Technische Universität Berlin
Fakultät IV - Elektrotechnik und Informatik
Modelle und Theorie verteilter Systeme

Bachelor's Thesis

# A Video Game about Reactive Bisimilarity

Eloi-Noël Sandt
ID: 401675

Thesis Advisor
Benjamin Bisping

Thesis Supervisors
Prof. Dr. Uwe Nestmann
Prof. Dr. Stephan Kreutzer

Berlin
December 2022

## Declaration of Independence / Selbstständigkeitserklärung

I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

The independent and unaided completion of the thesis is affirmed by affidavit:

Eloi-Noël Sandt

[illegible]

Signature

Berlin 2.12.2022

Place, Date

## Acknowledgements

I wish to express my sincere thanks to Benjamin Bisping for his invaluable advice and guidance over the course of writing this thesis.

My thanks also go to my mother for her unfailing kindness and encouragement; Haydn Robertson for his verbal support; and Bruno Lönne for agreeing with me that occasionally playing video games together *does* count as valuable thesis research.

## Abstract

*Reactive Bisimilarity* is a notion of equivalence for systems with timeout actions. Such systems can be used to model the event that a process may not be able to carry out certain tasks in certain environments, causing a timeout to occur. To determine whether two systems behave equivalently — with respect to a chosen notion of equivalence — one can play logical games on their model representation as labelled transition systems. Until now, these kinds of logical games did not exist for reactive bisimilarity. Furthermore, reactive bisimilarity is a rather difficult concept. Video games can be used as a means to awaken an intuition for complex concepts in players, in an interactive and educational manner.

This bachelor's thesis presents a characterisation of reactive bisimilarity as a game. The game can be utilised to prove or refute that two systems are reactive bisimilar. Moreover, the theoretical game acts as a basis for the development of a video game. The main goal of the video game is to ease players into the rules of reactive bisimilarity by creating an engaging and instructional experience. Additionally, this work puts forward three algorithms: an algorithm to generate game graphs that depict all courses a game could take, an algorithm to find a shortest move sequence for the attacker to win the game and an algorithm to compute suitable game moves for the defender.

The video game accompanying this thesis was developed in Typescript using the Phaser 3 framework.

## Zusammenfassung

*Reaktive Bisimilarität* ist ein Äquivalenzbegriff für Transitionssysteme mit Time-out-Aktionen. Solche Transitionssysteme können genutzt werden, um zu modellieren, dass ein Prozess in bestimmten Umgebungen bestimmte Aktionen nicht ausführen kann. Dann tritt ein Time-out auf. Soll gezeigt werden, dass zwei Systeme sich unter einem gewählten Äquivalenzbegriff gleich verhalten, können logische Spiele auf ihren Modellrepräsentationen (Transitionssysteme) gespielt werden. Diese Art der logischen Spiele existierte bis jetzt noch nicht für reaktive Bisimilarität. Des Weiteren ist reaktive Bisimilarität ein relativ schwieriges Konzept. Computerspiele können verwendet werden, um in Spielern eine Intuition für komplexe Konzepte auf eine interaktive und lehrreiche Weise zu wecken.

Diese Bachelorarbeit präsentiert eine Charakterisierung von reaktiver Bisimilarität als Spiel. Das Spiel kann die reaktive Bisimilarität zweier Systeme zeigen oder widerlegen. Außerdem dient es als Fundament für die Entwicklung eines Computerspiels. Das Hauptziel des Computerspiels ist es, Spieler in die Regeln von reaktiver Bisimilarität einzuführen und eine einnehmende sowie instruktive Erfahrung zu bieten. Des Weiteren stellt diese Arbeit drei Algorithmen vor: einen Algorithmus, um Spielgraphen zu generieren, die alle möglichen Spielverläufe darstellen; einen Algorithmus, um eine kürzeste Zugfolge für den/die Angreifer/-in zu finden, mit der diese/-r das Spiel gewinnt und einen Algorithmus, um geeignete Spielzüge für den/die Verteidiger/-in zu berechnen.

Das Computerspiel, welches diese Arbeit begleitet, wurde in Typescript mit dem Phaser 3 Framework entwickelt.

# Contents

# Chapter 1

# Introduction

When developing programs for computers, it can be beneficial to specify how processes will behave or react in certain situations. These specifications can for example be modelled with process algebras. [AI05] Let us say that one term of a process algebra describes the actual implementation of a system while another may stand for the specification of the behaviour to be achieved. Then we could compare these two terms to see if they are equal and therefore verify the correctness of the implemented system. This task of comparing two system models is also known as an “equivalence checking problem” [AI05]. Two systems can generally be seen as being equivalent if they exhibit the same behaviour [Ace+07] in the same context. To determine if two systems are equivalent, computer science has developed several so called notions of behavioural equivalence.

One particularly interesting notion of behavioural equivalence is *reactive bisimilarity* which is introduced by Rob van Glabbeek in [Gla22]. It is a rather fine notion and in addition to its popular relative — strong bisimilarity [Ace+07] — it takes into account that some processes may not be able to carry out certain tasks in certain situations and therefore a timeout occurs. A timeout models the end of a time-consuming activity (e.g. the system being idle for a discrete amount of time when no action is possible). The occurrence of a timeout means that all the tasks that could have happened right before the timeout cannot be possible immediately after the timeout, because that is why the timeout occurred in the first place. If we compared two processes with regular *strong bisimilarity* and a timeout occurred, we would have to examine the actions after the timeout even if they are not possible by the definition of timeouts. This could result in two processes not being classified equivalent by strong bisimilarity even though they behave completely the same when observing them. For systems with timeouts, strong bisimilarity therefore is a slightly too strong notion of equivalence. Reactive bisimilarity is an answer to this problem.

## 1.1 Reactive Bisimilarity as a Video Game

One can determine whether two processes are equivalent — with respect to a chosen notion of equivalence — by playing logical games on their model representation as an LTS (labelled transition system) [Ace+07]; an LTS is essentially a graph of nodes corresponding to states a system can be in. These states are connected by directed, labelled edges that symbolise actions a system can perform to ‘transition’ into another state.

As an example for logical games, the *strong bisimulation game* [BNP20] consists of two players, one attacker and one defender. The goal of the attacker is to refute that two processes are strongly bisimilar by finding an action for one process that can not be matched by the second process. The defender on the other hand tries to prove that the processes are strongly bisimilar by answering every action of the attacker for the first process with an appropriate action of the second process. Depending on who wins the game, the two processes are subsequently considered equivalent, or not. Until now, these kinds of logical games did not exist for reactive bisimilarity. This work aims to fill this gap and presents a characterisation of reactive bisimilarity as a game.

Furthermore, reactive bisimilarity is a rather difficult concept. The main goal of this thesis is to partially solve this problem by implementing a video game, easing players into the rules of reactive bisimilarity. It has been shown that video games can be used as a tool to develop an intuition for theoretical concepts as they are able to create an interactive and educational experience for players [MS04]. To guarantee that players will not be overwhelmed with difficulty, the rules of the logical game are gradually introduced in the form of levels building on top of each other, raising the complexity of the video game step by step.

These levels also feature logical games of other relations of equivalence: strong simulation [Ace+07] and the already mentioned strong bisimulation. They are important because the reactive bisimulation game is built upon them. Therefore, they are established to the player in the early levels of the video game. This supports the idea of gradually introducing new rules, thus providing a smooth learning experience.

## 1.2 This Thesis

The main goal of this thesis is to develop a video game based on reactive bisimilarity. This work makes the following contributions:

- *Characterise reactive bisimilarity by a game.* (Section 2.3.2)
- *Prove that the notion of reactive bisimilarity coincides with its game representation.* (Section 2.4)
- *Design a video game* around the reactive bisimulation game. (Section 3.3)
- *Adapt an algorithm to generate game position graphs* from labelled transition systems with timeouts. (Section 4.1)
- *Develop an algorithm which computes the minimum number of moves the attacking player needs to win the game*, given that the defender always chooses moves that maximise this number (worst case). (Section 4.2)
- *Develop an algorithm which computes moves for the defending player.* The video game's human player plays against this algorithm. (Section 4.3)

To afford greatest clarity to the topic at hand, these subjects will be discussed in the following order. Chapter 1 gives an abstract overview of the contents of this thesis. Chapter 2 then introduces essential definitions required to understand further sections of this thesis. Furthermore, it depicts the reactive bisimulation game followed by a proof that the game coincides with the definition of reactive bisimilarity by [Gla22]. Chapter 3 discusses how the theoretical game transfers into a video game and how the video game is designed to be fun and educational at the same time. Finally, chapter 4 will be about the algorithms mentioned in the contributions.

## 1.3 Accompanying artifacts

This work is accompanied by a video game developed in Typescript[1] using the Phaser 3 framework[2].

If you want to try the game out, you can head over to https://eloinoel.github.io/ReactiveBisimilarityGame/. Otherwise, if you want to download the source code and run the game locally, you can go to https://github.com/eloinoel/Reactive BisimilarityGame/tree/main. See the instructions in the README file for further information.

[1]TypeScript official website `https://www.typescriptlang.org/`

[2]Phaser 3 official website `https://phaser.io/phaser3`

# Chapter 2

# Foundation

This chapter establishes the basis to understand further contents of this thesis. Firstly, the chapter begins with definitions of labelled transition systems, followed by several notions of behavioural equivalence. Then, we will proceed by looking at existing characterisations of these notions as logical games as well as a definition for the reactive bisimulation game. Note that the term 'logical games' does not necessarily have deeper meaning from previous literature. It is used in this thesis to differentiate a game from *video* games and refers to theoretical game characterisations. The chapter concludes with a proof that the reactive bisimulation game coincides with the definition of reactive bisimilarity.

## 2.1 Labelled Transition Systems

Labelled transition systems are a way of modelling and visualising the behaviour of processes. The following definition can be found in [BN19, Definition 2.1].

**Definition 2.1** (Labelled Transition System)**.** A *labelled transition system (LTS)* is a triple $(\mathbf{Proc}, \mathbf{Act}, \rightarrow)$ where

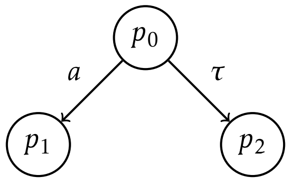
- $\mathbf{Proc}$ is a set of *states* (or *processes*),
- $\mathbf{Act}$ is a set of *actions* containing a special *hidden action* $\tau \in \mathbf{Act}$, and
- $\rightarrow\, \subseteq \mathbf{Proc} \times \mathbf{Act} \times \mathbf{Proc}$ is the *transition relation*

Going forward, LTSs will be used as in [Gla22]. A process or state in an LTS corresponds to a momentary state of a system. Two processes can have action-labelled transitions between them. These symbolise the actions a system can perform in one state to transition into another state. The choice and execution of an action is non-deterministic and assumed to be instantaneous. This means that at any given time, the system must be in one of its states.

A system interacts with its environment by performing visible actions. The environment can be seen as a mechanism that at any moment allows only a subset of the visible actions to be performed. Actions not allowed by the environment in a given state are called *blocked.* When the system performs a visible action, the environment can observe and react to it by changing the set of actions it allows. The effect of an environment is often only treated implicitly in classical literature about LTS. However, this concept will become important when talking about Reactive Bisimilarity.

