfy}_{BWFA}(\lambda_0)]\!]_{\epsilon[X:=\mathcal{F}]}$. By Lemma B.1, we know that $s$ admits a finite, possibly partial, path $\pi_0$ that meets the following requirements:

1. $\pi_0$ is $\delta_{dis}$-free, and
2. at least one of the following conditions is satisfied:
    (a) the path ends in a state in which $\delta_{en}$ is enabled and $\pi_0$ is $\lambda_0$-free, or
    (b) $\pi_0$ ends in a state satisfying $\mathcal{F}$ in which $\lambda_0$ is disabled and $\pi_0$ is $\lambda$-free, or
    (c) the last transition in $\pi_0$ is a $\lambda_0$-transition, this is the only $\lambda_0$-transition on $\pi_0$, and the last state satisfies $\mathcal{F}$.

We let $\pi_0$ be the start of our path $\pi$. How we continue constructing $\pi$ depends on which condition holds for 2. We do a case distinction on whether 2a holds.

- If 2a holds, then we stop the algorithm for construction here. Instead of further considering enabled actions, we simply arbitrarily extend $\pi$ to a complete WFA path. We know this is possible, because WFA is feasible. In this case, the $\pi$ we have constructed is a complete WFA path with a $\delta_{dis}$-free prefix which ends in a state where $\delta_{en}$ is enabled: this prefix is $\pi_0$.

- If 2a does not hold, then either 2c or 2b holds. Either way, $\pi_0$ ends in a state $s_1$ with $s_1 \in \mathcal{F}$. We continue construction of $\pi$ from $s_1$. Let $L_1$ be the set of actions enabled in $s_1$. Let $\lambda_1$ be the least action in $L_0 \cap L_1$ such that $\lambda_0 < \lambda_1$. Since $\mathcal{F} = T_{BWFA}(\mathcal{F})$, $s_1 \in T_{BWFA}(\mathcal{F})$. Hence, since $\lambda_1$ is enabled in $s_1$, we know that $s_1 \in [\![ \mathit{satisfy}_{BWFA}(\lambda_1) ]\!]_{\epsilon[X:=\mathcal{F}]}$.

Assuming we were in the second scenario, and so did not terminate the construction algorithm, we can now repeat the construction we have done with $s_0$ for $s_1$. We know a path $\pi_1$ exists that meets the condition that it is $\delta_{dis}$-free and either ends in a state where $\delta_{en}$ is enabled, or in a state $s_2$ with $s_2 \in \mathcal{F}$. In both cases, we append $\pi_1$ to the path $\pi$ we are constructing. In the first case, where $\pi$ now ends in a state where $\delta_{en}$ is enabled, we arbitrarily extend $\pi$ to a complete WFA path and abort the algorithm. In the latter case, we continue the construction from $s_2$, now taking the next smallest action $\lambda_2$ of the actions that are enabled in $s_0$, $s_1$ and $s_2$.

We repeat the above construction until there is no more action we can choose, i.e when $L_0 \cap L_1 \cap L_2 \cap \ldots \cap L_k$ no longer contains any action greater than $\lambda_{k-1}$. This will eventually happen, since there are only finitely many actions in our system. Let $s_k$ be the final state of the $\pi$ we have constructed so far. We now continue the construction from $s_k$ by restarting the whole construction procedure, but now using $s_k$ in place of $s_0$. So we take $L_k$, the set of actions enabled in $s_k$, and simply take $\lambda_k$ to be the least action in $L_k$ according to $<$. We call this "refreshing the set of actions" in the remainder of this proof.

This procedure continues until either it is forced to terminate, because we reach a deadlock state or a state where $\delta_{en}$ is enabled, or it continues infinitely. We argue that in all three cases, the path $\pi$ we construct is a complete WFA path that is either entirely $\delta_{dis}$-free or contains a $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled.

- If the construction terminates because we reach a state where $\delta_{en}$ is enabled, then we have finished constructing $\pi$ by extending it to an arbitrary complete WFA path. Since every path segments we added to the construction of $\pi$ during the algorithm was $\delta_{dis}$-free, and $\delta_{en}$ is enabled in the last state before our arbitrary extension, we have a path here that has a $\delta_{dis}$-free prefix which ends in a state where $\delta_{en}$ is enabled.

- If the construction terminates because we reach a deadlock state, then we have a complete finite path, which is WFA because all complete finite paths are WFA. Additionally, we have constructed it only from segments we found via application of Lemma B.1, which were all $\delta_{dis}$-free. Hence, we have constructed a complete WFA path that is entirely $\delta_{dis}$-free.

- If the construction never terminates, $\pi$ is infinite. The construction only adds segments that are $\delta_{dis}$-free, because all segments come from applications of Lemma B.1, so $\pi$ is $\delta_{dis}$-free. Additionally, $\pi$ is infinite so it is complete. The only thing we still need to prove is that $\pi$ is WFA. Consider an arbitrary suffix $\pi'$ of $\pi$, and consider an arbitrary action $\lambda$ that is perpetually enabled on $\pi'$. To prove $\pi$ is WFA, we need to prove that $\lambda$ occurs in $\pi'$.

  We know that the construction algorithm until we refresh the set of actions is finite, hence it occurs infinitely often while constructing the infinite path $\pi$. Let $s_r$ be the first state in $\pi'$ for which we refresh the set of actions when constructing $\pi$. Since $\lambda$ is perpetually enabled on $\pi'$, and $sr$ is part of $\pi'$, $\lambda$ is enabled in $s_r$. Hence, $\lambda \in L_r$. On every state in $\pi'$, $\lambda$ is enabled because $\lambda$ is perpetually enabled on $\pi'$. Hence, every time a restriction is added to the set of actions we consider through intersecting with the set of actions enabled in the next state, $\lambda$ remains in the set. We always take the next smallest action in the set, and since *Act* is finite this means we will eventually choose $\lambda$ as the action we use for constructing the next segment of $\pi$.

  So at some point during the constructing of $\pi$ after $s_r$, we add a path segment on which either $\delta_{en}$ is enabled, or $\lambda$ becomes disabled, ór $\lambda$ occurs. In the first case, the construction would terminate and we assumed that it did not. In the second case, $\lambda$ would not be perpetually enabled on $\pi'$. Hence, we must be in the last case: the segment we add to $\pi$ must be one in which $\lambda$ occurs. Thus $\lambda$ occurs in $\pi'$.

  Since for every suffix of $\pi$, every action that is perpetually enabled on that suffix also occurs in that suffix, $\pi$ is weakly fair of actions. Thus, $\pi$ is a complete WFA path which is entirely $\delta_{dis}$-free.

We have shown that for an arbitrary state $s$ in an arbitrary fixed point $\mathcal{F}$ of $T_{BWFA}$, we can construct a path starting in $s$ that witnesses $s \in S_{BWFA}$. We conclude that $S_{BWFA}$ is the greatest fixed point of $T_{BWFA}$. $\square$

The semantics of $invariant_{BWFA}$ are exactly the greatest fixed point of $T_{BWFA}$, hence we conclude the following.

**Corollary B.4.** *The set of states characterised by $invariant_{BWFA}$ is exactly the set $S_{BWFA}$.*

The difficult part of the proof is now done, since $violate_{BWFA}$ only adds a prefix, and then to get Formula 6.1 we simply negate $violate_{BWFA}$.

**Lemma B.5.** *For all environments $\epsilon$ and states $s \in \mathcal{S}$, $s \in [\![violate_{BWFA}]\!]_\epsilon$ if, and only if, $s$ admits a complete WFA path that starts with $\delta_{pre}$, after which it is either entirely $\delta_{dis}$-free or has a finite $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled.*

*Proof.* This follows from Corollary B.4 and the definition of $violate_{BWFA}$. The semantics of $\langle\delta_{pre}\rangle invariant_{BWFA}$ is exactly those states that admits paths that have a prefix $\delta_{pre}$ that ends in a state that satisfies $invariant_{BWFA}$. By Corollary B.4, states satisfying $invariant_{BWFA}$ admit complete WFA paths that are either entirely $\delta_{dis}$-free or have a finite $\delta_{dis}$-free prefix which ends in a state where $\delta_{en}$ is enabled. The addition of $\langle\delta_{pre}\rangle$ to the formula simply ensures there is a prefix exactly matching $\delta_{pre}$ prepended to the path. By Proposition 4.4, the resulting path is still WFA. □

We now restate Theorem 6.1:

**Theorem B.6.** *A state $s \in \mathcal{S}$ satisfies Formula 6.1 if, and only if, it does not admit a path $\pi$ meeting the following requirements:*

1. *$\pi$ is complete and satisfies weak fairness of actions, and*
2. *there is a prefix of $\pi$ that matches $\delta_1 \cdot \delta_2$, and*
3. *there is no occurrence of any action in $\delta_4$ or $\delta_5$ in $\pi$ after $\delta_1 \cdot \delta_2$ and before the first occurrence of an action in $\delta_3$.*

*Proof.* This follows directly from Lemma B.5 and the fact that Formula 6.1 is the negation of $violate_{BWFA}$. Recall that for this proof, we have renamed $\delta_1 \cdot \delta_2$ to $\delta_{pre}$, $\delta_3$ to $\delta_{en}$ and $\delta_4 \cup \delta_5$ to $\delta_{dis}$. □

## B.2 Proof of Theorem 6.2

We need to prove that Formula 6.2:

$$\neg(\langle\delta_1 \cdot \delta_2\rangle(\bigvee_{F \subseteq Act}(\mu Y.(\langle\overline{\delta_4 \cup \delta_5}\rangle Y \vee \langle\delta_3\rangle tt \vee \nu X.inf(F)))))$$

Where $\mathit{inf}(F)$ is defined as follows. We fix an arbitrary order on the actions in $F$ such that $\alpha_1$ is the first action, $\alpha_2$ the second, etc. Let $n = |F|$.

$$\begin{aligned}
\mathit{inf}(F) &= \mathit{exec}(F, n)\\
\mathit{exec}(F, 0) &= [\overline{F}]\mathit{ff} \wedge X\\
\mathit{exec}(F, k+1) &= \mu W_{k+1}.([\overline{F}]\mathit{ff} \wedge\\
&\qquad (\langle\overline{\delta_4 \cup \delta_5}\rangle W_{k+1} \vee \langle \alpha_{k+1} \setminus (\delta_4 \cup \delta_5)\rangle \mathit{exec}(F, k)))
\end{aligned}$$

Describes exactly those states that do not admit complete SFA paths that have $\delta_1 \cdot \delta_2$ as a prefix and no occurrence of $\delta_4 \cup \delta_5$ after $\delta_1 \cdot \delta_2$ but before an occurrence of $\delta_3$.

For this proof, we again rename $\delta_1 \cdot \delta_2$ to $\delta_{pre}$, $\delta_3$ to $\delta_{en}$ and $\delta_4 \cup \delta_5$ to $\delta_{dis}$. We do this renaming and the splitting of the formula into parts in one step.

$$\begin{aligned}
\mathit{violate}_{BSFA} &= \langle \delta_{pre}\rangle \mathit{disjunct}_{BSFA}\\
\mathit{disjunct}_{BSFA} &= \bigvee_{F \subseteq Act} \mathit{finprefix}_{BSFA}(F)\\
\mathit{finprefix}_{BSFA}(F) &= \mu Y.(\langle\overline{\delta_{dis}}\rangle Y \vee \langle \delta_{en}\rangle \mathit{tt} \vee \mathit{satisfy}_{BSFA}(F))\\
\mathit{satisfy}_{BSFA}(F) &= \nu X.(\mathit{inf}_{BSFA}(F))
\end{aligned}$$

Where, of course, we define $\mathit{inf}_{BSFA}(F)$ as follows, fixing an arbitrary order on the actions in $F$, with $n = |F|$.

$$\begin{aligned}
\mathit{inf}_{BSFA}(F) &= \mathit{exec}_{BSFA}(F, n)\\
\mathit{exec}_{BSFA}(F, 0) &= [\overline{F}]\mathit{ff} \wedge X\\
\mathit{exec}_{BSFA}(F, k+1) &= \mu W_{k+1}.([\overline{F}]\mathit{ff} \wedge\\
&\qquad (\langle\overline{\delta_{dis}}\rangle W_{k+1} \vee \langle \alpha_{k+1} \setminus \delta_{dis}\rangle \mathit{exec}_{BSFA}(F, k)))
\end{aligned}$$

We start with proving the semantics of $\mathit{exec}_{BSFA}$. For this, we first prove the semantics of the simpler formula $\mu W.([D]\mathit{ff} \wedge (\langle\overline{\delta_{dis}}\rangle W \vee \langle \alpha \setminus \delta_{dis}\rangle \phi$ for a $\phi$ that does not depend on $W$.

**Lemma B.7.** *For arbitrary environment $\epsilon$, state $s \in \mathcal{S}$, set $F \subseteq Act$, action $\alpha \in F$ and $\mu$-calculus formula $\phi$ which does not depend on $W$, we have that $s \in [\![\mu W.([D]\mathit{ff} \wedge (\langle\overline{\delta_{dis}}\rangle W \vee \langle \alpha \setminus \delta_{dis}\rangle\phi))]\!]_\epsilon$ if, and only if, $s$ admits a finite, possibly partial, path $\pi$ meeting the following requirements:*

1. *$\pi$ is $\delta_{dis}$-free, and*
2. *in every state of $\pi$ with the possible exception of the last state, it must be the case that all actions in $D$ are disabled, and*

3. *the last transition of $\pi$ is labelled with $\alpha$, and*

4. *the last state of $\pi$ is in $[\![\phi]\!]_\epsilon$.*

*Proof.* For this proof, we first apply Theorem 4.3 to the formula $\mu W.(\langle\overline{\delta_{dis}}\rangle W \vee \langle\alpha \setminus \delta_{dis}\rangle\phi)$. From this, we know that that formula characterises exactly those paths that admit a finite, possibly partial, that is $\delta_{dis}$-free and of which the final state is in $[\![\langle\alpha \setminus \delta_{dis}\rangle\phi]\!]_\epsilon$. Note that the semantics of $\langle\alpha \setminus \delta_{dis}\rangle\phi$ is exactly those states that admit an $\alpha$ transition to states in $[\![\phi]\!]_\epsilon$, as long as $\alpha \notin \delta_{dis}$. In other words, this simpler formula characterises exactly those states that admit finite, possibly partial, paths that meet requirements 1, 3 and 4. The only difference between $\mu W.([D]\mathit{ff} \wedge (\langle\overline{\delta_{dis}}\rangle W \vee \langle\alpha \setminus \delta_{dis}\rangle\phi))$ and the simpler formula is that there is the action requirement that all states that satisfy $W$ must satisfy $[\![[D]\mathit{ff}]\!]_\epsilon$. This is exactly the condition that there are no enabled transitions labelled with any action in $D$. This is enforced in every state along the path, with the exception of the last state since there only $\phi$ needs to hold, not $W$. In other words, 2 is enforced. □

We now want to use this lemma to prove the semantics of $\mathit{exec}_{BSFA}$ as they are truly defined. The way we defined $\mathit{exec}_{BSFA}(F, 0)$ makes this proof a bit complicated; if the formula said there could be a finite sequence of non-$\delta_{dis}$ transitions before $X$ had to hold again the proof would be a lot simpler, but there would be an extra and unnecessary least fixpoint. Therefore we accept the more complex proof, so that we can have a slightly simpler formula.

**Lemma B.8.** *Let $F$ be an arbitrary set of actions of size $n$, on which an arbitrary order has been fixed. Let $\alpha_1$ be the smallest action in $F$, $\alpha_2$ the second smallest, etc. Let $k$ be an arbitrary natural number between $0$ and $n$. For arbitrary environment $\epsilon$, set $\mathcal{F} \subseteq \mathcal{S}$, and state $s \in \mathcal{S}$, $s \in [\![\mathit{exec}_{BSFA}(F, k)]\!]_{\epsilon[X:=\mathcal{F}]}$ if, and only if, $s$ admits a finite, possibly partial, path $\pi$ meeting the following requirements:*

1. *$\pi$ is $\delta_{dis}$-free, and*

2. *every action $\alpha_m$ with $m \leq k$ occurs at least once in $\pi$, and*

3. *there are transitions $t_{i_k}$, $t_{i_{k-1}}$, $t_{i_{k-2}}$, ..., $t_{i_1}$ in $\pi$ such that $\mathit{action}(t_{i_m}) = \alpha_m$ for all $1 \leq m \leq k$, and $i_m < i_{m-1}$ for all $1 < m \leq k$. More informally: there is a subsequence of the transitions in $\pi$ in which all actions in $F$ smaller than or equal to $\alpha_k$ occur exactly once and in reverse order*[1]*. And*

4. *on $\pi$, the actions in $\overline{F}$ are perpetually disabled, and*

[1]Note that 3 implies 2, so the latter is superfluous. However, we find it useful in further proofs to have this condition stated explicitly.