The *hidden action* $\tau$ models the occurrence of an instantaneous action that is unobservable by the environment and therefore not *visible*. It cannot trigger any state-change in the environment. Moreover, the environment cannot block the occurrence of this action.

**Example 2.2.** In this LTS with $\mathbf{Proc} = \{p_0, p_1, p_2\}$, $\mathbf{Act} = \{a, \tau\}$ and $\rightarrow = \{(p_0, a, p_1), (p_0, \tau, p_2)\}$, the process $p_0$ can perform the action $a$ in environments allowing $a$ to transition into $p_1$ or the hidden action $\tau$ to transition into $p_2$. If the environment were to block all visible actions, $p_0$ could only perform the hidden action $\tau$.



Rob van Glabbeek introduces a specific type of LTS in [Gla22, Section 2] which adds the special *timeout action t* to **Act**:

**Definition 2.3** (Labelled Transition System with timeouts). A *labelled transition system with timeouts (LTS$_t$)* is a triple $(\mathbf{Proc}, \mathbf{Act}, \rightarrow)$ where

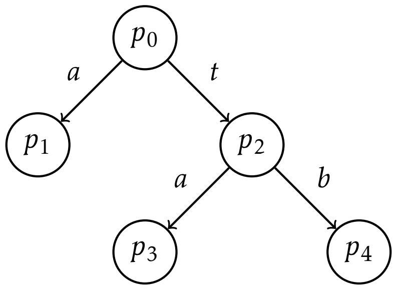
- **Proc** is a set of *states* (or *processes*),
- **Act** is a set of *actions* containing a special *hidden action* $\tau \in \mathbf{Act}$ along with a special *timeout action* $t \in \mathbf{Act}$, and
- $\rightarrow \subseteq \mathbf{Proc} \times \mathbf{Act} \times \mathbf{Proc}$ is the *transition relation*

The *timeout action t* models the end of a time-consuming activity. When a system arrives in a state where no (non-timeout) action is permitted by the current environment, the system idles for a positive amount of time. There are then two possible outcomes. Either the environment spontaneously changes the set of actions it allows so that some visible action $a$ of the current state becomes possible. In that case, an $a$ transition occurs. In the other case, if the current state has a timeout-transition $t$, the system can choose to perform a timeout-transition. The resolution of an idling period is non-deterministic. It is only possible to stay forever in one state, if there are no outgoing timeout transitions $t$. Like the hidden action $\tau$, the timeout action $t$ cannot be observed, caused or blocked by the environment.

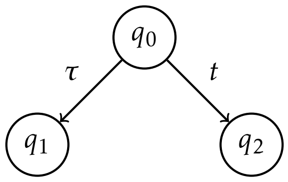
The addition of timeouts enhances the expressive power of LTSs [Gla22]. For instance, it can be used to more accurately model priority mechanisms, the point of which is to prioritise the execution of one process over another. Mutual exclusion, the scenario of granting a resource to only one process at a time, is another example of a mechanism that cannot be correctly modelled in standard process algebras without timeouts [GH15], such as CCS [Mil90].

The following two examples illustrate the general functionality of labelled transition systems with timeouts.

**Example 2.4.** The process $p_0$ can perform a $t$-action only in an environment blocking $a$. As actions occur instantaneously and the timeout action is unobservable, this environment remains the same after performing a $t$-transition to $p_2$. If the environment allowed the action $b$ before the $t$-transition, a $b$-transition occurs immediately after. If $a$ and $b$ were blocked by the environment, the system idles for a possibly infinite amount of time until the environment changes so that at least one of the two visible actions are allowed.



**Example 2.5.** The process $q_0$ can only perform a $\tau$-transition as no environment can block it. Therefore the $t$-transition will never occur in $q_0$.



**Notation 2.6.** We write $A := \textbf{Act} \setminus \{\tau, t\}$ for the set of *visible actions* and $p \xrightarrow{\mathrm{a}} p'$ for $(p, \mathrm{a}, p') \in \rightarrow$. The letters $P, Q, p, q$ are generally used to range over states, $\alpha, \beta$ to range over actions that are not the timeout action $t$, and $a, b, c$ to range over visible actions. Furthermore, let $\mathcal{I}(P) := \{\alpha \in A \cup \{\tau\} \mid P \xrightarrow{\alpha}\}$ be the set of *initial actions* of a process $P \in \textbf{Proc}$. $P \xrightarrow{\alpha}$ means that there is a $Q$ with $P \xrightarrow{\alpha} Q$.

## 2.2 Notions of Equivalence

Two processes are said to be behaviourally equivalent if they exhibit the same (observable) behaviour [Ace+07]. There exist several *notions of behavioural equivalence* that can be used to identify whether two systems exhibit the same behaviour.

This section defines three notions of equivalence: *strong similarity* builds a basis while *strong bisimilarity* acts as a finer notion for LTS. *Strong reactive bisimilarity* seeks to extend strong bisimilarity to LTS$_t$ with timeouts, aiming to more appropriately handle the nature of timeouts. In fact, Rob van Glabbeek states that reactive bisimilarity "coincides with strong bisimilarity when there are no timeout transitions" [Gla22, p. 3].

Note that subsequently, the notions of *strong* similarity, *strong* bisimilarity and *strong* reactive bisimilarity will sometimes be referenced by omitting the 'strong'. There also exist weak versions of similarity and bisimilarity but these are not of importance for this thesis.

### 2.2.1 Strong Similarity

[Gla90, p. 289] defines (strong) simulation as follows:

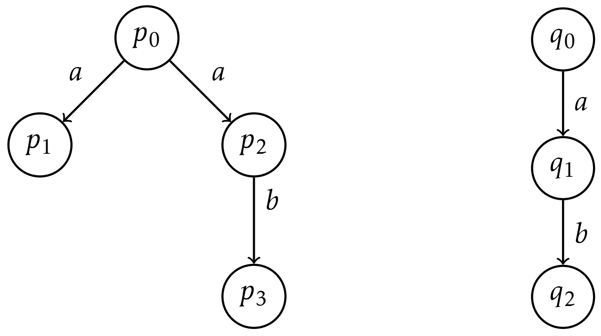
**Definition 2.7** (Simulation)**.** A *simulation* is a binary relation $R$ on processes such that for $\alpha \in \mathbf{Act}$:

- if $(p, q) \in R$ and $p \xrightarrow{\alpha} p'$, then $\exists q' : q \xrightarrow{\alpha} q'$ and $(p', q') \in R$.

A Process $p$ can be *simulated* by $q$ if there is a strong simulation $R$ with $(p, q) \in R$. $p$ and $q$ are (strongly) *similar*, denoted p $\leftrightarrows$, if p is *simulated* by q and q is *simulated* by p.

Simulation means for a pair of states p and q, that for every action $\alpha$ the first state p can perform, the second state q can simulate this behaviour by performing the same action $\alpha$. The resulting state from p performing an $\alpha$-action is then again *simulated* by the resulting state from q performing an $\alpha$-action.

**Example 2.8.** $p_0$ is *simulated* by $q_0$. If $p_0$ performs an $a$-transition to $p_2$, $q_0$ can answer with an $a$-transition to $q_1$. $p_2$ is then again simulated by $q_1$ because both states have a possible $b$-transition. The resulting processes $p_3$ and $q_2$ both have no actions. It remains the $a$-transition from $p_0$ to $p_1$ which can be answered by $q_0$ again choosing the $a$-transition to $q_1$. Analogously, $q_0$ is also *simulated* by $p_0$. Therefore, $p_0$ and $q_0$ are (strongly) *similar.*


$$\begin{tikzcd} & p_0 \arrow[dl, "a"'] \arrow[dr, "a"] & & q_0 \arrow[d, "a"] \\ p_1 & & p_2 \arrow[d, "b"] & q_1 \arrow[d, "b"] \\ & & p_3 & q_2 \end{tikzcd}$$

### 2.2.2 Strong Bisimilarity

[Gla22, Definition 19] defines strong bisimulation as follows:

**Definition 2.9** (Strong Bisimulation)**.** A *strong bisimulation* is a symmetric relation $\mathscr{B}$ on **Proc**, such that, whenever $(p, q) \in \mathscr{B}$,

- if $p \xrightarrow{\alpha} p'$ with $\alpha \in \mathbf{Act}$, then $\exists q' : q \xrightarrow{\alpha} q'$ and $(p', q') \in \mathscr{B}$.

Two processes $p, q \in \mathbf{Proc}$ are *strongly bisimilar*, denoted $p \leftrightarroweq q$, if $(p, q) \in \mathscr{B}$ for some strong bisimulation $\mathscr{B}$.

In contrast to strong simulation, strong bisimulation is a symmetric relation. That means that whenever $(p, q) \in \mathscr{B}$, then also $(q, p) \in \mathscr{B}$.

The two processes $p_0$ and $q_0$ of example 2.8 would not be considered bisimilar. Assume $p_0$ and $q_0$ are part of a bisimulation, $(p_0, q_0) \in \mathscr{B}$. The $a$-transition from $p_0$ to $p_1$ is matched by an $a$-transition from $q_0$ to $q_1$ and $(p_1, q_1) \in \mathscr{B}$. Since $\mathscr{B}$ is a bisimulation and therefore a symmetric relation, also $(q_1, p_1) \in \mathscr{B}$. $q_1$ has a $b$-transition to $q_2$ which $p_1$ cannot answer. Thus, $\mathscr{B}$ is not a bisimulation and $p_0$, $q_0$ are not bisimilar. We can see in this example that bisimilarity is a finer notion of equivalence than similarity.

### 2.2.3 Strong Reactive Bisimilarity

If we used strong bisimilarity on $\text{LTS}_t$ with timeouts, it would treat the timeout action t and the hidden action $\tau$ like any visible action without taking into account their special nature. As stated in the introduction of this thesis, this results in a slightly too strong notion of equivalence for $\text{LTS}_t$. Reactive bisimilarity aims to answer this problem by more appropriately modelling timeout actions and hidden actions. Rob van Glabbeek introduces a definition for reactive bisimilarity in [Gla22, Definition 1] which will be depicted in the following.

Moving forward, the term $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ will be used a great deal. It expresses that a process P has no immediate transitions it can perform in an environment $X$ (it is idle).

**Definition 2.10** (Strong reactive bisimulation). A *strong reactive bisimulation* is a symmetric relation $\mathcal{R} \subseteq (\textbf{Proc} \times \mathcal{P}(A) \times \textbf{Proc}) \cup (\textbf{Proc} \times \textbf{Proc})$ (meaning that $(P, X, Q) \in \mathcal{R} \Leftrightarrow (Q, X, P) \in \mathcal{R}$ and $(P, Q) \in \mathcal{R} \Leftrightarrow (Q, P) \in \mathcal{R}$), such that,

1) if $(P, Q) \in \mathcal{R}$ and $P \xrightarrow{\tau} P'$, then there exists a $Q'$ such that $Q \xrightarrow{\tau} Q'$ and $(P', Q') \in \mathcal{R}$,
2) if $(P, Q) \in \mathcal{R}$ then $(P, X, Q) \in \mathcal{R}$ for all $X \subseteq A$,

and for all $(P, X, Q) \in \mathcal{R}$,

3) if $P \xrightarrow{a} P'$ with $a \in X$, then there exists a $Q'$ such that $Q \xrightarrow{a} Q'$ and $(P', Q') \in \mathcal{R}$,
4) if $P \xrightarrow{\tau} P'$, then there exists a $Q'$ such that $Q \xrightarrow{\tau} Q'$ and $(P', X, Q') \in \mathcal{R}$,
5) if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$, then $(P, Q) \in \mathcal{R}$, and
6) if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{t} P'$, then there exists a $Q'$ such that $Q \xrightarrow{t} Q'$ and $(P', X, Q') \in \mathcal{R}$.

Processes $P, Q \in \textbf{Proc}$ are *strongly X-bisimilar*, denoted $P \leftrightarrow_r^X Q$, if $(P, X, Q) \in \mathcal{R}$ for some strong reactive bisimulation $\mathcal{R}$. They are *strongly reactive bisimilar*, denoted $P \leftrightarrow_r Q$, if $(P, Q) \in \mathcal{R}$ for some strong reactive bisimulation $\mathcal{R}$.

In addition to the definition of strong bisimulation, the definition of *strong reactive bisimulation* features not only pairs of processes $(P, Q)$ but also triples $(P, X, Q)$ consisting of two processes and a set of actions $X$. These triples symbolise that two processes behave the same way (under the notion of strong reactive bisimulation) when they are placed in the environment $X$, which allows only a subset of all the visible actions to occur. A tuple $(P, Q)$, in the context of strong reactive bisimulation, means that two processes $P, Q$ behave the same under indeterminate environments (any environment is possible). This is expressed by clause 2.

Since transitions involving visible actions are affected by the environment, there are no clauses involving visible actions for tuples $(P, Q) \in \mathcal{R}$. However, hidden $\tau$-transitions are not affected by the environment and can therefore occur in any environment. This is reflected in clause 1.

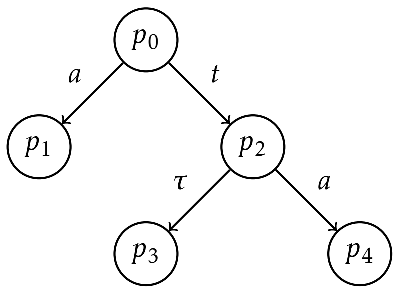
The remaining clauses deal with the more restrictive triples $(P,X,Q)\in\mathcal{R}$. Clause 3 requires that if the state $P$ has a visible $a$-action to state $P'$ that is allowed by the environment $X$, Q has to be able to mirror this behaviour by performing an $a$-transition to state $Q'$. Because $a$ is a visible action, the transition can trigger the environment to change into any other environment. The resulting states $P'$ and $Q'$ should be related by $\mathcal{R}$ under arbitrary environments, thus $(P',Q')\in\mathcal{R}$.

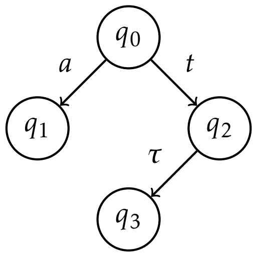
Clause 4 models the mirroring of behaviour for $\tau$-actions pertaining to triples $(P,X,Q)\in\mathcal{R}$. Since $\tau$-actions are unobservable by the environment, these cannot trigger a change in the set of actions allowed by it. Therefore, $(P',X,Q')\in\mathcal{R}$.

Clauses 5 and 6 consider the case of a processes being idle for a positive amount of time due to no immediate transition being possible by the environment. In case of clause 5, it says that during an idling period, the environment may be triggered to change spontaneously. An action that was previously blocked may then become possible. Thus, $P$ and $Q$ should be related in arbitrary environments.

The last clause expresses the other option to end an idling period. If $P$ performs a $t$-transition to $P'$, this behaviour shall be matched by the mirroring process $Q$. As the timeout action is not observable by the environment, the resulting processes $P'$ and $Q'$ should again be related under the environment $X$.

**Example 2.11.** The processes $p_0$ and $q_0$ are strongly reactive bisimilar.





**Proposition 2.12.** *Strong X-bisimilarity and strong reactive bisimilarity are equivalence relations.*

This proposition and its proof can be found in [Gla22, Proposition 2].

Furthermore, Rob van Glabbeek introduces an alternative notion inducing the same concept of strong reactive bisimilarity, called *generalised strong reactive bisimulation (gsrb)* [Gla22, Definition 3].

**Definition 2.13** (gsrb). A *gsrb* is a symmetric relation $\mathcal{R}\subseteq(\mathbf{Proc}\times\mathcal{P}(A)\times\mathbf{Proc})\cup(\mathbf{Proc}\times\mathbf{Proc})$ such that, for all $(P,Q)\in\mathcal{R}$,

1) if $P\xrightarrow{\alpha}P'$ with $\alpha\in A\cup\{\tau\}$, then there exists a $Q'$ such that $Q\xrightarrow{\alpha}Q'$ and $(P',Q')\in\mathcal{R}$,
2) if $\mathcal{I}(P)\cap(X\cup\{\tau\})=\emptyset$ with $X\subseteq A$ and $P\xrightarrow{t}P'$, then there exists a $Q'$ such that $Q\xrightarrow{t}Q'$ and $(P',X,Q')\in\mathcal{R}$,

and for all $(P, Y, Q) \in \mathcal{R}$,

3) if $P \xrightarrow{a} P'$ with either $a \in Y$ or $\mathcal{I}(P) \cap (Y \cup \{\tau\}) = \emptyset$, then there exists a $Q'$ with $Q \xrightarrow{a} Q'$ and $(P', Q') \in \mathcal{R}$,
4) if $P \xrightarrow{\tau} P'$, then there exists a $Q'$ such that $Q \xrightarrow{\tau} Q'$ and $(P', Y, Q') \in \mathcal{R}$,
5) if $\mathcal{I}(P) \cap (X \cup Y \cup \{\tau\}) = \emptyset$ with $X \subseteq A$ and $P \xrightarrow{t} P'$, then there exists a $Q'$ with $Q \xrightarrow{t} Q'$ and $(P', X, Q') \in \mathcal{R}$.

Unlike definition 2.10 for strong reactive bisimulation, a gsrb only needs the triples $(P, X, Q)$ after a $t$-transition occurs (clause 2). This means that two systems without $t$-transitions can be related without using these triples in a gsrb at all. It is supposed to make the notion of strong reactive bisimilarity convenient to use for further analysis. It might also seem more familiar because the first clause is like the common transfer property of strong bisimulation (see definition 2.9). The notion of gsrb builds the core for the reactive bisimulation game which will be introduced and explained in section 2.3.

**Proposition 2.14.** *$P \leftrightarrows_r Q$ iff there exists a gsrb $\mathcal{R}$ with $(P, Q) \in \mathcal{R}$.*
*Likewise, $P \leftrightarrows_r^X Q$ iff there exists a gsrb $\mathcal{R}$ with $(P, X, Q) \in \mathcal{R}$.*

This proposition and proof for it can be found in [Gla22, Proposition 4]. The proof has further been formalised in [Poh21, p. 28] using the interactive proof assistant Isabelle.

## 2.3 Game Theory

In this section, different game characterisations of notions of behavioural equivalence (from section 2.2) are introduced. Playing these games is a way to determine whether two processes are equivalent under a respective notion of equivalence. The theory behind these games has its foundation in Gale-Stewart-style games [GS53].

For this thesis, we look at Gale-Stewart-style *reachability* games where the defender wins all infinite plays. The fundamental idea is the following: Two players, attacker and defender, compete against each other on an LTS representing two systems. If the defender wins, the two systems are considered equivalent. It is a perfect information game with exactly one winner and one loser. The result can be computed from the initial game position [BNP20, p. 451], assuming that both players play 'optimally'.

The following definitions are taken from [BJN21, Definition 2.3], with some slight modifications.

**Definition 2.15** (Game)**.** A *reachability game* $\mathcal{G}[p_0] = (G, G_{\mathrm{d}}, \rightarrowtail, p_0)$ is played on a directed graph consisting of

- a set of *game positions* $G$,
  - partitioned into a set of *defender positions* $G_{\mathrm{d}} \subseteq G$
  - and *attacker positions* $G_{\mathrm{a}} := G \backslash G_{\mathrm{d}}$,
- a transition relation of *game moves* $\rightarrowtail \subseteq G \times G$, and
- an *initial position* $p_0 \in G$.

**Definition 2.16** (Plays and wins)**.** The path $p_0 p_1 \dots \in G^\infty$ with $p_i \rightarrowtail p_{i+1}$ is called a *play* of $\mathcal{G}[p_0]$. The defender *wins* infinite plays. If a finite play $p_0 \dots p_n \not\rightarrowtail$ is stuck, then the stuck player loses. Therefore, the defender wins if $p_n \in G_{\mathrm{a}}$, and the attacker wins if $p_n \in G_{\mathrm{d}}$.

**Definition 2.17** (Strategies and winning strategies)**.** A *strategy* is a (usually partial) mapping from initial play fragments to next moves. If following a strategy $f$ ensures a player to win, $f$ is called a *winning strategy* for this player. The player with a winning strategy for $\mathcal{G}[p_0]$ is said to win $\mathcal{G}[p_0]$.

**Definition 2.18** (Winning regions)**.** The set $W_{\mathrm{a}} \subseteq G$ of all positions $g$ where the attacker has a winning strategy for $\mathcal{G}[g]$ is called the *attacker winning region.* (The defender winning region $W_{\mathrm{d}}$ is defined analogously.)

**Proposition 2.19.** *Reachability games are determined.*

It is generally known that reachability games are determined [KI16, p. 130].

### 2.3.1 Simulation and Bisimulation games

The (strong) simulation game's definition can be derived from the weak simulation game definition found in [BN19, Definition 2.30] by omitting the notion of weak transitions.

**Definition 2.20** (Simulation game)**.** For a labelled transition system $(\mathbf{Proc}, \mathbf{Act}, \rightarrow)$, the (strong) *simulation game* $\mathcal{G}_S[p_0] = (G, G_{\mathrm{d}}, \rightarrowtail, p_0)$ consists of

- *attacker nodes* $(P, Q)_{\mathrm{a}} \in G_{\mathrm{a}}$ with $P, Q \in \mathbf{Proc}$,
- *simulation defender nodes* $(\alpha, P, Q)_{\mathrm{d}} \in G_{\mathrm{d}}$ for situations where a simulation challenge for $\alpha \in \mathbf{Act}$ has been formulated,

and two kinds of moves

- *simulation challenges* $(P, Q)_{\mathrm{a}} \rightarrowtail (\alpha, P', Q)_{\mathrm{d}}$ if $P \xrightarrow{\alpha} P'$,
- *simulation answers* $(\alpha, P', Q)_{\mathrm{d}} \rightarrowtail (P', Q')_{\mathrm{a}}$ if $Q \xrightarrow{\alpha} Q'$,

This game mimics the concept of simulation: the attacker tries to find an action sequence for one process that can not be matched by the second process. The goal of the attacker is to refute that the first process can be simulated by the second one. On the other hand, the defender tries to prove that the second process can indeed simulate the first one by answering every move of the attacker with an appropriate action of the second process. Since reachability games are determined, the defender should always win the simulation game, provided that the initial position is in the defender winning region and the defender follows one of his/her winning strategies.