5. *the final state of $\pi$ satisfies $\mathcal{F}$.*

*Additionally, there is a special requirement when $k = 0$:*

6. *When $k = 0$, $\pi$ consists of a single state.*

*And a special requirement when $k \neq 0$:*

7. *When $k > 0$, the last transition of $\pi$ is labelled with $\alpha_1$.*

*Proof.* For this, we do an inductive proof on $k$.

For the *base*, take $k = 0$. Note that $exec_{BSFA}(F, 0) = [\overline{F}]\mathit{ff} \wedge X$. We need to prove an arbitrary state $s$ is in $[\![ [\overline{F}]\mathit{ff} \wedge X ]\!]_{\epsilon[X:=\mathcal{F}]}$ if, and only if, it admits a path $\pi$ satisfying requirements 1 to 7. We prove both directions separately.

- Assume the state $s$ admits such a path $\pi$. By 6 we know $\pi$ consists only of the state $s$. By 4, all actions in $\overline{F}$ are disabled on all states in $\pi$, hence they are all disabled in $s$. Additionally, by 5, the final state of $\pi$, which is $s$, satisfies $\mathcal{F}$. Hence, $s \in [\![ [\overline{F}]\mathit{ff} ]\!]_{\epsilon[X:=\mathcal{F}]}$ and $s \in [\![ X ]\!]_{\epsilon[X:=\mathcal{F}]}$. Hence, $s \in [\![ exec_{BSFA}(F, 0) ]\!]_{\epsilon[X:=\mathcal{F}]}$.

- Assume $s \in [\![ [\overline{F}]\mathit{ff} \wedge X ]\!]_{\epsilon[X:=\mathcal{F}]}$. Then all actions in $\overline{F}$ are disabled in $s$, and $s \in \mathcal{F}$. Let $\pi = s$, this is a finite, possibly partial path. There are no transitions on this path, hence it is trivially $\delta_{dis}$-free. Since $k = 0$, there are no actions $\alpha_m$ with $m \leq k$, hence conditions 2 and 3 are both satisfied. Since all actions in $\overline{F}$ are disabled in $s$ and $s \in \mathcal{F}$, conditions 4 and 5 are also satisfied. Finally, 6 is satisfied and $k = 0$ so 7 is trivially satisfied. Hence, $s$ admits a path satisfying all requirements.

For our *induction hypothesis* we assume that a state is in $[\![ exec_{BSFA}(F, k) ]\!]_{\epsilon[X:=\mathcal{F}]}$ for $k \geq 0$ if, and only if, it admits a finite, possibly partial, path satisfying requirements 1 to 7.

For our *step* case, we prove the claim for $k + 1$. Note that, since $k \geq 0$, $k + 1$ is definitely greater than 0. Hence, we know that $exec_{BSFA}(F, k + 1) = \mu W_{k+1}.([\overline{F}]\mathit{ff} \wedge (\langle \overline{\delta_{dis}} \rangle W_{k+1} \vee \langle \alpha_{k+1} \setminus \delta_{dis} \rangle exec_{BSFA}(F, k)))$. Using Lemma B.7, we know a state $s'$ satisfies this formula if, and only if, it admits a finite, possibly partial path $\pi'$ meeting the following requirements:

8. $\pi'$ is $\delta_{dis}$-free, and

9. in every state of $\pi'$, with the possible exception of the last state, it must be the case that all actions in $\overline{F}$ are disabled, and

10. the last transition of $\pi'$ is labelled with $\alpha_{k+1}$, and

11. the last state of $\pi'$ satisfies $exec_{BSFA}(F, k)$.

We once again prove both sides of the claim separately, both using the observation above.

- Assume a state $s$ admits a path $\pi$ that is finite, possibly partial, and satisfies all seven conditions with $k+1$ filled in for $k$. We prove $s \in [\![exec_{BSFA}(F, k+1)]\!]_{\epsilon[X:=\mathcal{F}]}$. By our observation earlier, it suffices to prove that $s$ admits a path $\pi'$ meeting the requirements 8 to 11.

  Note that, by 2, the action $\alpha_{k+1}$ definitely occurs in $\pi$ at least once. Note that there may be several transitions labelled with $\alpha_{k+1}$ in $\pi$. For our construction, we want to identify a particular one. By 3, there exists a subsequence of transitions in $\pi$ where all actions $\alpha_m$ with $0 < m \leq k+1$ occur exactly once and in reverse order. Let $t$ be the last possible transition labelled with $\alpha_{k+1}$ that is a candidate for $t_{i_{k+1}}$. Effectively, we want to consider all possible ways to make such a subsequence and ensure we pick the last possible transition that can be the start of such a subsequence. Let $\pi'$ be the prefix of $\pi$ of which $t$ is the last transition. Let $s'$ be the final state of $\pi'$.

  We prove $\pi'$ meets the four requirements. Since $\pi$ is $\delta_{dis}$-free (by 1) and $\overline{F}$ is disabled on all states on $\pi$ (by 4), conditions 8 and 9 are trivially satisfied. We choose $t$, the last transition of $\pi'$, to be a transition labelled with $\alpha_{k+1}$, hence condition 10 is also satisfied. It remains for us to prove that $s'$ satisfies $exec_{BSFA}(F, k)$. We do this through application of the induction hypothesis: $s'$ satisfies $exec_{BSFA}(F, k)$ if it admits a finite, possibly partial path satisfying requirements 1 to 7. We take $\pi''$, the suffix of $\pi$ starting in $s'$, as the witness for $s'$ admitting such a path.

  Since $\pi''$ is the suffix of $\pi$, conditions 1, 4 and 5 are trivially satisfied. For 2 and 3, note that we chose $t$ such that $t$ is the start of a subsequence of $\pi$ on which all actions $\alpha_{k+1}$ to $\alpha_1$ occur exactly once, in that order. Hence, all actions $\alpha_m$ with $m \leq k$ still occur on $\pi$ after $t$, and there is still a subsequence of the actions $\alpha_k$ to $\alpha_1$ in that order. So these two conditions are also satisfied. We do a case distinction on whether $k = 0$:

  - If $k = 0$, then $t$ is labelled with $\alpha_{k+1} = \alpha_1$. The subsequence of all actions $m$ with $m \leq 1$ consists of only $\alpha_1$, so when we selected $t$ to be the last possible transition labelled with $\alpha_{k+1}$ so that such a subsequence could still be formed, $t$ was simply the last transition in $\pi$ labelled with $\alpha_1$. Since $\pi$ satisfies 7 and $k+1 > 0$, the last transition of $\pi$ labelled with $\alpha_1$ is, in fact, the last transition of $\pi$ period. Hence, when taking the suffix of $\pi$ starting in $s'$, there are no transitions left. Hence, $\pi'' = s'$ and condition 6 is satisfied. Of course, 7 is trivially satisfied since $k \not> 0$.

- If $k > 0$, then 6 is trivially satisfied since $k \neq 0$. Additionally, there must still be transitions in $\pi''$ because there must be at least an occurrence of $\alpha_k$, and since $\pi$ satisfies the condition that the last transition is labelled with $\alpha_1$, $\pi''$ satisfies this condition as well. Hence, 7 is satisfied by $\pi''$.

We have shown $\pi''$ satisfies all seven conditions, and so by applying the induction hypothesis we conclude that $s \in [\![exec_{BSFA}(F, k)]\!]_{\epsilon[X:=\mathcal{F}]}$. From this, we can conclude that $\pi'$ satisfies 11, and therefore $\pi'$ witnesses that $s \in [\![exec_{BSFA}(F, k+1)]\!]_{\epsilon[X:=\mathcal{F}]}$.

- Assume a state $s$ is in $[\![exec_{BSFA}(F, k+1)]\!]_{\epsilon[X:=\mathcal{F}]}$. We prove that $s$ admits a path $\pi$ meeting requirements 1 to 7 with $k+1$ in place of $k$. We use the observation due to Lemma B.7, which gives us that $s$ admits a finite, possibly partial path $\pi'$ meeting conditions 8 to 11. Note that, by 11, the final state $s'$ of $\pi'$ is in $[\![exec_{BSFA}(F, k)]\!]_{\epsilon[X:=\mathcal{F}]}$. We apply the induction hypothesis to conclude that $s'$ admits a path $\pi''$ satisfying conditions 1 to 7. We define $\pi$ to be $\pi'\pi''$. We prove $\pi$ meets the requirements 1 to 7 when filling in $k+1$ for $k$.

  Both $\pi'$ and $\pi''$ are $\delta_{dis}$-free by 8 and 1 respectively, hence $\pi$ satisfies 1. Additionally, by 9, all states on $\pi'$ with the possible exception of the last state $s'$ satisfy the condition that all actions in $\overline{F}$ are disabled. Since $s'$ is on $\pi''$, and 4 holds on $\pi''$, $s'$ and all other states on $\pi''$ also satisfy this condition, hence $\pi$ satisfies 4.

  Regarding 2 and 3, observe that since 10 is satisfied by $\pi'$, $\alpha_{k+1}$ occurs in $\pi$ before any of the transitions in $\pi''$. Additionally, because 3 is satisfied by $\pi''$, there exists a subsequence of $\alpha_k$ to $\alpha_1$, in that order, on the transitions of $\pi''$. Since the occurrence of $\alpha_{k+1}$ in $\pi'$ can be prepended to any such subsequence in $\pi''$, it is the case that $\pi$ satisfies 3 and hence also 2.

  For 6, observe that $k+1$ cannot be zero so 6 is trivially satisfied. For 7, we do a case distinction on whether $k = 0$.

  - If $k = 0$, then $\pi''$, which satisfies 6, must consist of only a single state, which is $s'$. Therefore, the last transition of $\pi = \pi'\pi''$ is the last transition of $\pi'$. By 10, this transition is labelled with $\alpha_{k+1} = \alpha_1$. Hence, $\pi$ satisfies 7.
  - If $k > 0$, then since $\pi''$ is a suffix of $\pi$ and $\pi''$ satisfies 7, the last transition of $\pi$ must be labelled with $\alpha_1$ and hence $\pi$ satisfies 7.

  We conclude that $\pi$ satisfies all seven conditions for $k+1$, and hence $s$ admits a path that satisfies these conditions.

By induction, we have proven that for all $0 \leq k \leq n$, it holds that a state $s$ is in $[\![exec_{BSFA}(F, k)]\!]_{\epsilon[X:=\mathcal{F}]}$ if, and only if, it admits a path $\pi$ meeting the requirements stated as part of the lemma. □

Observe that $inf_{BSFA}(F) = exec_{BSFA}(F, n)$ when $|F| = n$. We can therefore observe the following: when $n = 0, F = \emptyset$ in which case $exec_{BSFA}(F, 0)$ will include $[true]ff$. Hence, the fact that when $n = 0$ the path consists of only a single state does not need to be a special requirement, since it follows directly from there being a deadlock state.

**Corollary B.9.** *Let $F$ be an arbitrary set of actions of size $n$, on which an arbitrary order has been fixed. Let $\alpha_1$ be the smallest action in $F$, $\alpha_2$ the second smallest, etc. For arbitrary environment $\epsilon$, set $\mathcal{F} \subseteq \mathcal{S}$, and state $s \in \mathcal{S}$, $s \in [\![inf_{BSFA}(F)]\!]_{\epsilon[X:=\mathcal{F}]}$ if, and only if, $s$ admits a finite, possibly partial, path $\pi$ meeting the following requirements:*

1. *$\pi$ is $\delta_{dis}$-free, and*
2. *every action in $F$ occurs at least once in $\pi$, and*
3. *there is a subsequence of the transitions in $\pi$ in which all actions in $F$ occur exactly once and in reverse order, and*
4. *on $\pi$, the actions in $\overline{F}$ are perpetually disabled, and*
5. *the final state of $\pi$ satisfies $\mathcal{F}$.*

*And a special requirement when $n \neq 0$:*

6. *When $n > 0$, the last transition of $\pi$ is labelled with $\alpha_1$.*

Let us now consider the semantics of $satisfy_{BSFA}(F)$. We define $S_{BSFA,F}$ to be the set of states that admit a $\delta_{dis}$-free path on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled. Note that, unless $F$ is the empty set, the path must be infinite, otherwise the actions in $F$ could not occur infinitely often. We prove that the semantics of $satisfy_{BSFA}(F)$ are exactly this set. We first prove that $S_{BSFA,F}$ is a fixed point of the transformer belonging to the semantics of $satisfy_{BSFA}(F)$.

**Lemma B.10.** *Let $F$ be an arbitrary subset of $Act$ of size $n$, and $\epsilon$ be an arbitrary environment. Let there be an arbitrary ordering on $F$ and let $\alpha_1$ be the least action of $F$, followed by $\alpha_2$, etc. until $\alpha_n$. The set $S_{BSFA,F}$ is a fixed point of the transformer $T_{BSFA,F}$, where*

$$T_{BSFA,F}(\mathcal{F}) = \{s \in \mathcal{S} \mid s \in [\![inf_{BSFA}(F)]\!]_{\epsilon[X:=\mathcal{F}]}\}$$

*Proof.* We prove $S_{BSFA,F}$ is a fixed point of this transformer by proving $S_{BSFA,F} = T_{BSFA,F}(S_{BSFA,F})$ through mutual set inclusion.

- Let $s$ be an arbitrary state in $S_{BSFA,F}$, we prove that $s \in T_{BSFA,F}(S_{BSFA,F})$. From $s \in S_{BSFA,F}$, we conclude that $s$ admits a $\delta_{dis}$-free path $\pi$ on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled. To prove $s \in T_{BSFA,F}(S_{BSFA,F})$, we need to prove that $s \in [\![inf_{BSFA}(F)]\!]_{\epsilon[X:=S_{BSFA,F}]}$. By Corollary B.9, we know this is the case when $s$ admits a finite, possibly partial, path $\pi'$ meeting the following requirements:

  1. $\pi'$ is $\delta_{dis}$-free, and
  2. every action in $F$ occurs at least once in $\pi'$, and
  3. there is a subsequence of the transitions in $\pi'$ in which all actions in $F$ occur exactly once and in reverse order, and
  4. on $\pi'$, the actions in $\overline{F}$ are perpetually disabled, and
  5. the final state of $\pi'$ satisfies $S_{BSFA,F}$, and
  6. if $n > 0$ then the last transition of $\pi'$ is labelled with $\alpha_1$.

  We will find such a path $\pi'$ in a prefix of $\pi$. We find this $\pi'$ as follows: we first take the prefix of $\pi$ up to the first occurrence of $\alpha_n$, we then append to this the next part of $\pi$ up to the first occurrence of $\alpha_{n-1}$ from this point onwards, and we repeat this until we have get to $\alpha_1$. Once we have added the transition labelled with $\alpha_1$, we stop the construction. This is the path $\pi'$ we use. If $F = \emptyset$, this construction is empty and simply produces $\pi' = s$. It is always possible to find an occurrence of a particular action in $F$ after an arbitrary point on $\pi$, because all actions in $F$ occur infinitely often on $\pi$.

  We now show this construction leads to a path $\pi'$ meeting all requirements. Most of these follow quite directly from the construction. Firstly, $\pi$ is a $\delta_{dis}$-free path so $\pi'$ is as well, meaning 1 is satisfied. As part of the construction, we ensure every action in $F$ must occur at least once in $\pi'$, hence 2 is satisfied. This construction specifically ensures that there is a an occurrence of $\alpha_n$, eventually followed by an occurrence of $\alpha_{n-1}$, etc. until $\alpha_1$, hence 3 is satisfied as well. Since on $\pi$ the actions in $\overline{F}$ are perpetually disabled, this is also the case for $\pi'$ and therefore 4 holds. If $n > 0$, then the construction terminates in the state immediately after we add the last $\alpha_1$ transition, hence 6 holds as well.

  Finally, 5 requires a slightly longer argument. We need to show that the last state of $\pi'$, $s'$ is in $S_{BSFA,F}$. To do this, we must show $s'$ admits a $\delta_{dis}$-free path $\pi''$ on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$

are perpetually disabled. If $F = \emptyset$, this is just the path consisting of only the state $s'$. Otherwise, we find this in the suffix of $\pi$ starting in $s'$. After all, all three of the noted properties of $\pi$ ( $\delta_{dis}$-free, $F$ occurs infinitely often, $\overline{F}$ perpetually disabled) hold for any suffix of a path that satisfies them as well.

We have constructed a path $\pi'$ that witnesses $s \in [\![ inf_{BSFA}(F) ]\!]_{\epsilon[X:=S_{BSFA,F}]}$. This allows us to conclude that $s \in T_{BSFA,F}(S_{BSFA,F})$.