In [BN19, Proposition 2.31], it was proposed that the defender wins the *weak simulation game* $\mathcal{G}_{WS}[(P, Q)_{\mathrm{a}}]$ precisely if there exists a weak simulation $R_{WS}$ with $(P, Q) \in R_{WS}$. Analogously, it can be proposed that:

**Proposition 2.21.** *The defender wins* $\mathcal{G}_S[(P, Q)_{\mathrm{a}}]$ *precisely if* $P$ *is simulated by* $Q$.

Furthermore, [BNP20, Definition 19] defines the (strong) bisimulation game as follows:

**Definition 2.22** (Bisimulation game)**.** For a labelled transition system $(\mathbf{Proc}, \mathbf{Act}, \rightarrow)$, the (strong) *bisimulation game* $\mathcal{G}_B[p_0] = (G, G_{\mathrm{d}}, \rightarrowtail, p_0)$ consists of

- *attacker nodes* $(P, Q)_{\mathrm{a}} \in G_{\mathrm{a}}$ with $P, Q \in \mathbf{Proc}$,
- *simulation defender nodes* $(\alpha, P, Q)_{\mathrm{d}} \in G_{\mathrm{d}}$ for situations where a simulation challenge for $\alpha \in \mathbf{Act}$ has been formulated,

and three kinds of moves

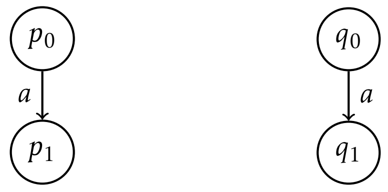
- *simulation challenges* $(P, Q)_{\mathrm{a}} \rightarrowtail (\alpha, P', Q)_{\mathrm{d}}$ if $P \xrightarrow{\alpha} P'$,
- *simulation answers* $(\alpha, P', Q)_{\mathrm{d}} \rightarrowtail (P', Q')_{\mathrm{a}}$ if $Q \xrightarrow{\alpha} Q'$,
- *symmetry moves* $(P, Q)_{\mathrm{a}} \rightarrowtail (Q, P)_{\mathrm{a}}$.

The bisimulation game adds the symmetric nature of bisimulations to the simulation game by allowing the attacker to ‘switch sides’ with *symmetry moves.* This way, the attacker can choose actions for the second process.

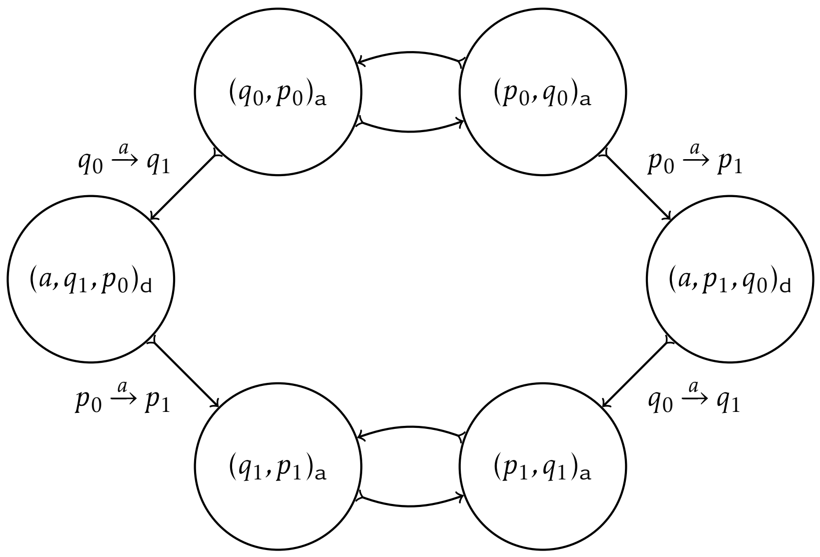
Similarly to proposition 2.21, it can be proposed that:

**Proposition 2.23.** *The defender wins* $\mathcal{G}_B[(P, Q)_{\mathrm{a}}]$ *precisely if* $P \leftrightarroweq Q$.

**Example 2.24.** For the subsequent LTS



a schematic representation of a bisimulation game as a graph might look like the following:


$(q_0, p_0)_a$
$(p_0, q_0)_a$
$q_0 \xrightarrow{a} q_1$
$p_0 \xrightarrow{a} p_1$
$(a, q_1, p_0)_d$
$(a, p_1, q_0)_d$
$p_0 \xrightarrow{a} p_1$
$q_0 \xrightarrow{a} q_1$
$(q_1, p_1)_a$
$(p_1, q_1)_a$

The right-hand side of the game graph describes simulation while the left-hand side — which can be attained with symmetry moves — lifts the game to bisimulation.

### 2.3.2 The Reactive Bisimulation Game

Coming now to labelled transition systems *with timeouts*, the following definition attempts to characterise reactive bisimulations as a game by augmenting the regular bisimulation game with rules derived from the *gsrb*-definition of reactive bisimulations (see 2.13). It utilises the property of *gsrb*s that triples $(P, X, Q)$ can only occur once a timeout transition $t$ has happened.

**Definition 2.25** (Reactive bisimulation game). For labelled transition systems with timeouts, the *reactive bisimulation game* $\mathcal{G}_{RB}[p_0] = (G, G_d, \rightarrowtail, p_0)$ extends the bisimulation game of definition 2.22 and consists of

- *attacker nodes* $(P, Q)_a \in G_a$ with $P, Q \in \mathbf{Proc}$,
- *simulation defender nodes* $(\alpha, P, Q)_d \in G_d$ for $\alpha \in (A \cup \{\tau\})$,
- *restricted attacker nodes* $(P, X, Q)_a \in G_a$ with $X \subseteq A$,
- *restricted simulation defender nodes* $(z, P, X, Q)_d \in G_d$ for $z \in \{\tau, t\}$,

and the moves

- *simulation challenges* $(P,Q)_\mathsf{a} \rightarrowtail (\alpha,P',Q)_\mathsf{d}$ if $P \xrightarrow{\alpha} P'$ and $\alpha \in A \cup \{\tau\}$,
- *simulation answers* $(\alpha,P',Q)_\mathsf{d} \rightarrowtail (P',Q')_\mathsf{a}$ if $Q \xrightarrow{\alpha} Q'$,
- *symmetry moves* $(P,Q)_\mathsf{a} \rightarrowtail (Q,P)_\mathsf{a}$,
- *timeout simulation challenges* $(P,Q)_\mathsf{a} \rightarrowtail (t,P',Y,Q)_\mathsf{d}$ if $P \xrightarrow{t} P'$ and $\mathcal{I}(P) \cap (Y \cup \{\tau\}) = \emptyset$ and $Y \subseteq A$,
- *timeout simulation answers* $(t,P',Y,Q)_\mathsf{d} \rightarrowtail (P',Y,Q')_\mathsf{a}$ if $Q \xrightarrow{t} Q'$,
- *restricted simulation challenges* $(P,Y,Q)_\mathsf{a} \rightarrowtail (a,P',Q)_\mathsf{d}$ if $P \xrightarrow{a} P'$ with either $a \in Y$ or $\mathcal{I}(P) \cap (Y \cup \{\tau\}) = \emptyset$,
- *invisible simulation challenges* $(P,Y,Q)_\mathsf{a} \rightarrowtail (\tau,P',Y,Q)_\mathsf{d}$ if $P \xrightarrow{\tau} P'$,
- *invisible simulation answers* $(\tau,P',Y,Q)_\mathsf{d} \rightarrowtail (P',Y,Q')_\mathsf{a}$ if $Q \xrightarrow{\tau} Q'$,
- *timeouted timeout simulation challenges* $(P,Y,Q)_\mathsf{a} \rightarrowtail (t,P',X,Q)_\mathsf{d}$ if $P \xrightarrow{t} P'$ and $\mathcal{I}(P) \cap (X \cup Y \cup \{\tau\}) = \emptyset$ with $X \subseteq A$,
- *restricted symmetry moves* $(P,Y,Q)_\mathsf{a} \rightarrowtail (Q,Y,P)_\mathsf{a}$.

In addition to the regular bisimulation game, the reactive bisimulation game features *restricted attacker* and *restricted simulation defender nodes*, meaning that a consequent action is restricted to the environment $Y$.

These nodes can first be attained by the attacker posing a *timeout simulation challenge.* A *timeout simulation challenge* models the case that the environment changes so that no other action than the timeout action $t$ is possible in some state (Definition 2.13, point 2). The defender can then match the challenge with a *timeout simulation answer.* The environmental restriction remains.

In the next position, there are four possible moves for the attacker. Firstly, he can choose to perform a visible action of the current process that is allowed by the environment. If no initial action of the process is allowed by the environment, the environment can be caused to change to an arbitrary environment after the process idling for a certain amount of time. In both of these two cases, a *restricted simulation challenge* move is made and the game arrives back in a regular *simulation defender node.* This ties close to definition 2.13, point 3.

Secondly, if a hidden $\tau$-action is possible for the current process, an *invisible simulation challenge* can be posed. It can only be answered by an *invisible simulation answer* by the defender. As a $\tau$-action is not observable, there is no change in the environment arriving in the next position (Definition 2.13, point 4).

Thirdly, if the environment $Y$ changes to another (or the same) environment $X$ after the process idling for a discrete amount of time, and there is still no action possible other than a timeout-action $t$, the attacker can choose to perform a *timeouted timeout simulation challenge.* The restricting environment for the next node changes to the new environment $X$ (Definition 2.13, point 5).

And finally, the attacker can choose to change sides in a restricted attacker node, similarly to normal *symmetry moves* for nonrestricted attacker nodes, and make a *restricted symmetry move.* This reflects the symmetric nature of *gsrb*-triples in Definition 2.13.

## 2.4 The Reactive Bisimulation Game Characterises Reactive Bisimilarity

**Proposition 2.26.** *States $p_0$, $q_0$ of a labelled transition system with timeouts are strongly reactive bisimilar if and only if the defender has a winning strategy in the reactive bisimulation game $\mathcal{G}_{RB}[(p_0, q_0)_\mathrm{a}]$.*

*Proof.* The following proof uses that reactive bisimilarity is expressed by the existence of a *gsrb* relation (Proposition 2.14).

- Construct the relation $\mathcal{R} = \{(p,q) \mid (p,q)_\mathrm{a} \in W_\mathrm{d}\} \cup \{(p,X,q) \mid (p,X,q)_\mathrm{a} \in W_\mathrm{d}\}$. This has to be a *generalised strong reactive bisimulation* (Definition 2.13).
  - Symmetry:
    1. (i) Assume $(p,q) \in \mathcal{R}$. As the attacker can play from $(p,q)_\mathrm{a}$ to $(q,p)_\mathrm{a}$ (*symmetry move*), the two can only be in the defender winning region together. Therefore $(q,p) \in \mathcal{R}$.
    2. (ii) Assume $(p,X,q) \in \mathcal{R}$. As the attacker can play from $(p,X,q)_\mathrm{a}$ to $(q,X,p)_\mathrm{a}$ (*restricted symmetry move*), the two can again only be in the defender winning region together. Therefore $(q,X,p) \in \mathcal{R}$.
  - Rule 1: Assume $(p,q) \in \mathcal{R}$ and $p \xrightarrow{\alpha} p'$ with $\alpha \in A \cup \{\tau\}$. Then the attacker can play from $(p,q)_\mathrm{a}$ to $(\alpha,p',q)_\mathrm{d}$ (*simulation challenge*). As $(p,q)_\mathrm{a}$ is in the defender winning region, there exists a $q'$ with $q \xrightarrow{\alpha} q'$ so that the defender can answer from $(\alpha,p',q)_\mathrm{d}$ to $(p',q')_\mathrm{a}$ (*simulation answer*) and $(p',q') \in \mathcal{R}$.
  - Rule 2: Assume $(p,q) \in \mathcal{R}$ and $p \xrightarrow{t} p'$ and also $\mathcal{I}(p) \cap (X \cup \{\tau\}) = \emptyset$ for some $X \subseteq A$. Then the attacker can play from $(p,q)_\mathrm{a}$ to $(t,p',X,q)_\mathrm{d}$ (*timeout simulation challenge*). As $(p,q)_\mathrm{a}$ is in the defender winning region, there has to exist a $q'$ with $q \xrightarrow{t} q'$ so that the defender can answer from $(t,p',X,q)_\mathrm{d}$ to $(p',X,q')_\mathrm{a}$ (*timeout simulation answer*) and $(p',X,q')_\mathrm{a} \in \mathcal{R}$.
  - Rule 3: Assume $(p,X,q) \in \mathcal{R}$ and $p \xrightarrow{a} p'$ with either $a \in X$ or $\mathcal{I}(p) \cap (X \cup \{\tau\}) = \emptyset$. Then the attacker can play from $(p,X,q)_\mathrm{a}$ to $(a,p',q)_\mathrm{d}$ (*restricted simulation challenge*). As $(p,X,q)_\mathrm{a}$ is in the defender winning region, there exists a $q'$ with $q \xrightarrow{a} q'$ so that the defender can play from $(a,p',q)_\mathrm{d}$ to $(p',q')_\mathrm{a}$ (*simulation answer*) and $(p',q') \in \mathcal{R}$.
  - Rule 4: Assume $(p,X,q) \in \mathcal{R}$ and $p \xrightarrow{\tau} p'$. Then the attacker can play from $(p,X,q)_\mathrm{a}$ to $(\tau,p',X,q)_\mathrm{d}$ (*invisible simulation challenge.* As $(p,X,q)_\mathrm{a}$ is in the defender winning region, there exists a $q'$ with $q \xrightarrow{\tau} q'$ so that the defender can play from $(\tau,p',X,q)_\mathrm{d}$ to $(p',X,q')_\mathrm{a}$ (*invisible simulation answer*) and $(p',X,q')_\mathrm{a}$ has to be in the defender winning region again, thus $(p',X,q') \in \mathcal{R}$.
  - Rule 5: Assume $(p,X,q) \in \mathcal{R}$ and $p \xrightarrow{t} p'$ and also $\mathcal{I}(p) \cap (X \cup Y \cup \{\tau\}) = \emptyset$ with $Y \subseteq A$. Then the attacker can play from $(p,X,q)_\mathrm{a}$ to $(t,p',Y,q)_\mathrm{d}$ (*timeouted timeout simulation challenge*). As $(p,X,q)_\mathrm{a}$ is in the defender winning region, there exists a $q'$ with $q \xrightarrow{t} q'$ so that the defender can play from $(t,p',Y,q)_\mathrm{d}$ to $(p',Y,q')_\mathrm{a}$ (*timeout simulation answer*) and $(p',Y,q') \in \mathcal{R}$.

This covers all moves an attacker can make in any game position. If $\mathcal{G}_{RB}[(p_0,q_0)_\mathrm{a}]$ is won by the defender, this means $(p_0,q_0)_\mathrm{a} \in W_\mathrm{d}$ and thus $(p_0,q_0) \in \mathcal{R}$. $\mathcal{R}$ is a gsrb as every tuple in $\mathcal{R}$ fulfills definition 2.13. Therefore it is shown that $(p_0, q_0)$ are strongly reactive bisimilar if the defender has a winning strategy in the reactive bisimulation game $\mathcal{G}_{RB}[(p_0, q_0)_\mathsf{a}]$.

- Assume $\mathcal{R}$ is a gsrb and $(p_0, q_0) \in \mathcal{R}$. Construct the defender strategy
  - $F((\alpha, p', q)_\mathsf{d}) = \{(p', q')_\mathsf{a} \mid \exists q'.(p', q') \in \mathcal{R} \land q \xrightarrow{\alpha} q'\}$,
  - $F((t, p', X, q)_\mathsf{d}) = \{(p', X, q')_\mathsf{a} \mid \exists q'.(p', X, q') \in \mathcal{R} \land q \xrightarrow{t} q'\}$,
  - $F((\tau, p', X, q)_\mathsf{d}) = \{(p', X, q')_\mathsf{a} \mid \exists q'.(p', X, q') \in \mathcal{R} \land q \xrightarrow{\tau} q'\}$.

  This has to be a winning strategy for the defender. Starting with $(p_0, q_0)_\mathsf{a}$, we prove that whatever the attacker plays and whatever moves the defender chooses from F, the following invariant is maintained: For attacker nodes $(p, q)_\mathsf{a}$, $(p, q) \in \mathcal{R}$, and similarly for restricted attacker nodes $(p, X, q)_\mathsf{a}$, $(p, X, q) \in \mathcal{R}$. Furthermore, for simulation defender nodes $(\alpha, p, q)_\mathsf{d}$ there will always be a $q'$ such that $q \xrightarrow{\alpha} q'$ and $(p, q') \in \mathcal{R}$ , and for restricted simulation defender nodes $(\mathsf{z}, p, X, q)_\mathsf{d}$ with $\mathsf{z} \in \{\tau, t\}$, there will always be a $q'$ such that $q \xrightarrow{\mathsf{z}} q'$ and $(p, X, q') \in \mathcal{R}$.

  - For the initial position $(p_0, q_0)_\mathsf{a}$ the invariant is valid because $(p_0, q_0) \in \mathcal{R}$.
  - Simulation challenges: At $(p, q)_\mathsf{a}$, the attacker moves to $(\alpha, p', q)_\mathsf{d}$ with $p \xrightarrow{\alpha} p'$ and $\alpha \in A \cup \{\tau\}$. By the invariant, $(p, q) \in \mathcal{R}$. As $\mathcal{R}$ is a gsrb, there must be a $q'$ with $q \xrightarrow{\alpha} q'$ and $(p', q') \in \mathcal{R}$ (gsrb-definition 2.13, rule 1). Therefore, the invariant is maintained.
  - Simulation answers: At $(\alpha, p, q)_\mathsf{d}$, the defender can, following strategy F, move to $(p, q')_\mathsf{a}$ if there is a $q'$ with $q \xrightarrow{\alpha} q'$ and $(p, q') \in \mathcal{R}$. According to the invariant there must exist such a $q'$ and thus the invariant holds for $(p, q')_\mathsf{a}$.
  - Symmetry moves: At $(p, q)_\mathsf{a}$, the attacker moves to $(q, p)_\mathsf{a}$. By the invariant, $(p, q) \in \mathcal{R}$. As $\mathcal{R}$ is a gsrb and therefore a symmetric relation, $(q, p) \in \mathcal{R}$ and the invariant is maintained.
  - Timeout simulation challenges: At $(p, q)_\mathsf{a}$, the attacker can move to $(t, p', X, q)_\mathsf{d}$ if $p \xrightarrow{t} p'$ and $\mathcal{I}(p) \cap (X \cup \{\tau\}) = \emptyset$ with $X \subseteq A$. By the invariant, $(p, q) \in \mathcal{R}$. If the attacker makes the move to $(t, p', X, q)_\mathsf{d}$, as $\mathcal{R}$ is a gsrb, then there must exist a $q'$ such that $q \xrightarrow{t} q'$ and $(p', X, q') \in \mathcal{R}$ (gsrb-definition 2.13, rule 2). The invariant is preserved.
  - Timeout simulation answers: At $(t, p, X, q)_\mathsf{d}$, the defender can, following strategy F, move to $(p, X, q')_\mathsf{a}$ if there is a $q'$ with $q \xrightarrow{t} q'$ and $(p, X, q') \in \mathcal{R}$. According to the invariant there must exist such a $q'$ and thus the invariant holds for $(p, X, q')$.
  - Restricted simulation challenges: At $(p, X, q)_\mathsf{a}$, the attacker can move to $(a, p', q)_\mathsf{d}$ if $p \xrightarrow{a} p'$ with either $a \in X$ or $\mathcal{I}(p) \cap (X \cup \{\tau\}) = \emptyset$. By the invariant, $(p, X, q) \in \mathcal{R}$. If the attacker makes the move to $(a, p', q)_\mathsf{d}$, as $\mathcal{R}$ is a gsrb, then there must exist a $q'$ such that $q \xrightarrow{a} q'$ and $(p', q') \in \mathcal{R}$ (gsrb-definition 2.13, rule 3). The invariant is maintained.
  - Invisible simulation challenges: At $(p, X, q)_\mathsf{a}$, the attacker moves to $(\tau, p', X, q)_\mathsf{d}$ with $p \xrightarrow{\tau} p'$. By the invariant, $(p, X, q) \in \mathcal{R}$. As $\mathcal{R}$ is a gsrb, there must be a $q'$ such that $q \xrightarrow{\tau} q'$ and $(p', X, q') \in \mathcal{R}$ (gsrb-definition 2.13, rule 4). Hence, the invariant holds.

- Invisible simulation answers: At $(\tau, p, X, q)_\mathrm{d}$, the defender can, following strategy F, move to $(p, X, q')_\mathrm{a}$ if there is a $q'$ with $q \xrightarrow{\tau} q'$ and $(p, X, q') \in \mathcal{R}$. According to the invariant there must exist such a $q'$. The invariant is preserved.
- Timeouted timeout simulation challenges: At $(p, X, q)_\mathrm{a}$, the attacker moves to $(t, p', X, q)_\mathrm{d}$ with $p \xrightarrow{t} p'$, $\mathcal{I}(p) \cap (X \cup Y \cup \{\tau\}) = \emptyset$ and $X \subseteq A$. By the invariant, $(p, X, q) \in \mathcal{R}$. As $\mathcal{R}$ is a gsrb, there must be a $q'$ such that $q \xrightarrow{t} q'$ and $(p', X, q') \in \mathcal{R}$ (gsrb-definition 2.13, rule 5).
- Restricted symmetry moves: At $(p, X, q)_\mathrm{a}$, the attacker moves to $(q, X, p)_\mathrm{a}$. By the invariant, $(p, X, q) \in \mathcal{R}$. As $\mathcal{R}$ is a gsrb and therefore a symmetric relation, $(q, X, p) \in \mathcal{R}$ and the invariant is maintained.