- Let $s$ be an arbitrary state in $T_{BSFA,F}(S_{BSFA,F})$, we prove $s \in S_{BSFA,F}$. From the assumption, we know that $s \in [\![ inf_{BSFA}(F) ]\!]_{\epsilon[X:=S_{BSFA,F}]}$. Applying Corollary B.9, this allows us to conclude $s$ admits a finite, possible partial, path $\pi'$ such that:

  1. $\pi'$ is $\delta_{dis}$-free, and
  2. every action in $F$ occurs at least once in $\pi'$, and
  3. there is a subsequence of the transitions in $\pi'$ in which all actions in $F$ occur exactly once and in reverse order, and
  4. on $\pi'$, the actions in $\overline{F}$ are perpetually disabled, and
  5. the final state of $\pi'$ satisfies $S_{BSFA,F}$, and
  6. if $n > 0$ then the last transition of $\pi'$ is labelled with $\alpha_1$.

  We use this to prove $s \in S_{BSFA,F}$. To prove this, we need to show that $s$ admits a $\delta_{dis}$-free path $\pi$ on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled.

  From 5, we know that the final state of $\pi'$, $s'$ is in $S_{BSFA,F}$, hence $s'$ admits a $\delta_{dis}$-free path on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled. Let this path be $\pi''$. We use $\pi = \pi'\pi''$ as a witness for $s \in S_{BSFA,F}$ when $F \neq \emptyset$, otherwise we can just use $\pi = \pi'$. Since both $\pi'$ and $\pi''$ are $\delta_{dis}$-free, so is $\pi$. Since all actions in $F$ occur infinitely often in $\pi''$, they do in $\pi$ as well. Finally, since the actions in $\overline{F}$ are perpetually disabled in both $\pi'$ and $\pi''$, they are in $\pi$ as well.

  We conclude that, since $s$ admits $\pi$, $s$ is in $S_{BSFA,F}$.

We have proven that if an arbitrary state $s$ is in the set $S_{BSFA,F}$ then it is also in $T_{BSFA,F}(S_{BSFA,F})$ and vice versa. Hence, $S_{BSFA,F}$ is a fixed point of $T_{BSFA,F}$. □

To prove $S_{BSFA,F}$ is the exact set characterised by $satisfy_{BSFA}(F)$ we need to show that $S_{BSFA,F}$ is not merely a fixed point of this transformer, but in fact the greatest fixed point.

**Lemma B.11.** *Let $F$ be an arbitrary subset of Act of size $n$, and $\epsilon$ be an arbitrary environment. Let there be an arbitrary ordering on $F$ and let $\alpha_1$ be the least action of $F$, followed by $\alpha_2$, etc. until $\alpha_n$. The set $S_{BSFA,F}$ is the greatest fixed point of the transformer $T_{BSFA,F}$, defined in Lemma B.10.*

*Proof.* We prove this by showing that any state in an arbitrary fixed point of $T_{BSFA,F}$ is also in $S_{BSFA,F}$. Let $\mathcal{F}$ be an arbitrary fixed point of $T_{BSFA,F}$ and let $s$ be an arbitrary state in $\mathcal{F}$. We proceed to prove $s \in S_{BSFA,F}$. To this end, we must demonstrate that $s$ admits a path $\pi$ that is $\delta_{dis}$-free, and on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled.

Since $s \in \mathcal{F}$ and, because $\mathcal{F}$ is a fixed point of $T_{BSFA,F}$, $\mathcal{F} = T_{BSFA,F}(\mathcal{F})$, we know that $s \in T_{BSFA,F}(\mathcal{F})$. By definition of $T_{BSFA,F}$, we conclude that $s \in [\![inf_{BSFA}(F)]\!]_{\epsilon[X:=\mathcal{F}]}$. By applying Corollary B.9, we conclude that $s$ must admit a finite, possibly partial, path $\pi'$ meeting the following requirements:

1. $\pi'$ is $\delta_{dis}$-free, and
2. every action in $F$ occurs at least once in $\pi'$, and
3. there is a subsequence of transitions in $\pi'$ in which all actions in $F$ occur exactly once and in reverse order, and
4. on $\pi'$, the actions in $\overline{F}$ are perpetually disabled, and
5. the final state of $\pi'$ satisfies $\mathcal{F}$, and
6. if $n > 0$, the last transition of $\pi'$ is labelled with $\alpha_1$.

We start our construction of $\pi$ with $\pi'$.

If $F$ is the empty set, then $\pi'$ will consist of only the state $s$ since, by 4, all states on $\pi'$ are deadlock states. Consequently, $\pi'$ cannot contain transitions and is only the state $s$, which is a deadlock state. In this case, $\pi = \pi'$ is a witness of $s \in S_{BSFA,F}$, since this path is $\delta_{dis}$-free by virtue of not having any transitions, and all actions in $F = \emptyset$ occur infinitely often because there are no such actions. Since $s$ is a deadlock state, it is also the case that on all states in $\pi$, all actions in $\overline{F}$ are perpetually disabled.

Henceforth, we assume $F \neq \emptyset$. In this case, $\pi'$ contains at least $n$ transitions since, by 2, all actions in $F$ occur at least once in $\pi'$. Let $s'$ be the final state of $\pi'$. By 5, $s' \in \mathcal{F}$. Since $\mathcal{F}$ is a fixed point of the transformer, we also have $s' \in T_{BSFA,F}(\mathcal{F})$ and hence, by definition of the transformer, $s' \in [\![inf_{BSFA}(F)]\!]_{\epsilon[X:=\mathcal{F}]}$. We can again apply Corollary B.9 to conclude that $s'$ admits a path $\pi''$ that meets conditions 1 to 6 but with $\pi''$ in place of $\pi'$. Append $\pi''$ to the path $\pi$ we have constructed so far.

This construction can be repeated infinitely often: every time we add a partial path to the path $\pi$ we have constructed so far, the final state is in $\mathcal{F}$ and so there is always a next partial path to append. This way, we construct the infinite path $\pi$.

To prove $\pi$ has the required properties, we reference 1 to 6, which hold for every segment of the path we add. That $\pi$ is $\delta_{dis}$-free follows from the fact that every segment we add to $\pi$ is $\delta_{dis}$-free by 1. Additionally, by 4, every state on every segment we add to $\pi$ meets the condition that all actions in $\overline{F}$ are perpetually disabled, so the same holds for $\pi$. Finally, since $\pi$ is constructed from infinitely many segments, and by 2 every action in $F$ occurs at least once in every segment, every action in $F$ occurs infinitely often in $\pi$. We conclude $\pi$ is $\delta_{dis}$-free, all actions in $F$ occur infinitely often on it and all actions in $\overline{F}$ are perpetually disabled on it. Since $s$ admits this path $\pi$, we have established $s \in S_{BSFA,F}$.

Since for every state $s$ in every fixed point $\mathcal{F}$ of $T_{BSFA,F}$, $s \in S_{BSFA,F}$, we conclude $S_{BSFA,F}$ is the greatest fixed point of $T_{BSFA,F}$. $\square$

Since the semantics of $\mathit{satisfy}_{\mathit{BSFA}}(F)$ are exactly the greatest fixed point of $T_{BSFA,F}$ as defined in Lemma B.10, we can conclude that these semantics correspond to $S_{BSFA,F}$.

**Corollary B.12.** . *For all environments $\epsilon$, sets $F \subseteq Act$ and states $s \in \mathcal{S}$, $s \in [\![\mathit{satisfy}_{\mathit{BSFA}}(F)]\!]_\epsilon$ if, and only if, $s$ admits a $\delta_{dis}$-free path $\pi$ on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled.*

Note that since $F \cup \overline{F} = Act$ and $F \cap \overline{F} = \emptyset$, a path on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled is a path on which any action that is enabled infinitely often must be part of $F$ and therefore also occurs infinitely often. Hence, it is an SFA path.

**Corollary B.13.** *For all environments $\epsilon$, sets $F \subseteq Act$ and states $s \in \mathcal{S}$, $s \in [\![\mathit{satisfy}_{\mathit{BSFA}}(F)]\!]_\epsilon$ if, and only if, $s$ admits an **SFA** and $\delta_{dis}$-free path $\pi$ on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled.*

This concludes the main bulk of the proof. We find that proving the semantics of $\mathit{satisfy}_{\mathit{BSFA}}(F)$ is by far the most complicated part. However, we still need to prove the semantics of the other parts of the formula to drawn our final conclusion.

**Lemma B.14.** *For all environments $\epsilon$, sets $F \subseteq Act$ and states $s \in \mathcal{S}$, $s \in [\![\mathit{finprefix}_{\mathit{BSFA}}(F)]\!]_\epsilon$ if, and only if, $s$ admits a finite, possibly partial, $\delta_{dis}$-free path ending in a state $s'$ such that either $s' \in [\![\langle \delta_{en} \rangle tt]\!]_\epsilon$ or $s' \in [\![\mathit{satisfy}_{\mathit{BSFA}}(F)]\!]_\epsilon$.*

*Proof.* This follows directly from Theorem 4.3, using $\alpha = \overline{\delta_{dis}}$ and $\phi = \langle \delta_{en} \rangle tt \vee \mathit{satisfy}_{\mathit{BSFA}}(F)$. $\square$

**Lemma B.15.** *For all environments $\epsilon$ and states $s \in \mathcal{S}$, we know that $s \in [\![disjunct_{BSFA}]\!]_\epsilon$ if, and only if, $s$ admits an SFA path that is either entirely $\delta_{dis}$-free or has a $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled.*

*Proof.* By definition of $disjunct_{BSFA}$, a state $s$ is in $[\![disjunct_{BSFA}]\!]_\epsilon$ if, and only if, there exists some $F \subseteq Act$ such that $s \in [\![finprefix_{BSFA}(F)]\!]_\epsilon$. From Lemma B.14, $s \in [\![finprefix_{BSFA}(F)]\!]_\epsilon$ for some arbitrary $F$ if, and only if, $s$ admits a finite, possibly partial, $\delta_{dis}$-free path ending in a state $s'$ such that either $s' \in [\![\langle\delta_{en}\rangle tt]\!]_\epsilon$ or $s' \in [\![satisfy_{BSFA}(F)]\!]_\epsilon$.

The claim in the lemma is a bi-implication, so we prove both sides separately.

- Assume that for some arbitrary $F$, an arbitrary state $s$ admits a finite, possibly partial, $\delta_{dis}$-free path $\pi'$ ending in a state $s'$ such that $s' \in [\![\langle\delta_{en}\rangle tt]\!]_\epsilon$ or $s' \in [\![satisfy_{BSFA}(F)]\!]_\epsilon$. We prove that $s$ admits an SFA path $\pi$ that is either entirely $\delta_{dis}$-free or has a $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled. We do a case distinction on whether $s' \in [\![\langle\delta_{en}\rangle tt]\!]_\epsilon$.

  - If this is the case, then $\pi'$ is a $\delta_{dis}$-free path ending in a state where $\delta_{en}$ is enabled. This path is guaranteed to be partial, since $s'$ cannot be a deadlock state because $\delta_{en}$ is enabled. Let $\pi$ be an arbitrary SFA path that has $\pi'$ as a prefix. We know such a path must exist, because SFA is a feasible fairness assumption. Hence, $s$ admits an SFA path that has a $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled.
  - If $s' \notin [\![\langle\delta_{en}\rangle tt]\!]_\epsilon$, then we must have $s' \in [\![satisfy_{BSFA}(F)]\!]_\epsilon$. From Corollary B.13, we conclude that $s'$ must admit an SFA path $\pi''$ that is $\delta_{dis}$-free and on which all actions in $F$ occur infinitely often and all actions in $\overline{F}$ are perpetually disabled. We take $\pi = \pi'\pi''$. By Proposition A.16, since $\pi''$ is SFA, so is $\pi$. And since both $\pi'$ and $\pi''$ are $\delta_{dis}$-free, so is $\pi$. Hence, $s$ admits an SFA path that is entirely $\delta_{dis}$-free.

  In both cases, $s$ admits an SFA path that is either entirely $\delta_{dis}$-free or has a $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled. To reiterate, if we assume $s \in [\![disjucnt_{BSFA}]\!]_\epsilon$, then there must be some $F$ such that $s \in [\![finprefix_{BSFA}(F)]\!]_\epsilon$, and then by Lemma B.14 we can eventually prove that $s$ admits such a path.

- Assume an arbitrary state $s$ admits an SFA path $\pi$ that is either entirely $\delta_{dis}$-free or has a $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled. We prove that $s \in [\![disjunct_{BSFA}]\!]_\epsilon$ by proving there exists some $F$ such that $s \in [\![finprefix_{BSFA}(F)]\!]_\epsilon$.

We know $\pi$ is an SFA path, and hence the set $Act$ can be split into the set $F$ of actions that occur infinitely often in $\pi$ and the set $\overline{F}$ of actions that occur finitely often in $F$. Note that an action occurring finitely often in an SFA path does not mean it is perpetually disabled, only that there is some suffix of the path on which that action is perpetually disabled. We prove that for this $F$, $s \in [\![\mathit{finprefix}_{BSFA}(F)]\!]_\epsilon$. We do a case distinction on whether $\delta_{en}$ is enabled at any point along $\pi$.

– If $\delta_{en}$ is enabled on $\pi$, then there is a state $s'$ on $\pi$ that is the first state in which $\delta_{en}$ is enabled. Let $\pi'$ be the prefix of $\pi$ ending in $s'$. Since $\pi$ is either entirely $\delta_{dis}$-free or has a $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled, and $s'$ is the first state on $\pi$ in which $\delta_{en}$ is enabled, we know that all transitions on $\pi$ before $s'$ are guaranteed to not be labelled with actions in $\delta_{dis}$. Hence, $\pi'$ is $\delta_{dis}$-free. Therefore, $\pi'$ is a witness to $s$ admitting a finite, possibly partial, $\delta_{dis}$-free path ending in a state in which $\delta_{en}$ is enabled. By Lemma B.14, $s \in [\![\mathit{finprefix}_{BSFA}(F)]\!]_\epsilon$.

– If $\delta_{en}$ is never enabled on $\pi$, then $\pi$ must be an entirely $\delta_{dis}$-free path. As argued previously, there must be some suffix of $\pi$ in which all actions in $\overline{F}$ are perpetually disabled. After all, if no such suffix existed then these actions would be infinitely often enabled and therefore occur infinitely often, since $\pi$ is SFA. Let $s'$ be the first state on $\pi$ such that the suffix of $\pi$ starting in $s'$, $\pi''$, satisfies the requirement that all actions in $\overline{F}$ are perpetually disabled and $\pi''$ is $\delta_{dis}$-free. By Proposition A.17, a suffix of an SFA path is SFA, so $\pi''$ is SFA. Additionally, since all actions in $F$ occur infinitely often in $\pi$, they also occur infinitely often in $\pi''$. Hence, $\pi''$ witness that $s'$ admits an $\delta_{dis}$-free, SFA path on which all actions in $\overline{F}$ are perpetually disabled and all actions in $F$ occur infinitely often. Using Corollary B.13, we can conclude $s' \in [\![\mathit{satisfy}_{BSFA}(F)]\!]_\epsilon$.

  Let $\pi'$ be the prefix of $\pi$ ending in $s'$. We can use $\pi'$ as a witness for the fact that $s$ admits a finite, $\delta_{dis}$-free path ending in a state that satisfies $\mathit{satisfy}_{BSFA}(F)$. Hence, by Lemma B.14, $s \in [\![\mathit{finprefix}_{BSFA}(F)]\!]_\epsilon$.

In both cases we have shown $s \in [\![\mathit{finprefix}_{BSFA}(F)]\!]_\epsilon$ for this $F$, hence such an $F$ exists and $s \in [\![\mathit{disjunct}_{BSFA}]\!]_\epsilon$.

We have proven both sides of the bi-implication, and can therefore conclude the lemma holds. □

**Lemma B.16.** *For all environments $\epsilon$ and states $s \in \mathcal{S}$, $s \in [\![\mathit{violate}_{BSFA}]\!]_\epsilon$ if, and only if, $s$ admits an SFA path that starts with $\delta_{pre}$, after which it is either entirely $\delta_{dis}$-free or has a finite $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled.*

*Proof.* This follows directly from Lemma B.15, we have simply stuck $\langle \delta_{pre} \rangle$ in front of $\mathit{disjunct}_{BSFA}$. Hence, $\mathit{violate}_{BSFA}$ characterises states that admit paths that have a prefix matching $\delta_{pre}$, after we the path is in a state $s' \in [\![\mathit{disjunct}_{BSFA}]\!]_\epsilon$. □

We restate Theorem 6.2:

**Theorem B.17.** *A state $s \in \mathcal{S}$ satisfies formula Formula 6.2 if, and only if, it does not admit a path $\pi$ meeting the following requirements:*

1. *$\pi$ is complete and satisfies strong fairness of actions, and*
2. *there is a prefix of $\pi$ that matches $\delta_1 \cdot \delta_2$, and*
3. *there is no occurrence of any action in $\delta_4$ or $\delta_5$ in $\pi$ after $\delta_1 \cdot \delta_2$ and before the first occurrence of any action in $\delta_3$.*

*Proof.* This follows from Lemma B.16 and the fact that Formula 6.2 is the negation of $\mathit{violate}_{BSFA}$. Recall that in our proofs, we have renamed $\delta_1 \cdot \delta_2$ to $\delta_{pre}$, $\delta_3$ to $\delta_{en}$ and $\delta_4 \cup \delta_5$ to $\delta_{dis}$. □

## B.3 Proof of Theorem 6.3

This proof is almost a copy of the proof for the WFA formula, which is unsurprising since the formulae are so similar.