We have shown that the invariant is maintained whatever the attacker plays and whatever moves the defender chooses from F. This means that the defender can never get stuck following F in the game $\mathcal{G}_{RB}[(p_0, q_0)_\mathrm{a}]$. Hence, F is a winning strategy if two processes $p_0$ and $q_0$ are strongly reactive bisimilar.

□

# Chapter 3

# The Video Game

A video game was implemented as part of this thesis. It aims to plant the seed of intuition for the rather complex topic of reactive bisimilarity in players.

This chapter will shed light on to the development of the video game. The first section deals with important techniques that can be used to design a 'fun' and educational video game. Proceeding, a description of the general concept and implementation of the video game accompanying this thesis will be laid out. Finally, the chapter closes off with considerations about the design of the video game, precisely how it is tailored to create an engaging and instructional experience (e.g. by gradually increasing difficulty through levels).

## 3.1 Teaching through Video Games

Games can be seen as a fun "means to foster learners' understanding of theoretical models" [MS04, p. 26] and "have been useful in [...] supporting the development of critical thinking [and] problem solving" [MS04, p. 26]. According to Raph Koster, "fun is just another word for learning" [Kos13, p. 46]. This section details some of the reoccurring principles or techniques found in literature about game design that can be applied to design and develop fun and educational video games.

The first technique is called *flow*. The notion of flow has been subject of extensive study, one major contributor being Mihaly Csikszentmihalyi (e.g. with his work [Csi90]). Flow can be defined as a "feeling of complete and energized focus in an activity, with a high level of enjoyment and fulfillment" [Che07, p. 31]. In the state of flow, we lose track of time and our performance and focus in an activity is maximised. Flow emerges when a challenging activity lies on the line between boredom and anxiety. According to [Mur12], there are four requirements for flow to arise:

- *Clear tasks*: The player understands what he has to do.
- *Immediate feedback*: The player receives immediate feedback about the consequences of his/her actions and progression towards goals.
- *Balanced, attainable goal*: The tasks should be challenging but not too hard or long to achieve.

- *Concentration*: The game should avoid distracting the player away from the task, e.g. with complex interfaces.

Flow establishes the motivation to play a game. To keep the state of flow, the difficulty of the game should increase along with the growing skill of the player.

The second technique is *simplicity* [Mur12]. It ties into the concentration aspect of flow. Simplicity doesn't mean that the game objective should necessarily be simple. The right balance between complexity and simplicity must be found [Sch19, p. 238]. However, the game should be presented in a way that is as simple *as possible*. This entails intuitive-to-use user interfaces, screen layout, presentation of game mechanics, et cetera.

Another technique could be labelled as *immersion* or *engagement* [Mur12]. It means the perceived intensity of an activity or the game itself. Immersion and engagement can boost the overall interest and spark motivation in the player. Immersion can for example be achieved by magical or fantastical settings of a game. It ties into flow in the sense that the player can escape into another world and forget everything around him/her. Visual effects or background music can also support immersion. Moreover, engagement is achieved when the player has to actively solve problems in addition to passively experiencing immersion. Engagement can also be attained by including overarching *goals* like high scores to levels so that the player feels motivated to deeply explore a game [Sch19, p. 345].

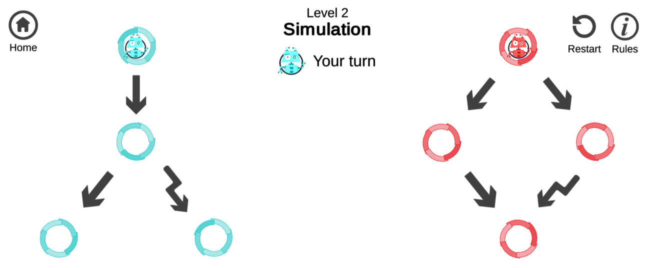
Meaningful *choices* can wake positive feelings of involvement in players [Mur12] and "enhance [...] the enduring interest of the game" [SZ08, p. 515]. However, too many choices could result in the player feeling overwhelmed.

*Practice* is especially important in educational video games. It means the repeated need of the player to use a certain skill [Mur12] in order to eventually achieve mastery.

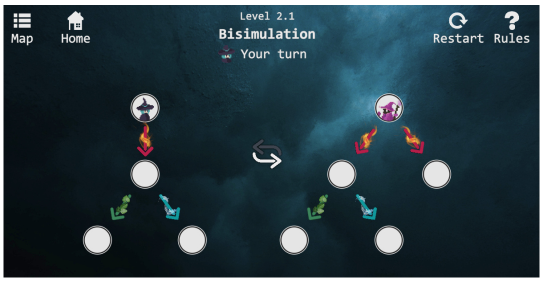
## 3.2 Game Description

The video game of this thesis is designed as a puzzle game where the player slips into the role of the attacker in equivalence games described in section 2.3. He/She has to find a sequence of moves that differentiates two systems modelled as an LTS. The defender is played by an algorithm that computes suitable next moves.

This game concept can also be found in another existing video game which has been developed by Dominik Andre Peacock as part of a bachelor's thesis titled "Process equivalences as a video game" [Pea20]. Peacocks's video game (see figure 3.1a) features multiple levels with varying LTSs, introducing different process equivalences to the player. The development of the video game of this thesis took inspiration from Peacock's game. The core game concept was taken over and augmented with the notion of reactive bisimilarity as well as new levels and features.



(a) Level in Peacock's game



(b) Level in this thesis' game

Figure 3.1: Video games in comparison

In the video game of this thesis, the player starts on the left side of the LTS. He/She is the attacking player and can click on adjacent nodes to move to another state. In the *bi*simulation levels (see figure 3.1b) the player also has access to a symmetry move button in the middle of the screen to swap positions with the defender. Finally, in the *reactive* bisimulation levels, timeout transitions are introduced. Every time the player tries to make such a transition, a small popup appears with which the player can select visible actions to disable or enable with the timeout transition. This symbolises the selection of environments for *timeout simulation challenges* or *timeouted timeout simulation challenges* of the theoretical game (definition 2.25).

The player wins the game if the defender is stuck. A popup then appears, telling the player how many moves he/she needed to solve the level and a score of one to three stars, three stars indicating that the player found the best solution with the least possible amount of moves. An algorithm for computing this number, given that the defender tries to maximise the number of moves the attacker needs, is introduced in chapter 4.

Solving a level also unlocks the next level. The player can select which level to play from the level map. It shows which levels are unlocked and how many stars have been earned for each level.

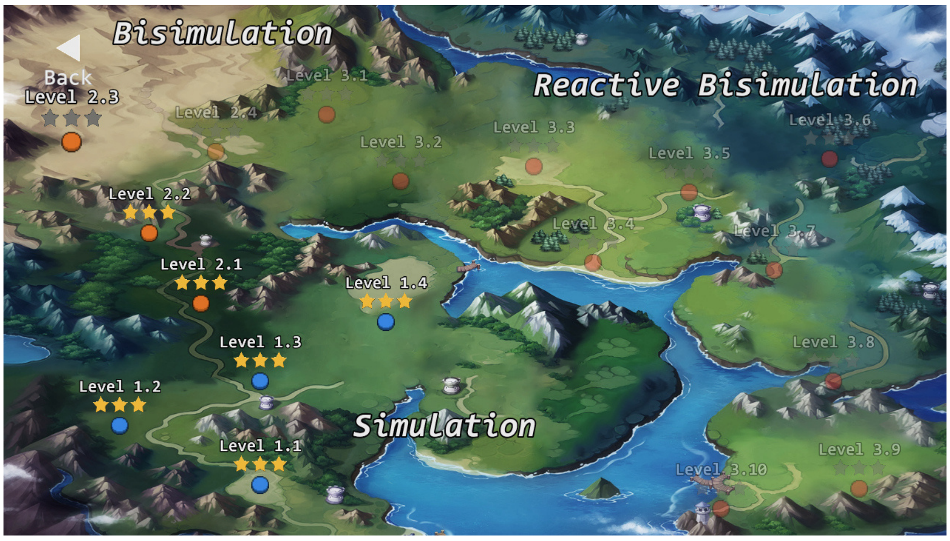


Figure 3.2: Level map

The levels are arranged in an order that gradually introduces more mechanics and difficulty to the gameplay. It starts off with the simulation game, followed by the bisimulation game and finally the reactive bisimulation game. These games conveniently build on each other. The levels about the reactive bisimulation game first introduce timeout transitions followed by hidden $\tau$-transitions a bit later.

To guarantee that the functionality of the video game ties close to the theoretical reactive bisimulation game, a model symbolising LTS and the theoretical rules of the equivalence games mentioned in section 2.3 was implemented in typescript code.

## 3.3 The Game's Design

Following up, the design decisions made while developing the video game of this thesis are discussed in the context of design principles introduced in section 3.1. The reader is encouraged to try out the video game[1] before proceeding.

### 3.3.1 Attacker or Defender

One of the most important design decisions is whether the player should take the role of the attacker or the defender. [Pea20, p. 24] argues that while the defender needs to have an answer to a multitude of future possible game positions, the attacker only needs to find one move sequence to complete its objective. The attacker would therefore be more suited because it should be *simpler* and easier to teach to the player. The defender is also more difficult to design a video game around because of the possibly infinite nature of its win conditions (the defender wins infinite plays). Furthermore, the role of the attacker may give the player more *meaningful* choices because it firstly has more available moves and secondly dictates the course of the game while the defender only reacts to the attacker. The gameplay of the attacker could thus involve more active thinking (by needing to find cases where two systems are not equivalent) and be more *engaging* in contrast to the more passive, reactive playstyle of the defender. That is why in the video game of this thesis, the player takes the role of the attacker.

### 3.3.2 Visual Presentation

The video game of this thesis is set in a fantasy universe with the attacker and defender being embodied by magicians. The actions between the states of an LTS are symbolised by different magic spells. For example, the attacker performing an action may be symbolised by the attacker magician casting a fire magic spell. This draws on the technique of *immersion* and may make the game seem more interesting to the player than if he/she just saw plain LTSs. To enforce the immersion, the video game displays visual magic spell animations when a magic spell is performed (a transition occurs).

[1]Link to the game: `https://eloinoel.github.io/ReactiveBisimilarityGame/`

The first approach while trying to find an appropriate visual representation for the action-labelled edges between the states of an LTS, was to display different icons next to an edge, symbolising a magic spell (e.g. a fire icon for fire magic). However, this proved to be obstructing the *flow* of the game by being somewhat difficult to read. In the end, another approach was taken where each action is represented by an arrow of distinct color. Additionally, arrows for different actions also slightly differ in their form. For instance, an arrow symbolising an action that is represented by fire magic is surrounded by a fire effect. On the whole, the information of the action is visually merged with the edge.

### 3.3.3 Difficulty

As previously described, the levels of the video game are arranged and unlocked in an order that gradually increases the difficulty for the player. When a new rule is introduced, the LTS of a level will have a reduced amount of choices (outgoing transitions per state), so that the player can focus on and learn the new mechanic. More choices and diverse selections of LTSs are then added in subsequent levels to raise the difficulty and letting the player *practice* the newly learned skill. New rules are introduced, when an 'appropriate' amount of use cases has been covered for one concept.

Compared to the transition from the simulation game to the bisimulation game, the jump from bisimulation to reactive bisimulation is very big. The reactive bisimulation game is fairly complex and introduces many new game moves in contrast to for example the bisimulation game which only adds symmetry moves to the simulation game. The subtleties of the reactive bisimulation game had to be split up and fed to the player bit by bit over multiple levels. For example, the goal for the first reactive bisimulation level is to introduce timeout transitions (in the video game they are called time magic spells) and make the player accustomed to the operation of the new user interface for setting environments. The third reactive bisimulation level then introduces the idea that any visible action is allowed if the system is idling (restricted simulation challenges) while the fifth level deals with timeouted timeout simulation challenges (definition 2.25).

### 3.3.4 Clear tasks

Naturally, the video game has to provide guidance to the player to avoid artificially created difficulty caused by the lack of knowledge of the player about a functionality or how this functionality can be accessed and operated. Three features attempt to solve this issue. Firstly, each time a new mechanic is introduced, an overlay will be displayed over the current level. This overlay contains text and images and explains the new mechanic. Secondly, after closing the overlay the level might visually highlight what the player is supposed to click by for example making buttons pulsate. Lastly, if the player still needs help, he/she can click on a 'Rules' button to get an alternative description of the rules.

### 3.3.5 Feedback

After completing each level, the player receives *feedback* for his performance through a popup. The performance is measured in how many moves the attacker used to solve the level. Depending on the number of moves, the player is awarded with up to three stars. One star signifies an improvement worthy solution, two stars mean the second best solution was found and three stars denote that there is no better solution. The stars evaluation acts as a high score which adds a secondary *goal* to the game. The earned stars are displayed on the level map. It adds another motivation to continue playing the game and a sense of progression and reward. One issue with the concept of this video game is that almost all of the levels can be solved by just randomly trying out different move sequences without thinking about it too much. A performance evaluation encourages a deeper exploration of the game contents where the player needs to actively think about the levels to solve them perfectly. The game may become more *engaging*.

To make a game feel responsive, every action of the player should be answered with some form of *feedback*. The video game at hand does so by giving visual feedback. If a player wants to perform a move in a level, he/she can either observe a magic spell animation, the current state of the player is updated or, if the move was not possible, the clicked state will blink in red. Furthermore, when the player hovers over, clicks or releases a button or a part of interactive user interface, he/she receives immediate visual feedback for that.

As a funny little feature, a button to restart the level will start pulsating when the current game position is in the defender winning region (the player cannot win anymore). This feature was adopted from [Pea20].

To make it more clear to the player what the effect of his/her environment selection is when choosing to perform a timeout transition and why some environment selection may not be valid, a feature was implemented that grays out all outgoing transitions of the current state that become impossible by the current selection (see figure 3.3).

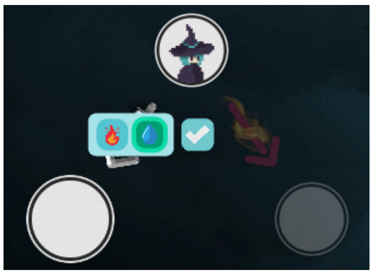


Figure 3.3: Fire magic is disabled by the selection and the fire transition is grayed out. The timeout transition on the left is possible with the selected environment and is therefore not grayed out

### 3.3.6 Further considerations

To afford *simplicity* not only in the design of the video game but also its delivery, and in order to make the barrier of entry as small as possible, the game is accessible over the internet as a browser game. [2] Therefore, users do not have to download the source code or install any additional software to make it work.

User feedback is very important during the process of developing a video game [RW13] because developers cannot fully grasp how new players will approach and view a game. A small group of people with varying backgrounds (not only computer science related) were asked to test the game. The feedback aided the development in finding clearer explanation texts for the user guidance features in the game. Furthermore, the idea of visualising the effect of an environment selection on screen was voiced and resulted in the graying-out-feature of transitions that was previously described. Moreover, the reactive bisimulation game levels seemed to sometimes leave the player with some level of confusion due to the player being overwhelmed with information. A few more reactive bisimulation levels were stuffed in between the levels introducing new information to alleviate this issue. One tester also commented on an early version of the game that "magic spell animations and sound effects would go a long way" in making the game more interesting. That is how the spell animations came into existence. However, it was decided against using sound effects to accompany the magic spell animations because it was hard finding fitting sounds that would not break the immersion.

[2] `https://eloinoel.github.io/ReactiveBisimilarityGame/`

# Chapter 4

# Algorithms

This chapter details three noteworthy algorithms used in the video game of this thesis. The first algorithm generates a game graph of all reachable game positions in order for a defender algorithm to be able to compute what moves to perform after the player made a move. The second algorithm computes the shortest path through the game graph to a node where the defender is stuck, given that the defender always performs moves to maximise the length of this path. This path is utilised to evaluate the performance of the player. The third algorithm is the aforementioned defender algorithm.

## 4.1 Generating Game Position Graphs

Peacock talks about an algorithm to generate a graph representing all possible courses a game could take [Pea20]. A game graph is advantageous to have for algorithms that need to compute something in the game. It is used for the other two algorithms that will be discussed in this chapter. The idea of generating game graphs was adapted and resulted in a new version of the algorithm that is tailored to the reactive bisimulation game. The algorithm computes a directed graph of game positions with unweighted edges. Node $a$ has an edge to node $b$ if, in the game the graph is representing, there exists a move from the game position that coincides with $a$ to the game position that coincides with $b$. A visualisation for a game graph can be found in example 2.24.

**Algorithm 1**

```
1: function generateGameGraph($\mathcal{G}[p_0] = (G, G_d, \rightarrowtail, p_0)$)
2:     graph = initialiseEmptyGraph()
3:     graph.addNode($p_0$)
4:     appendNodesRecursively($p_0$, graph, $\mathcal{G}[p_0]$)
5:     return graph
6:
7: function appendNodesRecursively($p$, graph, $\mathcal{G}[p_0]$)
8:     nextPositions = getPossibleMoves($p$, $\mathcal{G}[p_0]$)
9:     for each position $p_i \in$ nextPositions do
10:        if graph.hasNode($p_i$) then
11:            if not graph.hasEdge($p$, $p_i$) then
12:                graph.addEdge($p$, $p_i$)
13:        else
14:            graph.addNode($p_i$)
15:            graph.addEdge($p$, $p_i$)
16:            appendNodesRecursively($p_i$, graph, $\mathcal{G}[p_0]$)
```

The graph is created recursively (line 4) from the starting position of a game in depth first search [Tar72] fashion. Each node (representing a game position) is expanded by looping through all moves (line 9) that are possible in that position. Moves in this context mean optional next game positions from the position the game is currently in. When calculating the moves that are possible in one game position (line 8), one needs to look at all possible combinations of visible actions for environments for attacker moves involving a timeout action. If a next game position node was not visited yet, it is explored recursively (line 16). The algorithm should also handle the case of cycles or multiple visitations of a node (line 10). For games with a finite amount of game positions, the algorithm should terminate because no recursive call occurs when a node was already visited (the graph already contains the node).

## 4.2 Minimum Number of Moves Algorithm

Instead of manually determining the number of moves needed to get three stars for each level, an algorithm was developed to compute it. The goal of the algorithm is to find a path through the game graph to a leaf node that is in the attacker winning region and where the defender is stuck. The algorithm needs to take into account that the defender may choose moves that make the path to such a leaf node longer. Thus, the algorithm needs to look at the 'worst case shortest path' for the attacker. Since the defender does not have consecutive moves, such a worst case shortest path entails the least amount of moves the attacker can make to win the game, given that the defender tries to maximise the length of this path.

The functionality of the algorithm is similar to a minimax algorithm [CM83] which can be used to compute optimal strategies for players of perfect information games where there are two players playing against each other. Each node of a game graph can be assigned an evaluation score, which depends on whose player's turn it is. One player tries to minimise this score while the other player tries to maximise it.

The modified minimax algorithm of this thesis should be called with a node that is in the attacker winning region and therefore always has at least one solution that is reachable after a finite amount of moves. Furthermore, the game graphs for the video game of this thesis contain cycles that should be taken care of.

**Algorithm 2**

```
 1: function modifiedMiniMax(node, currentPath)
 2:     if currentPath.containsNode(node) then                ▷ Cycle detection
 3:         return (Infinity, [])
 4:     if node.isLeaf() then
 5:         return (1, [node])
 6:
 7:     if node.isDefenderPosition() then                     ▷ Maximising player
 8:         maxEval = (0, [])
 9:         for each child c of node do
10:             tmpEval = modifiedMiniMax(c, [currentPath + node])
11:             if tmpEval[0] > maxEval[0] then
12:                 maxEval = tmpEval
13:         return (maxEval[0] + 1, [maxEval[1] + node])
14:     else                                                  ▷ Minimising player (attacker)
15:         minEval = (Infinity, [])
16:         for each child c of node do
17:             if c.inDefenderWinningRegion() then
18:                 continue
19:             tmpEval = modifiedMiniMax(c, [currentPath + node])
20:             if tmpEval[0] < minEval[0] then
21:                 minEval = tmpEval
22:         return (minEval[0] + 1, [minEval[1] + node])
```

The algorithm at hand traverses the game graph in a depth first search [Tar72] manner. However, one property of the algorithm is that it only visits nodes that are in the attacker winning region (if it is initially called with an attacker winning region node) as the wanted solution should also be in this region. The algorithm works recursively with line 2 being the first recursion anchor and line 4 the second. The returned values of the algorithm are tuples with the first entry being the number of moves it takes to get to a leaf node and the second entry symbolising the path from this leaf node to the node the function was called from. The first recursion anchor returns infinity and an empty path if a cycle is encountered. The path is not important in this case. The second recursion anchor returns if a leaf node is encountered. Since the algorithm stays in the attacker winning region, this leaf node must be a defender node.

The evaluation score of each non leaf node is dependant on whether it is the attackers turn or the defenders turn (lines 7 and 14). It is the minimal path cost to the 'nearest' defender node where the defender is stuck, given that the defender tries to maximise the length of this path. All children (or neighbours) of the current node are investigated (lines 9 and 16). For instance, the defender will in his turn choose the move (or neighbouring node) that has the highest evaluation score (line 11) because he/she is the maximising player. This will then be the evaluation score of the current node. A tuple is returned that contains the cost of the best neighbour plus one (see figure 4.1), and the path which that neighbour holds with the current node added to that path (lines 13 and 22). The defender cannot choose a move that leads into the defender winning region, otherwise the current node would also be in the defender winning region. The attacker also does not expand into nodes that are in the defender winning region (line 17). An algorithm for computing winning regions for defender and attacker in a game graph can be found in [BN19, p. 42].