We will prove formula Formula 6.3 correct. That means proving that

$$\begin{aligned}&\neg(\langle \delta_1 \cdot \delta_2 \rangle \nu X.( \bigwedge_{\lambda \in Act} (\langle \mathit{true}^\star \cdot \lambda \rangle \mathit{tt} \Rightarrow ( \\ &\qquad \mu Y.(\langle \delta_3 \rangle \mathit{tt} \vee ([\mathit{true}^\star \cdot \lambda]\mathit{ff} \wedge X) \vee \\ &\qquad\qquad \langle \lambda \setminus (\delta_4 \cup \delta_5) \rangle X \vee \langle \overline{\lambda \cup \delta_4 \cup \delta_5} \rangle Y)))))\end{aligned}$$

characterises exactly those states that do not admit complete and FRA paths with prefixes matching $\delta_1 \cdot \delta_2$ and no occurrences of actions in $\delta_4 \cup \delta_5$ after $\delta_1 \cdot \delta_2$ but before the first occurrence of $\delta_3$.

Similar to the WFA proof, we rename $\delta_1 \cdot \delta_2$ to $\delta_{pre}$, $\delta_3$ to $\delta_{en}$ and $\delta_4 \cup \delta_5$ to $\delta_{dis}$. We also split the formula into multiple parts:

$$\begin{aligned}\mathit{violate}_{BFRA} &= \langle \delta_{pre} \rangle \mathit{invariant}_{BFRA} \\ \mathit{invariant}_{BFRA} &= \nu X.( \bigwedge_{\lambda \in Act} (\langle \mathit{true}^\star \cdot \lambda \rangle \mathit{tt} \Rightarrow \mathit{satisfy}_{BFRA}(\lambda))) \\ \mathit{satisfy}_{BFRA}(\lambda) &= \mu Y.(\langle \delta_{en} \rangle \mathit{tt} \vee ([\mathit{true}^\star \cdot \lambda]\mathit{ff} \wedge X) \vee \\ &\qquad \langle \lambda \setminus \delta_{dis} \rangle X \vee \langle \overline{\lambda \cup \delta_{dis}} \rangle Y)\end{aligned}$$

We prove that Formula 6.3 captures exactly those states that do not admit FRA paths that meet the following conditions: the path begins with $\delta_{pre}$, after which it is either entirely $\delta_{dis}$-free or there is a $\delta_{dis}$-free sequence which ends in a state where $\delta_{en}$ is enabled, after which arbitrary actions may occur.

We do this in steps. First, we prove that $\mathit{satisfy}_{BFRA}(\lambda)$ characterises all states that allow the $\lambda$ action to be treated fairly or for $\delta_{en}$ to become enabled, without taking any action in $\delta_{dis}$. More formally:

**Lemma B.18.** *For all environments $\epsilon$, states $s \in \mathcal{S}$, actions $\lambda \in Act$ and sets $\mathcal{F} \subseteq \mathcal{S}$, we have that a state $s \in [\![\mathit{satisfy}_{BFRA}(\lambda)]\!]_{\epsilon[X:=\mathcal{F}]}$ if, and only if, $s$ admits a finite, possibly partial, path $\pi$ meeting the following conditions:*

1. *$\pi$ is $\delta_{dis}$-free, and*
2. *at least one of the following conditions is satisfied:*
   (a) *the path ends in a state in which $\delta_{en}$ is enabled and $\pi$ is $\lambda$-free, or*
   (b) *$\pi$ ends in a state satisfying $\mathcal{F}$ from which $\lambda$ is unreachable and $\pi$ is $\lambda$-free, or*
   (c) *the last transition in $\pi$ is a $\lambda$-transition, this is the only $\lambda$-transition on $\pi$, and the last state satisfies $\mathcal{F}$.*

*Proof.* For this proof, we apply Theorem 4.3, with $\phi = \langle\delta_{en}\rangle tt \vee ([\mathit{true}^\star \cdot \lambda]\mathit{ff} \wedge X) \vee \langle\lambda \setminus \delta_{dis}\rangle X$ and $\alpha = \overline{\lambda \cup \delta_{dis}}$. From the theorem, we conclude

$$s \in [\![\mu Y.(\langle\delta_{en}\rangle tt \vee ([\mathit{true}^\star \cdot \lambda]\mathit{ff} \wedge X) \vee \langle\lambda \setminus \delta_{dis}\rangle X \vee \langle\overline{\lambda \cup \delta_{dis}}\rangle\, Y)]\!]_{\epsilon[X:=\mathcal{F}]}$$

if, and only if, $s$ admits a finite, possibly partial path $\pi'$ meeting the following conditions:

3. on $\pi'$, only actions in $\overline{\lambda \cup \delta_{dis}}$ occur, and
4. $\pi'$ ends in a state $s'$ in $[\![\langle\delta_{en}\rangle tt \vee ([\mathit{true}^\star \cdot \lambda]\mathit{ff} \wedge X) \vee \langle\lambda \setminus \delta_{dis}\rangle X]\!]_{\epsilon[X:=\mathcal{F}]}$, which is equivalent to $s' \in [\![\langle\delta_{en}\rangle tt]\!]_{\epsilon[X:=\mathcal{F}]} \vee s' \in [\![[\mathit{true}^\star \cdot \lambda]\mathit{ff} \wedge X]\!]_{\epsilon[X:=\mathcal{F}]} \vee s' \in [\![\langle\lambda \setminus \delta_{dis}\rangle X]\!]_{\epsilon[X:=\mathcal{F}]}$ .

We now prove that $s$ admits such a path $\pi'$ if, and only if, it admits a path $\pi$ that satisfies 1 and 2.

First, we assume $s$ admits a path $\pi'$ satisfying 3 and 4 and prove it then also admits a path $\pi$ satisfying 1 and 2. From 4, we get that $s'$ is in at least one of three sets. How we construct $\pi$ depends on which of the three sets $s'$ is in.

- If $s'$ is in $[\![\langle\delta_{en}\rangle tt]\!]_{\epsilon[X:=\mathcal{F}]}$ then $\pi = \pi'$. In this case, since $\pi'$ is free from $\delta_{dis}$-actions and $\lambda$, due to 3, so is $\pi$. Additionally, since $\delta_{en}$ is enabled in the final state of $\pi'$, it is also enabled in the final state of $\pi$. Therefore, $\pi$ meets the conditions 1 and 2a.

- If $s'$ is in $[\![[\mathit{true}^\star \cdot \lambda]\mathit{ff} \wedge X]\!]_{\epsilon[X:=\mathcal{F}]}$, then we can conclude $\lambda$ is unreachable from $s'$ and $s' \in \mathcal{F}$. We take $\pi = \pi'$. Since $\pi'$ is free from $\delta_{dis}$-actions and $\lambda$, so is $\pi$. Additionally, since the final state of $\pi'$ satisfies $\mathcal{F}$ and $\lambda$ is unreachable from the final state of $\pi'$, the same holds for $\pi$. Hence, $\pi$ meets the conditions 1 and 2b.

- If $s'$ is in $[\![\langle\lambda \setminus \delta_{dis}\rangle X]\!]_{\epsilon[X:=\mathcal{F}]}$ then $s'$ admits a transition $t$ to a state $s''$ such that $\mathit{action}(t) = \lambda$ and $\mathit{action}(t) \notin \delta_{dis}$ and $s'' \in \mathcal{F}$. We let $\pi = \pi' t s''$. Since $\pi'$ is free from $\delta_{dis}$-actions, and $\mathit{action}(t) \notin \delta_{dis}$, $\pi$ is $\delta_{dis}$-free. Additionally, the last transition of $\pi$ is labelled with $\lambda$ and its last state satisfies $\mathcal{F}$. Finally, since $\pi'$ is $\lambda$-free, $t$ is the only $\lambda$-transition in $\pi$. We conclude that $\pi$ meets the conditions 1 and 2c.

In all three cases, we can construct a path $\pi$ that meets conditions 1 and at least one of the options from 2. We conclude $s$ admits such a path.

The other way around, we assume $s$ admits a path $\pi$ satisfying 1 and 2 and prove it also admits a path $\pi'$ satisfying 3 and 4. We know $\pi$ is $\delta_{dis}$-free from 1, we do a case distinction on which condition from 2 holds:

- If 2a holds on $\pi$, then $\pi$ is also $\lambda$-free and $\delta_{en}$ is enabled in the last state of $\pi$, $s'$. We take $\pi' = \pi$. Since $\pi$ is $\delta_{dis}$-free and $\lambda$-free, so is $\pi'$ and 3 is satisfied. Since $\delta_{en}$ is enabled in $s'$, $s' \in [\![\langle\delta_{en}\rangle tt]\!]_{\epsilon[X:=\mathcal{F}]}$ and hence $\pi'$ satisfies 4.

- If 2b holds on $\pi$, then $\pi$ is also $\lambda$-free, and in the last state $s'$ of $\pi$, $\lambda$ is unreachable and $s' \in \mathcal{F}$. This means $s' \in [\![[\mathit{true}^\star \cdot \lambda]\mathit{ff} \wedge X]\!]_{\epsilon[X:=\mathcal{F}]}$. We take $\pi' = \pi$, and because $\pi$ satisfies 3 and 4, so does $\pi'$.

- If 2c holds on $\pi$, then the last transition of $\pi$, $t$ is a $\lambda$-transition. Because $\pi$ is $\delta_{dis}$-free, we know $\lambda \notin \delta_{dis}$. Additionally, we know from 2c that the last state of $\pi$, $s'$, satisfies $\mathcal{F}$. Let $s'' = \mathit{source}(t)$, then we know $s'' \in [\![\langle\delta \setminus \delta_{dis}\rangle X]\!]_{\epsilon[X:=\mathcal{F}]}$. The prefix of $\pi$ ending in $s''$ is then our candidate for $\pi'$. Since $t$ is the only $\lambda$-transition on $\pi$, $\pi'$ is both $\delta_{dis}$ and $\lambda$-free, meaning it satisfies 3. Additionally, since $s''$ is the last state of $\pi'$, it also satisfies 4.

We conclude that if $s$ admits a path satisfying 1 and 2, it also admits a path satisfying 3 and 4.

We conclude that, since $s \in [\![\mathit{satisfy}_{BFRA}(\lambda)]\!]_{\epsilon[X:=\mathcal{F}]} \Leftrightarrow s$ admits a finite, possibly partial, path satisfying 3 and 4 $\Leftrightarrow s$ admits a finite, possibly partial, path satisfying 1 and 2, the lemma holds. $\square$

We now prove the meaning of $\mathit{invariant}_{BFRA}$. Let $S_{BFRA}$ be the set of states that admits complete, FRA paths that are either entirely $\delta_{dis}$-free, or contain a prefix which is $\delta_{dis}$-free and ends in a state where $\delta_{en}$ is enabled. We prove that $\mathit{invariant}_{BFRA}$ captures exactly the set $S_{BFRA}$.

To do this, we first prove $S_{BFRA}$ is a fixed point for the transformer belonging to the semantics of $\mathit{invariant}_{BFRA}$.

**Lemma B.19.** *The set $S_{BFRA}$ is a fixed point of the transformer $T_{BFRA}$, where*

$$T_{BFRA}(\mathcal{F}) = \bigcap_{\lambda \in Act} \{s \in \mathcal{S} \mid s \in [\![\langle \mathit{true}^\star \cdot \lambda \rangle tt]\!]_{\epsilon[X:=\mathcal{F}]} \Rightarrow s \in [\![\mathit{satisfy}_{BFRA}(\lambda)]\!]_{\epsilon[X:=\mathcal{F}]}\}$$

*For an arbitrary environment $\epsilon$.*

*Proof.* $S_{BFRA}$ being a fixed point of $T_{BFRA}$ means that $T_{BFRA}(S_{BFRA}) = S_{BFRA}$. We prove this through mutual set inclusion.

- Let $s$ be an arbitrary state in $\mathcal{S}$. We assume $s \in S_{BFRA}$ and prove $s \in T_{BFRA}(S_{BFRA})$. To prove $s \in T_{BFRA}(S_{BFRA})$ we need to prove that, for all $\lambda \in Act$, if $\lambda$ is reachable from $s$ then $s \in [\![\mathit{satisfy}_{BFRA}(\lambda)]\!]_{\epsilon[X:=S_{BFRA}]}$. Let $\lambda$ be an arbitrary action in $Act$. If $\lambda$ is not reachable from $s$, the implication trivially holds. Hence, we assume $\lambda$ is reachable from $s$. We need to prove that $s \in [\![\mathit{satisfy}_{BFRA}(\lambda)]\!]_{\epsilon[X:=S_{BFRA}]}$. Using Lemma B.18, we know this is the case if $s$ admits a finite, possibly partial, path $\pi$ such that

  1. $\pi$ is $\delta_{dis}$-free, and
  2. at least one of the following conditions is satisfied:
     (a) the path ends in a state in which $\delta_{en}$ is enabled and $\pi$ is $\lambda$-free, or
     (b) $\pi$ ends in a state satisfying $S_{BFRA}$ from which $\lambda$ is unreachable and $\pi$ is $\lambda$-free, or
     (c) the last transition in $\pi$ is a $\lambda$-transition, this is the only $\lambda$-transition on $\pi$, and the last state satisfies $S_{BFRA}$.

  We therefore only need to prove such a path $\pi$ indeed exists. From $s \in S_{BFRA}$ we know that $s$ admits a complete, FRA path $\pi_{FRA}$ that is either entirely $\delta_{dis}$-free or has a prefix $\pi_{pre}$ that is $\delta_{dis}$-free and ends in a state $s_{pre}$ such that $\delta_{en}$ is enabled in $s_{pre}$. We do a case distinction on whether $\lambda$ is perpetually reachable on $\pi_{FRA}$ or not.

  – If $\lambda$ is perpetually reachable, then since $\pi_{FRA}$ satisfies fair reachability of actions it must be the case that $\lambda$ occurs on $\pi_{FRA}$. Let $t$ be the first transition that is part of $\pi_{FRA}$ which is labelled with $\lambda$. We do a case distinction on whether any state in $\pi_{FRA}$ before the execution of $t$ there is a state $s_{en}$ on which $\delta_{en}$ is enabled.

  * If such a state $s_{en}$ exists, then $\pi_{FRA}$ has a prefix $\pi$ such that $s_{en}$ is the last state of $\pi$ and, because $s_{en}$ comes before the first transition labelled with $\lambda$ by assumption, $\pi$ is $\lambda$-free. Since $\pi_{FRA}$ is $\delta_{dis}$-free, so is $\pi$. We can therefore say that $\pi$ satisfies 1 and 2a.
  * If no such state $s_{en}$ exists, we instead construct a path that satisfies 2c. Recall that $t$ is the first $\lambda$-transition on $\pi_{FRA}$, let $\pi$ be the prefix of $\pi_{FRA}$ where $t$ is the last transition. Since $\pi_{FRA}$ is $\delta_{dis}$-free, so is $\pi$ and 1 is satisfied. We know that $t$ is the last transition of $\pi$, and since it is the first $\lambda$-transition on $\pi_{FRA}$ we also know that $\pi$ is $\lambda$-free with the exception of $t$. To establish $\pi$ satisfies 2c, it only rests to show that $\mathit{target}(t) \in S_{BFRA}$.
    To do this, we need to establish that $\mathit{target}(t)$ admits a complete, FRA path that is either entirely $\delta_{dis}$-free or contains a $\delta_{dis}$-free prefix that ends in a state where $\delta_{en}$ is enabled. The suffix of $\pi_{FRA}$ starting in $\mathit{target}(t)$ is such a path: from Proposition A.9 we know that a suffix of an FRA path is FRA, and since no state $s_{en}$ exists before $t$ on $\pi_{FRA}$, we are either dealing with a fully $\delta_{dis}$-free path, or we are still in the $\delta_{dis}$-free prefix. Either way, the suffix of $\pi_{FRA}$ starting in $\mathit{target}(t)$ is a witness to the fact that $\mathit{target}(t) \in S_{BFRA}$ and hence we can conclude that 2c is satisfied by the prefix of $\pi_{FRA}$ ending in $\mathit{target}(t)$.

– If $\lambda$ is not perpetually reachable on $\pi_{FRA}$ then there are states on $\pi_{FRA}$ from which $\lambda$ is not reachable. Let $s_{not}$ be the first of those states. We do a case distinction on whether $\lambda$ occurs on $\pi_{FRA}$ before $s_{not}$ is reached.
  * If $\lambda$ occurs before $s_{not}$ is reached, then we can repeat the argument for the case where $\lambda$ is perpetually reachable on $\pi_{FRA}$. In this argument, the only thing we use $\lambda$ being perpetually reachable for is to conclude $\lambda$ must occur. Since in this case $\lambda$ occurs anyway, even though it is not perpetually reachable, we can apply the same argument to conclude we can construct a path $\pi$ satisfying 1 and 2.
  * If $\lambda$ does not occur before $s_{not}$ in $\pi_{FRA}$, we know the prefix of $\pi_{FRA}$ before $s_{not}$ is $\lambda$-free. We need to make one more case distinction on whether there is a state $s_{en}$ before $s_{not}$ on which $\delta_{en}$ is enabled.
    · If such a state $s_{en}$ exists, then $\pi_{FRA}$ has a prefix $\pi$ such that $s_{en}$ is the last state of $\pi$ and, because we assumed that there is no occurrence of $\lambda$ before $s_{not}$ and therefore also not before $s_{en}$, $\pi$ is $\lambda$-free. Since $\pi_{FRA}$ is $\delta_{dis}$-free, so is $\pi$. Hence, $\pi$ satisfies 1 and 2a.
    · If no such state $s_{en}$ exists, we construct a path that satisfies

1 and 2b. For this, let $\pi$ be the prefix of $\pi_{FRA}$ ending in $s_{not}$. Since $\pi_{FRA}$ is $\delta_{dis}$-free, so is $\pi$. Hence, $\pi$ satisfies 1. Additionally, we have assumed $\lambda$ is unreachable from $s_{not}$, and we have established the prefix of $\pi_{FRA}$ before $s_{not}$ is $\lambda$-free. To establish $\pi$ satisfies 2b, we only need to show that $s_{not}$ is in $S_{BFRA}$.
This requires us to show that $s_{not}$ admits a complete, FRA path which is either entirely $\delta_{dis}$-free, or contains a $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled. We know $s_{not}$ admits such a path because it admits the suffix of $\pi_{FRA}$ starting in $s_{not}$. Using Proposition A.9, we know that the suffix of an FRA path is FRA. Additionally, by assumption $\delta_{en}$ has not been enabled yet when we get to $s_{not}$, so either $\pi_{FRA}$ is entirely $\delta_{dis}$-free, in which case its suffix is as well, or when we get to $s_{not}$ we are still in the $\delta_{dis}$-free prefix, in which case its suffix still has such a prefix. Either way, the suffix of $\pi_{FRA}$ starting in $s_{not}$ is a witness to the fact that $s_{not} \in S_{BFRA}$ and hence we can conclude that 2c is satisfied by the prefix of $\pi_{FRA}$ ending in $s_{not}$.

We have shown in every case that $s$ admits a path $\pi$ that satisfies 1 and 2. Using Lemma B.18, we conclude that $s \in [\![ \mathit{satisfy}_{BFRA}(\lambda) ]\!]_{\epsilon[X:=S_{BFRA}]}$. Hence, we have shown that for every arbitrary action $\lambda$, if $\lambda$ is reachable from $s$ then $s \in [\![ \mathit{satisfy}_{BFRA}(\lambda) ]\!]_{\epsilon[X:=S_{BFRA}]}$, from which we conclude that $s \in T_{BFRA}(S_{BFRA})$.

- Let $s$ be an arbitrary state in $\mathcal{S}$. We assume $s \in T_{BFRA}(S_{BFRA})$ and prove $s \in S_{BFRA}$. From our assumption, we know that for every action $\lambda \in \mathit{Act}$ that is reachable from $s$, $s \in [\![ \mathit{satisfy}_{BFRA}(\lambda) ]\!]_{\epsilon[X:=S_{BFRA}]}$. We use this to prove that $s \in S_{BFRA}$. To do this, we need to construct a path $\pi$ starting in $s$ such that $\pi$ is a complete, FRA path that is either entirely $\delta_{dis}$-free or has a prefix that is $\delta_{dis}$-free which ends in a state where $\delta_{en}$ is enabled. First, we do a case distinction on whether $s$ is a deadlock state.

  - If $s$ is a deadlock state, then $s$ admits the complete path $\pi = s$. This path is trivially FRA since it is finite, and it is trivially $\delta_{dis}$-free since it contains no transitions at all. Hence, this $\pi$ is a witness for $s$ being in $S_{BFRA}$.
  - If $s$ is not a deadlock state, there must be at least one action enabled in $s$, which means there is also at least one action reachable from $s$. Let $\lambda$ be an arbitrary action that is reachable from $s$. Since $\lambda$ is reachable from $s$, we know that $s \in [\![ \mathit{satisfy}_{BFRA}(\lambda) ]\!]_{\epsilon[X:=S_{BFRA}]}$. Using Lemma B.18,

this is sufficient to conclude that $s$ admits a finite, possibly partial, path $\pi'$ such that

1. $\pi'$ is $\delta_{dis}$-free, and
2. at least one of the following conditions is satisfied:
   (a) the path ends in a state in which $\delta_{en}$ is enabled and $\pi'$ is $\lambda$-free, or
   (b) $\pi'$ ends in a state satisfying $S_{BFRA}$ from which $\lambda$ is unreachable and $\pi'$ is $\lambda$-free, or
   (c) the last transition in $\pi'$ is a $\lambda$-transition, this is the only $\lambda$-transition on $\pi'$, and the last state satisfies $S_{BFRA}$.

We use $\pi'$ to construct $\pi$. For this, we need to do a case distinction on which of the options in 2 holds for $\pi'$.

∗ If the path $\pi'$ satisfies 2a, then $\pi'$ is a finite path ending in a state on which $\delta_{en}$ is enabled. Either $\pi'$ is a complete path, in which case it is trivially FRA because it is finite, or it is a partial path. If it is partial, then because FRA is feasible, it can be extended to a complete FRA path. We therefore know there exists a complete, FRA path $\pi$ of which $\pi'$ is a prefix. Since $\pi'$ is $\delta_{dis}$-free, due to 1, and ends in a state where $\delta_{en}$ is enabled, we know that $\pi$ contains a $\delta_{dis}$-free prefix which ends in a state where $\delta_{en}$ is enabled. So $\pi$ is a witness for $s$ being in $S_{BFRA}$.

∗ If the path $\pi'$ satisfies 2b or 2c, then $\pi'$ ends in a state satisfying $S_{BFRA}$. We do not need any other information from these two cases, so we address them both simultaneously. Let $s'$ be the final state of $\pi'$. Since $s' \in S_{BFRA}$, we know $s'$ admits a path $\pi''$ that is complete, FRA and either entirely $\delta_{dis}$-free or has a $\delta_{dis}$-free prefix which ends in a state where $\delta_{en}$ is enabled. We let $\pi = \pi'\pi''$. From Proposition A.8, we know that since $\pi''$ is FRA and $\pi'$ is finite, $\pi'\pi''$ is FRA. Additionally, since $\pi'$ is $\delta_{dis}$-free, due to 1, we can prepend $\pi'$ to $\pi''$ while preserving that the resulting path is either entirely $\delta_{dis}$-free or contains a $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled. Hence, $\pi = \pi'\pi''$ is a witness for $s$ being in $S_{BFRA}$.

In all three cases, we can construct a $\pi$ that witnesses $s \in S_{BFRA}$.

We conclude that if $s \in T_{BFRA}(S_{BFRA})$ then $s \in S_{BFRA}$.

Through mutual set inclusion, we have proven $T_{BFRA}(S_{BFRA}) = S_{BFRA}$, hence $S_{BFRA}$ is a fixed point of the transformer $T_{BFRA}$. □

To prove that $S_{BFRA} = [\![ invariant_{BFRA} ]\!]_\epsilon$, we still need to show that $S_{BFRA}$ is the *greatest* fixed point of the transformer $T_{BFRA}$.

**Lemma B.20.** *The set $S_{BFRA}$ is the greatest fixed point of $T_{BFRA}$ as defined in Lemma B.19.*

*Proof.* We show that for any $\mathcal{F} \subseteq \mathcal{S}$ satisfying $T_{BFRA}(\mathcal{F}) = \mathcal{F}$, it is the case that $\mathcal{F} \subseteq S_{BFRA}$. Let $\mathcal{F}$ be an arbitrary subset of $\mathcal{S}$ that is a fixed point for $T_{BFRA}$. Let $s$ be an arbitrary state in $\mathcal{F}$. We prove that $s \in S_{BFRA}$ by constructing a complete FRA path $\pi$ starting in $s_0 = s$ that is either entirely $\delta_{dis}$-free or contains a $\delta_{dis}$-free prefix which ends in a state where $\delta_{en}$ is enabled.

Observe that since $s \in \mathcal{F}$ and $\mathcal{F} = T_{BFRA}(\mathcal{F})$, we also have $s \in T_{BFRA}(\mathcal{F})$. From this we know that for all actions $\lambda$ that are reachable from $s$, it is the case that $s \in [\![satisfy_{BFRA}(\lambda)]\!]_{\epsilon[X:=\mathcal{F}]}$. Let $L_0$ be the set of actions that are reachable from $s$. If $L_0$ is empty, then $s$ is a deadlock state. In that case, the path consisting of only the state $s$ serves as a witness for $s$ being in $S_{BFRA}$: it is a complete FRA path which is entirely $\delta_{dis}$-free. We henceforth assume that $L_0$ is not empty.

Consider an arbitrary but fixed order $<$ on $L_0$, and let $\lambda_0$ be the least action in $L_0$ by this order. Since $\lambda_0$ is an action that is reachable from $s$, we know that $s \in [\![satisfy_{BFRA}(\lambda_0)]\!]_{\epsilon[X:=\mathcal{F}]}$. By Lemma B.18, we know that $s$ admits a finite, possibly partial, path $\pi_0$ that meets the following requirements:

1. $\pi_0$ is $\delta_{dis}$-free, and
2. at least one of the following conditions is satisfied:
   (a) the path ends in a state in which $\delta_{en}$ is enabled and $\pi_0$ is $\lambda_0$-free, or
   (b) $\pi_0$ ends in a state satisfying $\mathcal{F}$ from which $\lambda_0$ is unreachable and $\pi_0$ is $\lambda$-free, or
   (c) the last transition in $\pi_0$ is a $\lambda_0$-transition, this is the only $\lambda_0$-transition on $\pi_0$, and the last state satisfies $\mathcal{F}$.

We let $\pi_0$ be the start of our path $\pi$. How we continue constructing $\pi$ depends on which condition holds for 2. We do a case distinction on whether 2a holds.

- If 2a holds, then we stop the algorithm for construction here. Instead of further considering reachable actions, we simply arbitrarily extend $\pi$ to a complete FRA path. We know this is possible, because FRA is feasible. In this case, the $\pi$ we have constructed is a complete FRA path with a $\delta_{dis}$-free prefix which ends in a state where $\delta_{en}$ is enabled: this prefix is $\pi_0$.
- If 2a does not hold, then either 2c or 2b holds. Either way, $\pi_0$ ends in a state $s_1$ with $s_1 \in \mathcal{F}$. We continue construction of $\pi$ from $s_1$. Let $L_1$ be the set of actions reachable from $s_1$. Let $\lambda_1$ be the least action in $L_0 \cap L_1$[2]

[2]We stick as close as possible to the WFA proof here. However, in case of FRA it is unnecessary to specify that we take the intersection of $L_0$ and $L_1$, because any action reachable from $s_1$ is also reachable from $s_0$.

such that $\lambda_0 < \lambda_1$. Since $\mathcal{F} = T_{BFRA}(\mathcal{F})$, $s_1 \in T_{BFRA}(\mathcal{F})$. Hence, since $\lambda_1$ is reachable from $s_1$, we know that $s_1 \in [\![ \mathit{satisfy}_{BFRA}(\lambda_1) ]\!]_{\epsilon[X:=\mathcal{F}]}$. We continue the construction from $s_1$.

Assuming we were in the second scenario, and so did not terminate the construction algorithm, we can now repeat the construction we have done with $s_0$ for $s_1$. We know a path $\pi_1$ exists that meets the condition that it is $\delta_{dis}$-free and either ends in a state where $\delta_{en}$ is enabled, or in a state $s_2$ with $s_2 \in \mathcal{F}$. In both cases, we append $\pi_1$ to the path $\pi$ we are constructing. In the first case, where $\pi$ now ends in a state where $\delta_{en}$ is enabled, we arbitrarily extend $\pi$ to a complete FRA path and abort the algorithm. In the latter case, we continue the construction from $s_2$, now taking the next smallest action $\lambda_2$ of the actions that are reachable from $s_0$, $s_1$ and $s_2$.

We repeat the above construction until there is no more action we can choose, i.e when $L_0 \cap L_1 \cap L_2 \cap \ldots \cap L_k$ no longer contains any action greater than $\lambda_{k-1}$. This will eventually happen, since there are only finitely many actions in our system. Let $s_k$ be the final state of the $\pi$ we have constructed so far. We now continue the construction from $s_k$ by restarting the whole construction procedure, but now using $s_k$ in place of $s_0$. So we take $L_k$, the set of actions reachable from $s_k$, and simply take $\lambda_k$ to be the least action in $L_k$ according to $<$. We call this "refreshing the set of actions" in the remainder of this proof.

This procedure continues until either it is forced to terminate, because we reach a deadlock state or a state where $\delta_{en}$ is enabled, or it continues infinitely. We argue that in all three cases, the path $\pi$ we construct is a complete FRA path that is either entirely $\delta_{dis}$-free or contains a $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled.

- If the construction terminates because we reach a state where $\delta_{en}$ is enabled, then we have finished constructing $\pi$ by extending it to an arbitrary complete FRA path. Since every path segments we added to the construction of $\pi$ during the algorithm was $\delta_{dis}$-free, and $\delta_{en}$ is enabled in the last state before our arbitrary extension, we have a path that has a $\delta_{dis}$-free prefix which ends in a state where $\delta_{en}$ is enabled.

- If the construction terminates because we reach a deadlock state, then we have a complete finite path, which is FRA because all complete finite paths are FRA. Additionally, we have constructed it only from segments we found via application of Lemma B.18, which were all $\delta_{dis}$-free. Hence, we have constructed a complete FRA path that is entirely $\delta_{dis}$-free.

- If the construction never terminates, $\pi$ is infinite. The construction only adds segments that are $\delta_{dis}$-free, because all segments come from applications of

Lemma B.18, so $\pi$ is $\delta_{dis}$-free. Additionally, $\pi$ is infinite so it is complete. The only thing we still need to prove is that $\pi$ is FRA. Consider an arbitrary suffix $\pi'$ of $\pi$, and consider an arbitrary action $\lambda$ that is perpetually reachable on $\pi'$. To prove $\pi$ is FRA, we need to prove that $\lambda$ occurs in $\pi'$.

We know that the construction algorithm until we refresh the set of actions is finite, hence it occurs infinitely often while constructing the infinite path $\pi$. Let $s_r$ be the first state in $\pi'$ for which we refresh the set of actions when constructing $\pi$. Since $\lambda$ is perpetually reachable on $\pi'$, and $sr$ is part of $\pi'$, $\lambda$ is reachable from $s_r$. Hence, $\lambda \in L_r$. From every state in $\pi'$, $\lambda$ is reachable because $\lambda$ is perpetually reachable on $\pi'$. Hence, every time a restriction is added to the set of actions we consider through intersecting with the set of actions reachable in the next state, $\lambda$ remains in the set. We always take the next smallest action in the set, and since $Act$ is finite this means we will eventually choose $\lambda$ as the action we use for constructing the next segment of $\pi$.

So at some point during the constructing of $\pi$ after $s_r$, we add a path segment on which either $\delta_{en}$ is enabled, or $\lambda$ becomes unreachable, or $\lambda$ occurs. In the first case, the construction would terminate and we assumed that it did not. In the second case, $\lambda$ would not be perpetually reachable on $\pi'$, and we assumed it is. Hence, we must be in the last case: the segment we add to $\pi$ must be one in which $\lambda$ occurs. Thus $\lambda$ occurs in $\pi'$.

Since for every suffix of $\pi$, every action that is perpetually reachable on that suffix also occurs in that suffix, $\pi$ satisfies fair reachability of actions. Thus, $\pi$ is a complete FRA path which is entirely $\delta_{dis}$-free.