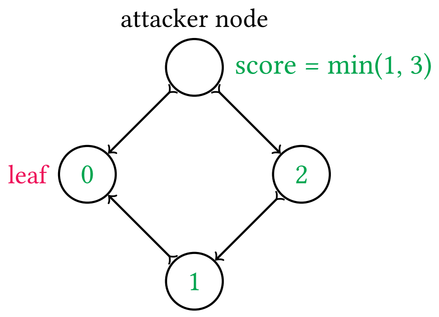


Figure 4.1: Evaluation score in green for each node of the exemplary game graph

The algorithm should terminate on game graphs with a finite number of nodes and edges. Cycles in the defender winning region are not visited and the cycle detection of line 2 prevents cycles in the expansion path. Furthermore, all paths not containing a cycle need to end in a leaf and line 4 of the algorithm covers this actuality. Proving the correctness of the algorithm would go beyond the scope of this thesis but the results appear correct when running it on levels of the video game. The expected number of moves for the stars score needs to be determined from the returned path because the path also contains game positions from defender moves. This is done by counting the number of defender positions in the path and then incrementing this number by one for every two consecutive attacker positions in the path (symmetry moves). The reason why the algorithm is made in a way to not only return a number but also a path is because it is used for the defender algorithm of the video game as well.

The algorithm utilises winning regions to reduce the amount of nodes that have to be visited. The winning regions can be computed once for each level of the game. On the other hand, the modified minimax algorithm is called multiple times each game because the defender algorithm makes use of it.

## 4.3 Defender Algorithm

The defender algorithm chooses suitable game moves for the defender. It should be tied to the previously talked about minimum-number-of-moves algorithm (section 4.2) because when a player solution to a level is evaluated and the player is awarded a number of stars, this evaluation is dependant on the choices the defender makes. The defender will thus perform moves that have been computed for the defender by the modified minimax algorithm of the previous section, as long as the current game position is in the attacker winning region. The defender's behaviour can be described as stalling out the attacker as much as possible until the attacker makes a mistake and the game switches to a defender winning region position. At that point, the algorithm of the previous section will stop being of use. The defender can then pick random moves, as long as the resulting game positions stay in the defender winning region. The player cannot complete his/her objective anymore and needs to restart the level.

### 4.3.1 Discussion of the Defender Algorithm

In the video game of this thesis, the adversary of the human player cannot win the game if the player chooses the right moves (the game starts in the attacker winning region). Still, the defender algorithm should play moves that 'make sense' and challenge the player so that it feels rewarding to beat a level.

There exist multiple possibilities of how the defender could behave to satisfy this criterion. [Pea20] employs a strategy where the defender chooses moves that lead to the nearest defender winning region node in the game graph. Another possibility could be that the defender chooses moves in a way that the probability of the player making a mistake is maximised. This could mean that a chosen subsequent node has the highest number of future choices that could lead into the defender winning region. The employed strategy discussed in section 4.3 was picked because it is directly linked to the scoring evaluation of a players performance.

If the player makes a mistake and the game arrives in a position that is in the defender winning region, the defender should play 'optimally' and not make mistakes that lead back to an attacker winning region position. Otherwise, the behaviour of the defender could seem illogical to the player and to win a level this way might feel less rewarding. Furthermore, the defender making mistakes could defeat the purpose of the reactive bisimulation game. As the defender wins all infinite plays, choosing random moves that make the game stay in the defender winning region is adequate.

# Chapter 5

# Conclusion

This thesis has presented a characterisation of reactive bisimilarity as a game and the implementation of a video game based on this characterisation.

The reactive bisimulation game offers another possibility of determining whether two processes $p$ and $q$ are reactive bisimilar or not. For instance, it may be more accessible to play the reactive bisimulation game to show that $p$ and $q$ are not reactive bisimilar than proving for each relation by definition 2.10 that if it contains $(p, q)$ then it cannot be a reactive bisimulation.

Furthermore, this thesis contains proof that the presented characterisation of reactive bisimilarity as a game coincides with its respective notion of equivalence (proposition 2.26). The proof is relatively overseeable because the rules of the reactive bisimulation game tie closely to the *gsrb*-definition of reactive bisimulations. It would be interesting to explore if in further work a more concise representation of reactive bisimulations as a game could be found.

The video game accompanying this thesis serves as a means to awaken an intuition for the rather complex topic of reactive bisimilarity in players. We have discussed how common game design principles have been applied with the goal of making the video game fun and educational. One way has been to raise the difficulty of the video game step by step as the players understanding grows while playing. The video game could thus for example be used as a tool to introduce students of computer science into the ways of reactive bisimilarity in a playful manner. Future work could include adapting other process equivalences such as different kinds of trace [Gla90], simulation [Gla90] and bisimulation [GW96][FG16] equivalences to the medium of video games.

This thesis put forward three algorithms that were initially developed for the video game. The first algorithm that was discussed can be used to construct a game graph that models all possible turns one of the logical games presented in section 2.3 could take. Creating the game graph is mainly dependant on which moves are possible in each game position. Therefore, by modifying the method that calculates possible moves for each node, the algorithm could be adapted to also be used for generating game graphs for logical games of other process equivalences that were not discussed in this thesis.

The other two algorithms presented in chapter 4 operate on game graphs generated by the first algorithm. The first of the two remaining algorithms finds a shortest path through the game graph to a leaf node where the defender loses, given that the defender tries to maximise the length of this path. The other algorithm chooses next moves for the defender.

Both of these algorithms could be utilised for logical games of other process equivalences as well, as long as winning regions can be computed for the game graphs (e.g. with the algorithm from [BN19, p. 42]).

It could be intriguing to find other game concepts that could present reactive bisimilarity in a video game format. For example, the video game of this thesis has not touched *X-bisimilarity* and only focuses on showing that two processes are not reactive bisimilar. One could also play the reactive bisimulation game from the perspective of the defender. It would then be important to design the game in a way to avoid the role of the player becoming too passive.

# Bibliography

[AI05] L. Aceto and A. Ingólfsdóttir. *The Equational Logic of Parallel Processes.* Research project proposal. 2005.

[Ace+07] L. Aceto, A. Ingólfsdóttir, K. G. Larsen and J. Srba. *Reactive Systems: Modelling, Specification and Verification.* Cambridge University Press, 2007. DOI: 10.1017/CBO9780511814105.

[Gla22] R. J. van Glabbeek. "Reactive bisimulation semantics for a process algebra with timeouts". In: *Acta Informatica* (Apr. 2022). DOI: 10.1007/s00236-022-00417-1.

[BNP20] B. Bisping, U. Nestmann and K. Peters. "Coupled similarity: the first 32 years". In: *Acta Informatica 57* (Oct. 2020), pp. 439–463. DOI: 10.1007/s00236-019-00356-4.

[MS04] A. Mitchell and C. Savill-Smith. *The use of computer and video games for learning. A review of the literature.* Learning and Skills Development Agency, 2004. ISBN: 1853389048.

[BN19] B. Bisping and U. Nestmann. "Computing Coupled Similarity". In: *Tools and Algorithms for the Construction and Analysis of Systems.* Ed. by T. Vojnar and L. Zhang. Cham: Springer International Publishing, 2019, pp. 244–261.

[GH15] R. J. van Glabbeek and P. Höfner. "CCS: It's not fair!" In: *Acta Informatica* 52.2-3 (Mar. 2015), pp. 175–205. DOI: 10.1007/s00236-015-0221-6.

[Mil90] R. Milner. "CHAPTER 19 - Operational and Algebraic Semantics of Concurrent Processes". In: *Formal Models and Semantics.* Ed. by J. van Leeuwen. Handbook of Theoretical Computer Science. Amsterdam: Elsevier, 1990, pp. 1201–1242. ISBN: 978-0-444-88074-1. DOI: https://doi.org/10.1016/B978-0-444-88074-1.50024-X.

[Gla90] R. J. van Glabbeek. "The linear time - branching time spectrum". In: *CONCUR '90 Theories of Concurrency: Unification and Extension.* Ed. by J. C. M. Baeten and J. W. Klop. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 278–297.

[Poh21] M. Pohlmann. "Reducing Strong Reactive Bisimilarity to Strong Bisimilarity". Technische Universität Berlin, June 2021.

[GS53] D. Gale and F. M. Stewart. "13. Infinite Games with Perfect Information". In: *Contributions to the Theory of Games (AM-28), Volume II.* Ed. by H. W. Kuhn and A. W. Tucker. Princeton University Press, 1953, pp. 245–266. DOI: doi:10.1515/9781400881970-014.

[BJN21] B. Bisping, D. N. Jansen and U. Nestmann. *Deciding All Behavioral Equivalences at Once: A Game for Linear-Time–Branching-Time Spectroscopy.* 2021. DOI: 10.48550/ARXIV.2109.15295.

[KI16] I. Khaliq and G. Imran. “Reachability Games Revisited”. In: *SOFTENG 2016* (2016), pp. 129–132.

[Kos13] R. Koster. *A Theory of Fun for Game Design.* Second edition. O’Reilly Media, Inc., 2013. ISBN: 9781449363215.

[Csi90] M. Csikszentmihalyi. *Flow: The Psychology of Optimal Experience.* New York: Harper & Row, Jan. 1990.

[Che07] J. Chen. “Flow in games (and everything else)”. In: *Communications of the ACM* 50.4 (2007), pp. 31–34.

[Mur12] C. Murphy. “Why games work and the science of learning”. In: *Selected Papers Presented at MODSIM World 2011 Conference and Expo.* 2012.

[Sch19] J. Schell. *The Art of Game Design: A Book of Lenses.* Third edition. New York: CRC press, 2019. ISBN: 9781138632097. DOI: https://doi.org/10.1201/b22101.

[SZ08] M. Song and S. Zhang. “EFM: A Model for Educational Game Design”. In: *Technologies for E-Learning and Digital Entertainment.* Ed. by Z. Pan, X. Zhang, A. El Rhalibi, W. Woo and Y. Li. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 509–517. ISBN: 978-3-540-69736-7.

[Pea20] D. A. Peacock. “Process equivalences as a video game”. Technische Universität Berlin, May 2020.

[RW13] R. Ramadan and Y. Widyani. “Game development life cycle guidelines”. In: *2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS).* 2013, pp. 95–100. DOI: 10.1109/ICACSIS.2013.6761558.

[Tar72] R. Tarjan. “Depth-First Search and Linear Graph Algorithms”. In: *SIAM Journal on Computing* 1.2 (1972), pp. 146–160. DOI: 10.1137/0201010.

[CM83] M. S. Campbell and T. Marsland. “A comparison of minimax tree search algorithms”. In: *Artificial Intelligence* 20.4 (1983), pp. 347–367. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(83)90001-2.

[GW96] R. J. van Glabbeek and W. P. Weijland. “Branching Time and Abstraction in Bisimulation Semantics”. In: *J. ACM* 43.3 (May 1996), pp. 555–600. DOI: 10.1145/233551.233556.

[FG16] W. Fokkink and R. Glabbeek. “Divide and Congruence II: Delay and Weak Bisimilarity”. In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science.* LICS ’16. 2016, pp. 778–787. DOI: 10.1145/2933575.2933590.