We have shown that for an arbitrary state $s$ in an arbitrary fixed point $\mathcal{F}$ of $T_{BFRA}$, we can construct a path starting in $s$ that witnesses $s \in S_{BFRA}$. We conclude that $S_{BFRA}$ is the greatest fixed point of $T_{BFRA}$. □

The greatest fixed point of the transformer $T_{BFRA}$ given in Lemma B.19 is the semantics of $invariant_{BFRA}$, hence we can conclude the following.

**Corollary B.21.** *The set of states characterised by $invariant_{BFRA}$ is exactly the set $S_{BFRA}$.*

The difficult part of the proof is now done, since $violate_{BFRA}$ only adds a prefix, and then to get Formula 6.3 we simply negate $violate_{BFRA}$.

**Lemma B.22.** *For all environments $\epsilon$ and states $s \in \mathcal{S}$, $s \in [\![violate_{BFRA}]\!]_\epsilon$ if, and only if, $s$ admits a complete FRA path that starts with $\delta_{pre}$, after which it is either entirely $\delta_{dis}$-free or has a finite $\delta_{dis}$-free prefix ending in a state where $\delta_{en}$ is enabled.*

*Proof.* This follows from Corollary B.21 and the definition of $violate_{BFRA}$. The semantics of $\langle \delta_{pre} \rangle invariant_{BFRA}$ is exactly those states that admits paths that have a prefix $\delta_{pre}$ that ends in a state that satisfies $invariant_{BFRA}$. By Corollary B.21, states satisfying $invariant_{BFRA}$ admit complete FRA paths that are either entirely $\delta_{dis}$-free or have a finite $\delta_{dis}$-free prefix which ends in a state where $\delta_{en}$ is enabled. The addition of $\langle \delta_{pre} \rangle$ to the formula simply ensures there is a prefix exactly matching $\delta_{pre}$ prepended to the path. By Proposition A.8, the resulting path is still FRA. □

We restate Theorem 6.3:

**Theorem B.23.** *A state $s \in \mathcal{S}$ satisfies Formula 6.3 if, and only if, it does not admit a path $\pi$ meeting the following requirements:*

1. *$\pi$ is complete and satisfies fair reachability of actions, and*
2. *there is a prefix of $\pi$ that matches $\delta_1 \cdot \delta_2$, and*
3. *there is no occurrence of any action in $\delta_4$ or $\delta_5$ in $\pi$ after $\delta_1 \cdot \delta_2$ and before the first occurrence of an action in $\delta_3$.*

*Proof.* This follows directly from Lemma B.22 and the fact that Formula 6.3 is the negation of $violate_{BFRA}$. Recall that for this proof, we have renamed $\delta_1 \cdot \delta_2$ to $\delta_{pre}$, $\delta_3$ to $\delta_{en}$ and $\delta_4 \cup \delta_5$ to $\delta_{dis}$. □

# Appendix C

# Note on the WFA Global Response Formula

Recall that in Section 4.3, we presented Formula 4.3, a formula representing global response under WFA where only the $r$ action is actually treated fairly. We restate the formula here.

$$[\mathit{true}^\star \cdot q]\mu Y.((\langle \mathit{true}\rangle \mathit{tt} \wedge [\overline{r}]Y) \vee \nu X.((\langle r\rangle \mathit{tt} \vee Y) \wedge [\overline{r}]X))$$

We noted that it may be confusing that we use $\langle r\rangle \mathit{tt} \vee Y$ instead of just $\langle r\rangle \mathit{tt}$ in this formula. We explain this detail here.

We had initially designed the formula without the extra $Y$ in this part, thinking it was not necessary. After all, a state in which $r$ is perpetually enabled as long as it is not taken would be characterised by $\nu X.(\langle r\rangle \mathit{tt} \wedge [\overline{r}]X)$, without adding the possibility of $Y$ being satisfied instead of $r$ being enabled. However, in Section 7.1 we designed a formula that can be adapted to express global response under the assumption only $r$ is treated weakly fair. When we compared this formula to Formula 4.3 without the extra $Y$ using MLSolver, we found they were not equivalent with the counterexample shown in Figure C.1.

We explain the counterexample in more detail below, but here do attempt an intuitive explanation. Without the extra $Y$, the formula says that in finitely many steps you always reach a state from which either only $r$ is enabled, or from which $r$ is perpetually enabled. It does not account for the possibility that in finitely many steps, you reach a state that admits both finite (partial) paths and infinite paths such that the finite paths end in states where only $r$ is enabled and along the infinite paths $r$ is perpetually enabled. Basically, even when we reach a state from which $r$ *can* be perpetually enabled, we must still allow recursion back into $Y$ because there may be other paths where $r$ is not (yet) perpetually enabled.

The counterexample clarifies this idea. The crucial element in this counterexample is as follows: after taking $t_1$, $t_2$, $t_5$ and $t_9$, we are in state $s_6$. At this point,

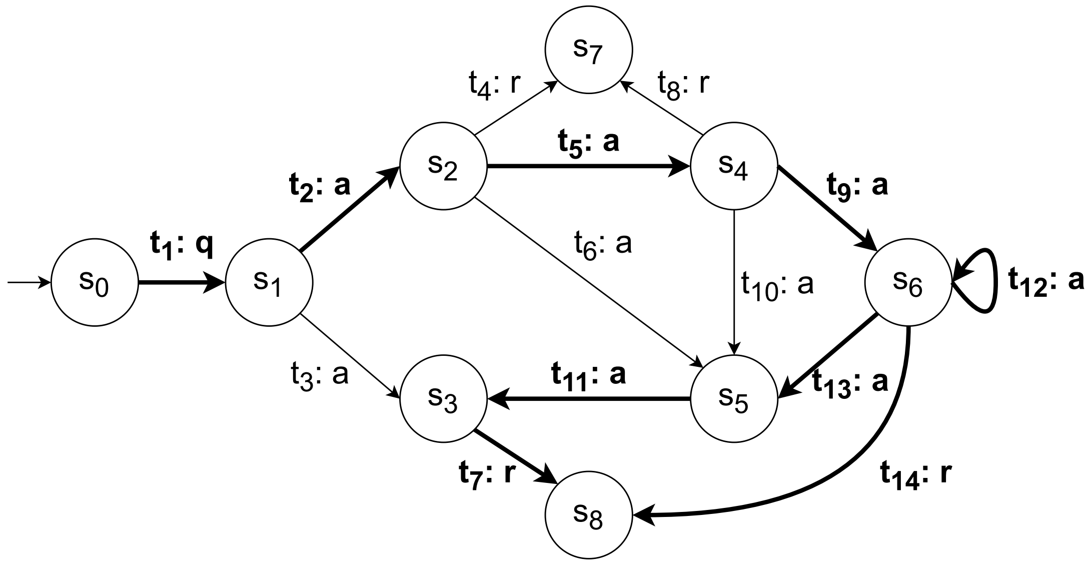


Figure C.1: The counterexample MLSolver found to illustrate why the extra $Y$ in Formula 4.3 is required. We have bolded the most important transitions for clarity.

a $q$ has occurred but no $r$. Since $s_6$ has a self-loop that is not labelled with $r$, $t_{12}$, if we do not assume fairness global response is violated. However, this is an unfair path when assuming the $r$-action is treated weakly fair, because $t_{14}$ is enabled in $s_6$ and is labelled with $r$. However, if we use $\nu X.(\langle r \rangle tt \wedge [\overline{r}]X)$ in Formula 4.3, it reports that global response is violated still. This happens because, while $r$ is enabled in $s_6$, it is not the case that every non-$r$ transition from $s_6$ leads to a state where $r$ is enabled. We could take $t_{13}$, labelled with $a$ to state $s_5$ in which $r$ is not enabled. So $\nu X.(\langle r \rangle tt \wedge [\overline{r}]X)$ is not satisfied in $s_6$. However, this does not lead to a fair and complete violating path either: once we are in $s_5$, we can only take $t_{11}$ to $s_3$ and there $t_7$ is the only enabled transition. Since $t_7$ is an $r$-transition, $r$ is guaranteed to occur on this path. Hence, while $s_6$ does not satisfy the condition that $r$ is enabled and will always remain enabled as long as it does not occur, it does satisfy the condition that $r$ is enabled and will either remain enabled or will eventually be required to occur.

This insight led to us including the extra $Y$ in Formula 4.3, which resulted in it being equivalent to the formula we derive from Section 7.1. We have not proven the equivalence, but MLSolver does report that they are equivalent up to an alphabet of size three.

This example alone, we think, demonstrates the difficulty of designing good modal $\mu$-calculus formulae.

# Appendix D

# Alternate Formulae

Of all the formulae we present in this thesis, Formula 6.1, Formula 6.2 and Formula 6.3 are the three most important ones. Almost all the formulae in this thesis are variations on these three. This is why we include detailed proofs of the semantics of these formulae. While writing these proofs, we noticed some of the details in the formulae seem to be superfluous, or could be presented more elegantly. We did not have time to confirm these suspicions by doing full proofs for these alternate formulae as well, which is why they are not presented in the main text of this thesis. Hence, the observations we make in this appendix are all speculative.

However, we still think it is interesting to include the alternate presentations here. In part because readers of this thesis might have the same ideas upon seeing the formulae and proofs, but primarily because we suspect these alternate formulae are a bit easier to prove correct. Hence, if and when we take the time to mechanically check these proofs with a proof assistant, these formulae may be easier to work with.

## D.1 Weak Fairness and Fair Reachability

The base WFA formula is Formula 6.1, which we restate here.

$$\begin{aligned}&\neg(\langle \delta_1 \cdot \delta_2\rangle \nu X.(\bigwedge_{\lambda\in Act}(\langle\lambda\rangle tt \Rightarrow (\\ &\qquad \mu Y.(\langle\delta_3\rangle tt \vee ([\lambda]ff \wedge X)\vee\\ &\qquad\qquad \langle\lambda \setminus (\delta_4 \cup \delta_5)\rangle X \vee \langle\overline{\lambda \cup \delta_4 \cup \delta_5}\rangle Y)))))\end{aligned}$$

While proving the semantics of this formula for Theorem 6.1 it stood out that the exclusion of $\lambda$ in $\langle\overline{\lambda \cup \delta_4 \cup \delta_5}\rangle Y$ seems to play no role in the semantics of this formula as a whole. It plays a role in the semantics of the least fixpoint, of course, but not in the final formula.

Removing the exclusion of $\lambda$ from this part would simplify the semantics of the least fixpoint a little, and hence make that part of the proof simpler as well. It would only be a matter of the complexity of the proof, the impact on the complexity of computing the formula would be minimal. Note that we have already proven Theorem 6.1, so excluding the $\lambda$ does not make the formula incorrect.

The same argument can be made for the base fair reachability formula Formula 6.3. We think it likely that the exclusion of the $\lambda$-action before Y is superfluous there as well.

## D.2 Strong Fairness

The base formula we present for strong fairness of actions is Formula 6.2, which we restate here.

$$\neg(\langle \delta_1 \cdot \delta_2 \rangle ( \bigvee_{F \subseteq Act} (\mu Y.(\langle \overline{\delta_4 \cup \delta_5} \rangle Y \vee \langle \delta_3 \rangle tt \vee \nu X.\mathit{inf}(F)))))$$

Where $\mathit{inf}(F)$ is defined as follows: we fix an arbitrary order on the actions in $F$ such that $\alpha_1$ is the first action, $\alpha_2$ the second, etc. Let $n = |F|$.

$$\begin{aligned}
\mathit{inf}(F) \quad &= \mathit{exec}(F, n) \\
\mathit{exec}(F, 0) \quad &= [\overline{F}]\mathit{ff} \wedge X \\
\mathit{exec}(F, k+1) \quad &= \mu W_{k+1}.([\overline{F}]\mathit{ff} \wedge \\
&\quad (\langle \overline{\delta_4 \cup \delta_5} \rangle W_{k+1} \vee \langle \alpha_{k+1} \setminus (\delta_4 \cup \delta_5) \rangle \mathit{exec}(F, k)))
\end{aligned}$$

In the $\nu X.\mathit{inf}(F)$ part of this formula we explicitly run through all the actions in $F$, from last to first, requiring them all to occur at least once before we repeat the whole process. In Formula 7.4, we showed that you could also approach weak fairness of actions and fair reachability of actions by running through all the actions one by one and requiring each to satisfy some condition. The formulae for WFA and FRA are written more elegantly, however, by stating the condition needs to eventually be satisfied for all enabled/reachable actions whenever $X$ holds. This way, there do not need to be separate fixpoints for each variable.

This raises the question whether we could not also use this approach for the SFA formula. We can try to write the SFA formula in the same style as the WFA and FRA ones. Unlike the WFA and FRA formulae, the SFA formula requires that all actions in $F$ occur infinitely often regardless of whether they are enabled or reachable, so the left-hand side of the implication that is used in the WFA and FRA formulae can be removed. However, WFA and FRA formulae both use that when there are no enabled/reachable actions, we have reached a deadlock. The

SFA formula would need to explicitly incorporate that when a deadlock is reached no further actions have to occur. We do not move the placement of $\langle\delta_3\rangle tt$, even though we could bring that part more in line with the WFA and FRA formulae, since that would likely just make the proof more complicated rather than less.

The result of these modifications is Formula D.1

$$\begin{aligned}
&\neg(\langle\delta_1 \cdot \delta_2\rangle( \bigvee_{F\subseteq Act} (\mu Y.(\langle\overline{\delta_4 \cup \delta_5}\rangle Y \vee \langle\delta_3\rangle tt \vee \\
&\quad \nu X.( \bigwedge_{\lambda\in F} (\langle true\rangle tt \Rightarrow ( \\
&\qquad \mu W.([\overline{F}]ff \wedge (\langle\lambda \setminus (\delta_4 \cup \delta_5)\rangle X \vee \langle\overline{\delta_4 \cup \delta_5}\rangle W)))))))))
\end{aligned} \tag{D.1}$$

This formula looks better, and it does not have the massive amount of fixpoint operators that Formula 6.2 has. However, since those fixpoints do not contribute to the alternation depth anyway, and the main issue with the SFA formula is the quantification over subsets of *Act*, Formula D.1 is likely barely more efficient than Formula 6.2.

The main difficulty in proving Theorem 6.2 was in proving the exact semantics of the *exec*-function and the $\nu X.inf(F)$-part. So simplifying how these parts of the formula are written would likely simplify the proof as well.

Of course, this is quite the change from Formula 6.2, so until this formula has been proven correct as well, we cannot recommend using it. If we can prove Formula D.1 correct, then it would also be interesting to change the combined formula, Formula 7.4, to use this approach for all types of fairness rather than iterating through all actions.

# Appendix E

# Comparisons with Remenska

This appendix serves as something of an extension to Chapter 6. In this chapter, we presented non-violate style formulae for PSP patterns under weak fairness of actions, strong fairness of actions and fair reachability of actions. We did not give any precondition-style formulae. As stated in that chapter, we generally prefer the non-violate approach since the shape of a fair path is much clearer in those. Not to mention, we actually have formulae in the non-violate style for all three assumptions.

However, in this appendix we would like to compare our fair reachability of actions formulae to the fairness formulae presented in [41], which are in the precondition style. We therefore do translate the precondition-style formula for global response under fair reachability of actions to a base formula. We take the same approach we took for translating the non-violate formulae to base formulae: replacing $\mathit{true}^\star \cdot q$ with $\delta_1 \cdot \delta_2$ and replacing explicit references to $r$ with $\delta_4 \cup \delta_5$. We also add that if $\delta_3$ becomes enabled after $\delta_1 \cdot \delta_2$ while no actions in $\delta_4 \cup \delta_5$ have occurred, there is a violation. The result is Formula E.1.

$$[\delta_1 \cdot \delta_2 \cdot \overline{\delta_4 \cup \delta_5}^\star]([\delta_3]\mathit{ff} \land \langle \mathit{true}^\star \cdot (\delta_4 \cup \delta_5)\rangle \mathit{tt}) \tag{E.1}$$

This formula can also be filled in according to Table 6.2. Due to time constraints, we do not have a correctness proof for this base formula. Once again, our recommendation is to use the non-violate formulae in practical model checking applications, not Formula E.1; this formula is exclusively used for comparison.

In the remainder of this appendix, we discuss some comparisons between Formula E.1 and Remenska's formulae under fairness. Regarding this comparison, it is important to note that Remenska does not explicitly use this exact fairness assumption for her formulae. She describes the underlying fairness assumption as "the assumption that in reality each process of the system is given a fair chance to execute", and she identifies problematic executions in a model as "runs in which a single process gets all execution time, while other processes starve" [41]. The description of the underlying fairness assumption is not as precise as those we present here, and as such we cannot assume that the FRA assumption is exactly what she intended. In fact, based on her wording it seems more likely some form of fairness of components is intended, although we do not see the components themselves referenced in the formulae she presents. However, the global response formula under fairness she presents corresponds to the global response formula under FRA we give, both of which match the fair reachability formula from [35]. As such, we think it is interesting to discuss where and how our FRA formulae differ from the formulae for fairness she presents

On an additional note, we repeat and expand on a footnote in Chapter 5: Remenska's formulae for the property specification patterns are part of PASS, a tool to help engineers generate the correct $\mu$-calculus formula for the property they wish to express. The underlying $\mu$-calculus formulae are not easy to directly extract from the tool, however. On the Github page of PASS[1], a link is included to a page which should contain all the formulae. Sadly, this link no longer leads to a publicly viewable page and the page was not archived. On a different, more recent, part of her Github, we did find an HTML file containing an overview of all patterns[2]. We operate on the assumption that this file is what is meant to be on the public page, and have used it as our source for her formulae where they were not further specified in [41]. However, we must allow for the possibility that these are not the final formulae she designed.

We present the comparisons between our formulae and Remenska's out of interest, and because the comparisons highlight some details of our formulae. However, when we discuss differences between the formulae presented in this thesis and Remenska's formulae, we do not wish to suggest her formulae are incorrect. She may be using a different fairness assumption, and we may be comparing against outdated versions of her formulae.

We now give the comparisons. This appendix very deliberately follows the same structure as Section 6.4.

## E.1 Response Before-variant

This is one of the patterns where our precondition FRA formula differs from Remenska's formula for fairness. Firstly, filling in the FRA precondition base, Formula E.1, with the response before-variant values results in Formula E.2.

$$[\overline{b}^{\star} \cdot q \cdot \overline{r}^{\star}]([b]\mathit{ff} \wedge \langle \mathit{true}^{\star} \cdot r\rangle \mathit{tt}) \tag{E.2}$$

[1]Archived at `https://web.archive.org/web/20230725152438/https://github.com/remenska/PASS`.

[2]Archived at `http://web.archive.org/web/20230901054952/https://raw.githubusercontent.com/remenska/remenska.github.io/master/patterns/index.html`.

This states that as long as no $b$ has occurred, if a $q$ occurs then subsequently until an $r$ occurs it must be the case that $b$ is not enabled and $r$ is reachable.

Note that the following formula is equivalent to Formula E.2, because we impose that the $b$ may not be enabled as long as the $r$ has not been done yet.

$$[\overline{b}^{\star} \cdot q \cdot (\overline{r \cup b})^{\star}]([b]\mathit{ff} \wedge \langle \overline{b}^{\star} \cdot r \rangle \mathit{tt}) \tag{E.3}$$

This alternate version is easier to compare to Remenska's formula.

Remenska's formula for response before-variant under fairness is Formula E.4.

$$[\overline{b}^{\star} \cdot q \cdot (\overline{r \cup b})^{\star}]\langle \overline{b}^{\star} \cdot r \rangle \mathit{tt} \tag{E.4}$$

This is very similar to Formula E.3, but misses the $[b]\mathit{ff}$ condition and as a consequence does not represent response before-variant under the fair reachability of actions assumption.

We show this with the following example.

**Example E.1.** See Figure E.1. After the $q$-step, we are in state $s_1$. In $s_1$, $[b]\mathit{ff}$ does not hold because transition $t_3$ is enabled, so we can immediately see that Formula E.2 and Formula E.3 are not satisfied by this LTS. However, $\langle \overline{b}^{\star} \cdot r \rangle \mathit{tt}$ does hold in $s_1$ because the action $r$ is enabled (transition $t_2$). Additionally, since both the $r$ and $b$ actions are forbidden in Formula E.4, we only care whether $\langle \overline{b}^{\star} \cdot r \rangle \mathit{tt}$ holds in $s_1$. So Formula E.4 is satisfied in this LTS.

Considering that the $s_0 t_1 s_1 t_3$ path is finite and therefore fair, it seems quite clear that response before-variant does not hold in this LTS, at least not under fair reachability of actions, or any feasible fairness assumption for that matter.

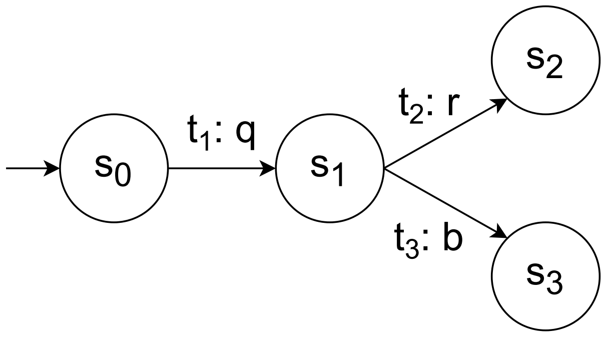


Figure E.1: An LTS in which Formula E.4 is satisfied, but Formula E.2 and Formula E.3 are not.

## E.2 Response After

Similar to Section 6.4, we discuss the variants of the after scope separately.

### Any

Note that the after-any scope is not considered by Remenska, so there is no comparison to be made here.

### First

The precondition formula for fair reachability of actions would be Formula E.5.

$$[\overline{a}^\star \cdot a \cdot \mathit{true}^\star \cdot q \cdot \overline{r}^\star]\langle \mathit{true}^\star \cdot r\rangle \mathit{tt} \tag{E.5}$$

This is also the formula Remenska proposes.

### Last

For response after-last, our formula differs from what is presented by Remenska. If the values from Table 6.2 are filled into Formula E.1, we get Formula E.6.

$$[\mathit{true}^\star \cdot a \cdot \overline{a}^\star \cdot q \cdot \overline{r \cup a}^\star]\langle \mathit{true}^\star \cdot (r \cup a)\rangle \mathit{tt} \tag{E.6}$$

Informally: after every $a$ that is followed by a $q$ without an $a$ in-between, as long as neither an $r$ nor an $a$ has occurred, it must be possible to reach either $r$ or $a$. The idea is that if you can reach an $r$ then by FRA the $r$ will occur and the behaviour is satisfied. If you can reach an $a$ then by FRA the $a$ will occur and this part of the path falls outside of the after-last scope. In either case, the property holds. We could have written $\langle \overline{a}^\star \cdot r\rangle \mathit{tt} \vee \langle \mathit{true}^\star \cdot a\rangle \mathit{tt}$ to highlight that the $r$ must be reachable without doing an $a$, but this does not affect the semantics of the formula. After all, if an $a$ must be taken to reach the $r$, then an $a$ is reachable and thus $\langle \mathit{true}^\star \cdot a\rangle \mathit{tt}$ is true.

The difference with Remenska's formula mainly comes from how the requirement for it being the *last* $a$ is represented, rather than the influence of the fairness assumption. We therefore discuss response after-last without fairness in this comparison. Remenska provides the following formula for response after-last without fairness assumptions:

$$[\mathit{true}^\star \cdot a \cdot \overline{a}^\star](([\mathit{true}^\star \cdot a]\mathit{ff}) \Rightarrow [\mathit{true}^\star \cdot q]\mu Y.(\langle \mathit{true}\rangle \mathit{tt} \wedge [\overline{r}]\, Y)) \tag{E.7}$$

This formula expresses that after an $a$ and subsequently only non-$a$ actions, if you are in a state from which $a$ can never be enabled again, then any subsequent $q$

must be followed by an $r$. At first glance, this indeed captures that the response condition is satisfied after the last $a$ on a path. However, as we demonstrate in Example E.2, a small edge case is missed here.

**Example E.2.** Consider Figure E.2. After taking an $a$-transition, we are in state $s_1$. The $[\overline{a}^\star]$ requires us to consider the subsequent part of the formula for $s_1$ and $s_3$, but not $s_2$, since these are the states than can be reached from $s_1$ with zero or more non-$a$-actions. In $s_1$, $[\mathit{true}^\star \cdot a]\mathit{ff}$ is false, since $a$ is enabled, so the implication is trivially true. In $s_3$, the left-hand side of the implication is true but since no $q$ will ever occur again the right-hand side is trivially true so the whole implication still holds. Thus, the formula holds in this LTS. However, $s_0t_1s_1t_3s_3$ is a complete path which involves an $a$-action, followed by a $q$ and then never followed by either an $a$ or an $r$. So after the last $a$, the response condition is violated. With our understanding of the after-last scope, this is not desirable behaviour. Hence why we propose a different way to represent the after-last scope.

Our approach for representing the after-last scope can also be used for response after-last *without* fairness, as follows:

$$[\mathit{true}^\star \cdot a \cdot \overline{a}^\star \cdot q]\mu Y.(\langle \mathit{true}\rangle \mathit{tt} \wedge [\overline{r \cup a}]\,Y) \tag{E.8}$$

Formula E.8 expresses that after an $a$, if we can reach a $q$ without doing another $a$, then after this $q$ we must always, in finitely many transitions that are neither labelled with $a$ nor $r$, be able to reach a state where only $a$ and/or $r$ are enabled, without encountering any deadlocks. If we reach a state with only $a$ and/or $r$ enabled, then either an $r$ is done and so the response condition is satisfied; or an $a$ is done in which case the previous $a$ was not the last $a$ and so the condition need not be satisfied. We designed Formula E.8 to compare our approach to Remenska's; it was MLSolver that found Example E.2 as a counterexample for the equivalence of these two formulae. Indeed, Formula E.8 correctly reports that the LTS in Example E.2 violates response after-last without fairness assumption.

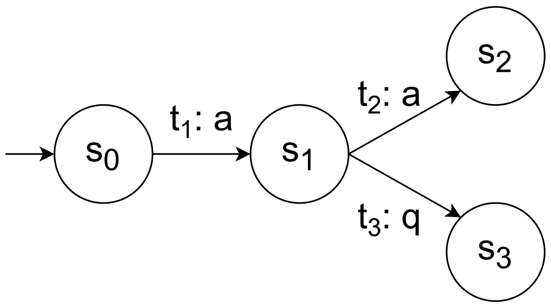


Figure E.2: An LTS demonstrating that Formula E.7 does not capture response after-last.

## E.3 Response Until & Existence

Since the until scopes depend on the before-variant and after scopes, the differences between our formulae and those by Remenska are inherited as well. We do not repeat the discussion of those differences here. There is also nothing further to discuss on existence that has not already been covered for response.

# Appendix F

# Task Formulae

In Section 7.2, we described how to turn the base fairness of actions formulae from Chapter 6 into formulae for fairness of tasks and demonstrated this with the weak fairness of tasks formula. We here show the strong fairness of tasks and fair reachability of tasks formulae as well.

For both formulae, $C$ is the set of actions that is in some task from $\mathcal{F}$.

**Strong fairness of tasks**

$$\neg(\langle \delta_1 \cdot \delta_2 \rangle (\bigvee_{T \subseteq \mathcal{T}} (\mu Y.(\langle \overline{\delta_4 \cup \delta_5} \rangle Y \vee [\mathit{true}]\mathit{ff} \vee \langle \delta_3 \rangle \mathit{tt} \vee \nu X.\mathit{inf}_T(T))))) \tag{F.1}$$

With

$$\begin{aligned}
\mathit{inf}_T(T) &= \mathit{exec}_T(T, n)\\
\mathit{exec}_T(T, 0) &= [D]\mathit{ff} \wedge \langle \overline{\delta_4 \cup \delta_5} \rangle X\\
\mathit{exec}_T(T, k+1) &= \mu W_{k+1}.([D]\mathit{ff} \wedge\\
&\qquad (\langle \overline{\delta_4 \cup \delta_5} \rangle W_{k+1} \vee \langle \tau_{k+1} \setminus (\delta_4 \cup \delta_5) \rangle \mathit{exec}_T(T, k)))
\end{aligned}$$

Where $\tau_1$ is the first task in $T$, $\tau_2$ the second, etc. until $\tau_n$, for some arbitrary order on the tasks in $T$. Additionally, $D = \{\alpha \in \mathit{Act} \mid \exists_{t \in \mathcal{T} \setminus T}.\alpha \in t\}$: the set of actions that must be disabled is any action that is in a task that must be disabled.

As you can see, $C$ is not referenced in this formula at all, this is because the strong fairness formula incorporates the progress assumption differently from the other two formulae: it just says explicitly that you either reach a deadlock in finitely many steps, or the tasks must occur infinitely often (hence it is an infinite path). The only place where $C$ is used in Formula 7.2 is to determine which actions should be treated fairly but are not in the set of actions that occurs infinitely often. With the task formula, an action that is not in any task (and hence not in $C$) will not be forced to be disabled because there is no task $t \in \mathcal{T} \setminus T$ that it appears in, hence it is not in $D$.

**Fair reachability of tasks**

$$\begin{aligned}
&\neg(\langle \delta_1 \cdot \delta_2 \rangle \nu X.( \\
&\quad \bigwedge_{t \in \mathcal{T}} (\langle true^\star \cdot t \rangle tt \Rightarrow ( \\
&\qquad \mu Y.(\langle \delta_3 \rangle tt \vee ([true^\star \cdot t]ff \wedge X) \vee \\
&\qquad\quad \langle t \setminus (\delta_4 \cup \delta_5) \rangle X \vee \langle \overline{t \cup \delta_4 \cup \delta_5} \rangle Y))) \\
&\quad \wedge ([true^\star \cdot C]ff \Rightarrow ([true]ff \vee \langle \delta_3 \rangle tt \vee \langle \overline{\delta_4 \cup \delta_5} \rangle X))))
\end{aligned} \tag{F.2}$$

There is little to say about this formula that has not been said about the WFA formula, since they are so similar.

# Appendix G

# Task Formulae in mCRL2

In Section 8.2 we showed how the base WFA, SFA and FRA formulae can be written in mCRL2 syntax. We briefly discussed how these could be modified to instead work with any arbitrarily chosen set of tasks. We here present the full mCRL2 versions of the task-based formulae. Recall from Section 8.1 that we have a mapping *task* from natural numbers to sets of labels, $T$ is the number of tasks we use and $C$ is the set of all labels that appear in some task.

**Weak fairness** We here show the mCRL2 formula for weak fairness of tasks.

```
!(<$\delta'_1$.$\delta'_2$>nu X.(
  (forall t:Nat. (val(t < T) =>
   ((exists a:Label. (val(a in task(t)) && <l(a)>true)) =>
    (mu Y.(
          <$\delta'_3$>true
       || ((forall a: Label. (val(a in task(t)) =>
          [l(a)]false)) && X)
       || ( exists a: Label. (val(a in task(t)) &&
          <l(a) && !($\delta'_4$ || $\delta'_5$)>X))
       || ( exists a: Label. (val(!(a in task(t))) &&
          <l(a) && !($\delta'_4$ || $\delta'_5$)>Y))
    )))))
  && ((forall a: Label. (val(a in C) => [l(a)]false)) => (
      [true]false || <$\delta'_3$>true || <!($\delta'_4$ || $\delta'_5$)>X
  ))))
```

**Strong fairness** We here show the mCRL2 formula for strong fairness of tasks.

```
forall F: List(Bool). (val(#F == T) => (
  !(<$\delta'_1$.$\delta'_2$>mu Y.(
    <!($\delta'_4$ || $\delta'_5$)>Y
    || <$\delta'_3$>true
    || [true]false
    || nu X.(
      mu W(num: Nat = 0).(
            (val(num == T) => (
              (forall t:Nat.(val(t < T && !(F.t)) =>
                (forall a:Label.(val(a in task(t)) =>
                  [l(a)]false)))) && <!($\delta'_4$||$\delta'_5$)>X))
         && (val(num < T && F.num) => (
              (forall t:Nat.(val(t < T && !(F.t)) =>
                (forall a: Label. (val(a in task(t)) =>
                  [l(a)]false))))
           && ((<!($\delta'_4$||$\delta'_5$)>W(num)
                || (exists a: Label. (val(a in task(num))
                  && <l(a) && !($\delta'_4$||$\delta'_5$)>W(num+1))))))
         && (val(num < T && !(F.num)) => (
              W(num + 1)))
  ))))))
```

**Fair reachability** We here show the mCRL2 formula for fair reachability of tasks.

```
!(<$\delta'_1$.$\delta'_2$>nu X.(
  (forall t: Nat. (val(t < T) =>
  ((exists a: Nat. (val(a < N && order(a) in task(t))
   && <true*.l(order(a))>true)) => (
    mu Y.(
          <$\delta'_3$>true
       || ((forall a: Nat.
          (val(a < N && order(a) in task(t)) =>
           [true*.l(order(a))]false)) && X)
       || ( exists a: Nat.
          (val(a < N && order(a) in task(t)) &&
           <l(order(a)) && !($\delta'_4$ || $\delta'_5$)>X))
       || (exists a: Nat.
          (val(a < N && !(order(a) in task(t))) &&
           <l(order(a)) && !($\delta'_4$ || $\delta'_5$)>Y))
    )))))
  && ((forall a: Nat. (val(a < N && order(a) in C) =>
    [true*.l(order(a))]false)) => (
     [true]false || <$\delta'_3$>true || <!($\delta'_4$ || $\delta'_5$)>X
  ))))
```

# Appendix H

# Dekker's Algorithm Model

## H.1 mCRL2 Model

We here include the mCRL2 model used in Section 8.3. It is based on the model used in [23], which is included with the mCRL2 distribution.

```
sort
    % Labels, this data type means the Label set is infinite
    Label = struct Crit(id: Nat) | Noncrit(id: Nat) |
                   GetFlag(id: Nat, owner: Nat, value: Bool) |
                   SetFlag(id: Nat, owner: Nat, value: Bool) |
                   GetTurn(id: Nat, value: Nat) |
                   SetTurn(id: Nat, value: Nat);
    % Components, also infinite
    Component = struct Comp(id: Nat);

% We added a parameter to every action representing
% which process is responsible for it
act crit, noncrit: Nat;
    get_flag_r, get_flag_s, get_flag,
      set_flag_r, set_flag_s, set_flag: Nat # Nat # Bool;
    get_turn_r, get_turn_s, get_turn,
      set_turn_r,set_turn_s,set_turn: Nat # Nat;
    % We added the action l to the existing model
    l: Label;

map
    order: Nat -> Label;
    N: Nat;
    task: Nat -> Set(Label);
    T: Nat;
    C: Set(Label);
eqn
    order(0) = Crit(0);
    order(1) = Crit(1);
    order(2) = Noncrit(0);
    order(3) = Noncrit(1);
    order(4) = SetFlag(0, 0, true);
    order(5) = SetFlag(1, 1, true);
    order(6) = GetFlag(0, 1, false);
    order(7) = GetFlag(1, 0, false);
    order(8) = SetTurn(0, 1);
    order(9) = SetTurn(1, 0);
    order(10) = GetFlag(1, 0, true);
    order(11) = GetFlag(0, 1, true);
    order(12) = GetTurn(1, 0);
    order(13) = GetTurn(0, 0);
    order(14) = SetFlag(0, 0, false);
    order(15) = SetFlag(1, 1, false);
    order(16) = GetTurn(1, 1);
    order(17) = GetTurn(0, 1);
    N = 18;

    task(0) = {Crit(0), Noncrit(0), SetFlag(0, 0, true), GetFlag(0, 1, false),
               SetTurn(0, 1), GetFlag(0, 1, true), GetTurn(0, 0),
               SetFlag(0, 0, false), GetTurn(0, 1)};
    task(1) = {Crit(1), Noncrit(1), SetFlag(1, 1, true), GetFlag(1, 0, false),
               SetTurn(1, 0), GetFlag(1, 0, true), GetTurn(1, 1),
               SetFlag(1, 1, false), GetTurn(1, 0)};
    T = 2;

    % In this example, C is the whole set of actions that appears in the LTS
    % so we can also leave that part of the formulae out,
    % since [F]ff always implies [true]ff.
    % For illustration purposes, we keep it.
    %C = {l: Label | exists n: Nat. (n < N && l in task(n))};
    % We find that making sets explicit, rather than implicitly defining them
    % as done above, makes verification a bit faster,
    C = {Crit(0), Noncrit(0), SetFlag(0, 0, true), GetFlag(0, 1, false),
            SetTurn(0, 1), GetFlag(0, 1, true), GetTurn(0, 0),
            SetFlag(0, 0, false), GetTurn(0, 1), Crit(1), Noncrit(1),
            SetFlag(1, 1, true), GetFlag(1, 0, false),
            SetTurn(1, 0), GetFlag(1, 0, true), GetTurn(1, 1),
            SetFlag(1, 1, false), GetTurn(1, 0)};

map other: Nat -> Nat;
eqn other(0) = 1;
    other(1) = 0;

proc
  % Shared variables flag (array).
  Flag(i: Nat, b: Bool)=
    sum j: Nat. (j <= 1) -> (
    sum b': Bool. set_flag_r(j, i, b')|l(SetFlag(j, i, b')).Flag(i, b')
    + get_flag_s(j, i, b)|l(GetFlag(j, i, b)).Flag(i, b));

  % Shared variable turn.
  Turn(n:Nat)=
    sum j: Nat. (j <= 1) -> (
    sum n': Nat. set_turn_r(j, n')|l(SetTurn(j, n')).Turn(n')
  + get_turn_s(j, n)|l(GetTurn(j, n)).Turn(n));

  % Main process for Dekker's algorithm.
  Dekker(i: Nat) =
    noncrit(i)|l(Noncrit(i)). set_flag_s(i, i, true) .
    Dekker_outer_loop(i);

  Dekker_outer_loop(i: Nat) =
      % Read shared variable flag
      sum flag_other: Bool . get_flag_r(i, other(i), flag_other) .
      flag_other -> ( sum turn: Nat .
                        get_turn_r(i, turn) .   % Read shared variable turn
                      (turn != i) -> (set_flag_s(i, i, false) . % flag[i] := false
                                    get_turn_r(i, i) .         % while turn != i
                                                               % { /* busy wait */}
                                    set_flag_s(i, i, true) .   % flag[i] := true
                                    Dekker_outer_loop(i)       % loop
                                   )
                                <> Dekker_outer_loop(i))       % if turn == i, loop
                 <> (set_turn_s(i, other(i)) .                 % turn := other(i)
                     crit(i)|l(Crit(i)) .
                     set_flag_s(i, i, false) .                 % flag[i] := false
                    Dekker(i));

init
    hide({ crit, noncrit,
           get_flag, set_flag, get_turn, set_turn},
    allow({ crit|l, noncrit|l,
           get_flag|l, set_flag|l, get_turn|l, set_turn|l},
    comm({ get_flag_r | get_flag_s -> get_flag,
          set_flag_r | set_flag_s -> set_flag,
          get_turn_r | get_turn_s -> get_turn,
          set_turn_r | set_turn_s -> set_turn },
      Dekker(0) || Dekker(1) ||
      Flag(0,false) || Flag(1,false) || Turn(0))));
```

## H.2 $\mu$-Calculus Formulae

We here present the full $\mu$-calculus formulae we used in our verification of Dekker's algorithm.

### Starvation Freedom without Fairness

```
forall i: Nat. (val(i < 2) => (
    !(<true*.l(Noncrit(i))>nu X. (
        [true]false || <!(l(Crit(i)))>X))
))
```

### Starvation Freedom under WFA

```
forall i: Nat. (val(i < 2) => (
    !(<true*.l(Noncrit(i))>nu X.(
      forall a:Label. (<l(a)>true => (
        mu Y.(
              ([l(a)]false && X)
          ||  <l(a) && !(l(Crit(i)))>X
          ||  <!(l(a) || l(Crit(i)))>Y
)))))))
```

### Starvation Freedom under SFA

Note that we could not verify this formula directly.

```
forall i: Nat. (val(i < 2) => (
  forall F: List(Bool). (val(#F == N && !(F.i)) => (
    !(<true*.l(Noncrit(i))>mu Y.(<!l(Crit(i))>Y || nu X.(
        mu W(num: Nat = 0).(
            (val(num == N) => (
                (forall j:Nat.(val(j < N && !(F.j)) =>
                    [l(order(j))]false)) && X))
         && (val(num < N && F.num) => (
                (forall j:Nat.(val(j < N && !(F.j)) =>
                    [l(order(j))]false))
             && (<!l(Crit(i))>W(num)
                    || <l(order(num))
                       && !l(Crit(i))>W(num+1))))
         && (val(num < N && !(F.num)) => (
                W(num+1)))
))))))))
```

### Starvation Freedom under FRA

```
forall i: Nat. (val(i < 2) => (
    !(<true*.l(Noncrit(i))>nu X.(
      forall j: Nat. (val(j < N) &&
      <true*.l(order(j))>true => (
        mu Y.(
              ([true*.l(order(j))]false && X)
          ||  <l(order(j)) && !(l(Crit(i)))>X
          ||  <!(l(order(j)) || l(Crit(i)))>Y
)))))))
```

### Starvation Freedom under WFC

```
forall i: Nat. (val(i < 2) => (
    !(<true*.l(Noncrit(i))>nu X.(
        (forall t: Nat. (val(t < T) =>
        ((exists a: Label. (val(a in task(t))
          && <l(a)>true)) => (
            mu Y.(
                ((forall a: Label.
                    (val(a in task(t)) =>
                      [l(a)]false)) && X)
              ||  (exists a: Label.
                      (val(a in task(t)) &&
                        <l(a) && !(l(Crit(i)))>X))
              ||  (exists a: Label.
                      (val(!(a in task(t))) &&
                        <l(a) && !(l(Crit(i)))>Y))
        ))))) &&
        ((forall a: Label.
            (val(a in C) => [l(a)]false)) => (
                [true]false || <!(l(Crit(i)))>X
        ))
))))
```

## Starvation Freedom under SFC

```
forall i: Nat. (val(i < 2) => (
  forall F: List(Bool). (val(#F == T) => (
    !(<true*.l(Noncrit(i))>mu Y.(
      <!l(Crit(i))>Y || [true]false || nu X.(
        mu W(num: Nat = 0).(
            (val(num == T) => (
                (forall t: Nat.
                  (val(t < T && !(F.t)) =>
                    (forall a: Label.
                      (val(a in task(t)) =>
                        [l(a)]false)))) && <!l(Crit(i))>X))
         && (val(num < T && F.num) => (
                (forall t: Nat.
                  (val(t < T && !(F.t)) =>
                    (forall a: Label.
                      (val(a in task(t)) =>
                        [l(a)]false))))
             && (<!l(Crit(i))>W(num)
                    || (exists a: Label.
                         (val(a in task(num)) &&
                           <l(a) &&
                             !l(Crit(i))>W(num+1))))))
         && (val(num < T && !(F.num)) => (
                W(num+1)))
))))))))
```

### Starvation Freedom under FRC

```
forall i: Nat. (val(i < 2) => (
    !(<true*.l(Noncrit(i))>nu X.(
        (forall t: Nat. (val(t < T) =>
          ((exists a: Nat.
          (val(a < N && order(a) in task(t))
          && <true*.l(order(a))>true)) => (
            mu Y.(
                ((forall a: Nat.
                    (val(a < N && order(a) in task(t)) =>
                      [true*.l(order(a))]false)) && X)
              ||  (exists a: Nat.
                    (val(a < N && order(a) in task(t)) &&
                      <l(order(a)) && !(l(Crit(i)))>X))
              ||  (exists a: Nat.
                    (val(a < N && !(order(a) in task(t))) &&
                      <l(order(a)) && !(l(Crit(i)))>Y))
        ))))) &&
        ((forall a: Nat.
            (val(a < N && order(a) in C) =>
              [true*.l(order(a))]false)) => (
                [true]false || <!(l(Crit(i)))>X
        ))
))))
```

## H.3 Script for SFA Formula

A batch script is provided which we used to run the experiment with the SFA formula. It also has a feature for picking up where the computation left off if interrupted, as long as you know which process and which permutation were last considered. The lts and lps files for the model should already exist.

On a personal note, I have very little experience writing batch files, so the code is functional (on my computer) but likely not particularly well-written.

```
::== Settings ==::
@echo off
::== Setup variables ==::
:: folder all files are in
set "folder=C:\...\case_study_Dekker\"
:: file to store the temporary formulae in
set "formulafile=strong_fairness_part.mcf"
```

```
:: path to lts file from folder
set "ltsfile=Dekker\Dekker_spec.lts"
:: path to lps file from folder
set "lpsfile=Dekker\Dekker_spec.lps"
:: file to store the temporary pbes in
set "pbesfile=Dekker_sfpart.pbes"
:: number of actions = N
set /a "numactions=18"
:: number of permutations that should be done, 2^numactions
set /a "permutations=262144"
:: number of processes
set /a "numprocesses=2"
:: set process number to start at
set /a "startprocess=0"
:: set permutation number to start at
set /a "startperm=0"
:: set whether to skip actions at x + process id
set "skipone=true"
:: set x in the above calculation
set /a "skipx=0"
:: choose whether to record results in a file
set "record=false"
:: set filename that results will be recorded in
set "resfile=starvation_freedom_sfa_results.txt"

::== Actual code ==::
set /a "sub=1"
set /a "perm = %permutations% - %sub%"
set /a "acts = %numactions% - %sub% - %sub%"
set /a "procs = %numprocesses% - %sub%"

SETLOCAL ENABLEDELAYEDEXPANSION
:: iterate through processes, start at startprocess in case
   the computation was interrupted.
for /l %%p in (%startprocess%,1,%procs%) do (
    :: iterate through permutations required, start at
       startperm in case computation was interrupted
    for /l %%n in (!startperm!,1,%perm%) do (
        :: initialise list for F
        set "list=["
        :: we need a bitmask comp to check whether each
           action is true in the list corresponding to this
           permutation
set /a "comp=1"
:: the variable skip tracks whether we should skip
   this list
set "skip=false"
:: iterate over the actions
for /l %%i in (0,1,%acts%) do (
    :: test if, in the bit representation of n, the
       index of this action is 1
    set /a "res=%%n & !comp!"
    IF !res! GEQ 1 (
        :: if yes, add true to the list
        set "list=!list! true,"
        :: if we decide to skip, and we are looking
           at the action that needs to be skipped
           when true, then set skip to true
        IF !skipone! EQU true (
            set /a "calc = %skipx% + %%p"
            IF %%i EQU !calc! (
                set "skip=true"
            )
        )
    ) ELSE (
        :: add false to the list if this action is
           not in this set
        set "list=!list! false,"
    )
    :: bitshift comp to test for the next action
    set /a "comp=!comp! << 1"
)
:: final action is handled separately to properly
   format list
set /a "res=%%n & !comp!"
IF !res! GEQ 1 (
    set "list=!list! true]"
    IF !skipone! EQU true (
        set /a "calc = %skipx% + %%p"
        IF %%i EQU !calc! (
            set "skip=true"
        )
    )
) ELSE (
    set "list=!list! false]"
)
```

```
:: report information
echo At p = %%p and n = %%n
echo ^* Gives list: !list!

IF !skip! EQU true (
    echo ^* Skipped
    :: if chosen to record information, add record
       to file
    IF !record! EQU true (
        echo At p = %%p and n = %%n >> "
           %folder%%resfile%"
        echo ^* List: !list! >> "%folder%%resfile%"
        echo ^* skipped >> "%folder%%resfile%"
    )
) ELSE (
    :: set formula for this process and this F
    set "formula=^!(<true*.l(Noncrit( %%p ))>mu Y.(<
       ^!l(Crit( %%p ))>Y || nu X.( mu W(num: Nat =
       0).((val(num == N) => ((forall j:Nat.(val(j <
        N && ^!( !list!.j)) => [l(order(j))]false))
       && X)) && (val(num < N && !list!.num) => ((
       forall j:Nat.(val(j < N && ^!( !list!.j)) =>
       [l(order(j))]false)) && (<^!l(Crit( %%p ))>W(
       num) || <l(order(num)) && ^!l(Crit( %%p ))>W(
       num+1)))) && (val(num < N && ^!( !list!.num))
        => (W(num+1)))))))"

    :: store formula temporarily
    echo !formula!>"%folder%%formulafile%"

    :: use powershell to call lts2pbes. I only know
       how to do this with powershell not batch
       directly, apologies
    powershell -C "lts2pbes -q -f \"
       %folder%%formulafile%\" \"%folder%%ltsfile%\"
        -l \"%folder%%lpsfile%\" \"
       %folder%%pbesfile%\""

    :: call pbessolve and save result
    FOR /F "tokens=* USEBACKQ" %%F IN (`powershell -
       C "pbessolve -s2 \"%folder%%pbesfile%\""`) DO
        (
                set var=%%F
            )

            :: report output
            echo ^* !var!
            :: if chosen to record information, add record
               to file
            IF !record! EQU true (
                echo At p = %%p and n = %%n >> "
                   %folder%%resfile%"
                echo ^* List: !list! >> "%folder%%resfile%"
                echo ^* Result: !var! >> "%folder%%resfile%"
            )

            :: check if output was false
            IF !var! NEQ true (
                :: if so, we have a violating path, which we
                    can report exists
                :: the violating path itself is not reported
                   , this requires a more expensive
                   calculation to find a counterexample
                echo Violating path found with p = %%p, n =
                   %%n and list = !list!, property is false

                :: quit, we know the property is false
                goto :done
            )
        )
    )

    :: reset startperm, the next process should start at 0
       again
    set /a "startperm = 0"
)

ENDLOCAL

:: if no violations are found, the property is true
echo No violating paths found, property is true
:done
:: keep on screen
PAUSE
```