Technische Universität Berlin
Fakultät iv - Elektrotechnik und Informatik
Modelle und Theorie Verteilter Systeme

Bachelor's Thesis

Linear-Time–Branching-Time Spectroscopy as an Educational Web Browser Game

Mariusz Trzeciakiewicz
ID: 403298

Supervised by
Prof. Dr. Uwe Nestmann
Prof. Dr. Stephan Kreutzer

Berlin
November 2021

Declaration of independence / Selbständigkeitserklärung

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:

Mariusz Trzeciakiewicz

Signature

Place, Date

Acknowledgements
I would like like to show gratitude to Benjamin Bisping for guiding me throughout my thesis.

Abstract

The main goal of my thesis is to develop a video game based on the spectroscopy game. The video game, titled "The Spectroscopy Invaders", that I present can be played by a human player and aims to be engaging, educational, and fun.

This thesis first describes process equivalences and explains how one can distinguish two processes using Hennesy-Milner Logic formulas. Then I will introduce the reader to the Linear-time-Branching-time spectrum, and the spectroscopy game along with the spectroscopy procedure, which finds the finest equivalence for two processes. This paper concludes by giving an overview of the design of the "The Spectroscopy Invaders" game, and by explaining the most important decisions made during the implementation process.

The video game has been implemented in TypeScript using the Phaser 3 game framework.

Zusammenfassung

Das Hauptziel meiner Bachelorarbeit ist es ein Computerspiel zu entwickeln, das auf Spektroskopie Spiel[1] basiert. Das Spiel, mit dem Titel "The Spectroscopy Invaders", ist ein Lernspiel, welches von einem Spieler gespielt werden kann.

Diese Arbeit beschreibt zunächst Prozessäquivalenzen und erklärt wie zwei Prozesse mittels Hennesy-Milner-Logik voneinander unterschieden werden können. Danach stellt die Arbeit das "Linear-time−Branching-time spectrum" und das Spektroskopie Spiel zusammen mit der Spektroskopie Prozedur vor. Abschließend gebe ich einen Überblick über den Entwurf vom "The Spectroscopy Invaders" Spiel und einer Erklärung über die wichtigsten Entscheidungen, die ich während der Implementierung getroffen habe.

[1] Original: "The spectroscopy game".

Contents

Chapter 1

Introduction

Even if two system models show different behaviors, under some assumptions, they might still behave equivalently. In computer science, we often equate two system models using notions of behavioral equivalences. There are many different equivalences and each of them requires two models to share specific similarities in order to be considered the same (equivalent).

1.1 The Spectroscopy Game

A way to prove that two models show different behaviors, with respect to a chosen equivalence, is to develop a Hennesy-Milner Logic formula, syntax of which reflects the similarities defined by the equivalence, that distinguishes the two models. If there is no such a formula, then the models must be equivalent. Some equivalences are finer than the other ones and in order to find the finest equivalence for two models, it is necessary to decide successively multiple equivalences for them. For each equivalence, a different approach is needed and the whole process can be very time-consuming. In order to make it easier and faster, Bisping and Nestmann sought a way to decide several equivalences at once. As a result, they have presented the spectroscopy game along with the spectroscopy procedure, which can find the set of Hennesy-Milner Logic formulas that distinguish two models. One can then use the formulas to find the finest equivalence for the models.

1.2 This Thesis

The main goal of this paper is to develop a video game based on the spectroscopy game and justify the decisions made during the implementation process. This work makes the following contributions:

- I *adjust the spectroscopy game* so it can be played by a human player as the attacker.
- I *design and implement a video game* based on the adjusted spectroscopy game.

To give a suitable foundation for understanding the spectroscopy game, chapter 2 firstly introduces necessary definitions to describe a computing model, then explains chosen equivalences and introduces distinguishing formulas, and finally present the linear-time−branching-time spectrum along with formula pricing system. Chapter 3 focuses mainly on the spectroscopy game and explains how to compute the cheapest distinguishing formulas. Chapter 4 describes the implementation process and the design of "The Spectroscopy Invaders" game, justifies my choices, and compares "The Spectroscopy Invaders" with other similar video games.

1.3 Accompanying Artifacts

"The Spectroscopy Invaders" game that I have implemented as a part of this work, has been developed in TypeScript[1] using the Phaser 3[2] framework. The source code[3] can be downloaded from the following GitHub repository `https://github.com/Marii19/the-spectroscopy-invaders`. In order to successfully install and run "The Spectroscopy Invaders", please follow the instructions in the README file.

[1] TypeScript official website `https://www.typescriptlang.org/` accesed: 27.11.2021
[2] Phaser 3 official website `https://phaser.io/phaser3` accesed: 27.11.2021
[3] Source code can be also found on an USB flash drive attached with this thesis.

Chapter 2

Theory of Behavioral Equivalences

This chapter introduces essential definitions required to understand this thesis's further sections. The chapter begins with an explanation of Labeled Transition System, the Calculus of Communicating Systems, and Hennessy-Milner logic. Then, I will define several notions of behavioral equivalences. The chapter concludes by introducing HML Semantics, distinguishing formulas, and the linear-time-Branching-time spectrum.

2.1 Description of Computing Systems

2.1.1 Transition Systems

An abstract way of understanding the behavior of a computing system is to portray it as a weighted, directed graph, with its nodes representing states [AILS07]. Such a graph is called a Labeled Transition System, short LTS. Outgoing labeled edges of a state represent what interactions are possible from that state. In the LTS terminology, interactions are called transitions [San11].

Definition 2.1 (LTS)**.** [AILS07] defines LTS as a triple $(S, \Sigma, \rightarrow)$ where

- S is a non-empty set of states,
- Σ is the set of actions, and
- $\rightarrow \subseteq S \times \Sigma \times S$ is the transition relation.

Instead of $(p, \alpha, p') \in \rightarrow$, one can just write $p \xrightarrow{\alpha} p'$. If for a chosen state p and an action α there is no $p' \in S$ so that $p \xrightarrow{\alpha} p'$, then one can write $p \not\xrightarrow{\alpha}$.

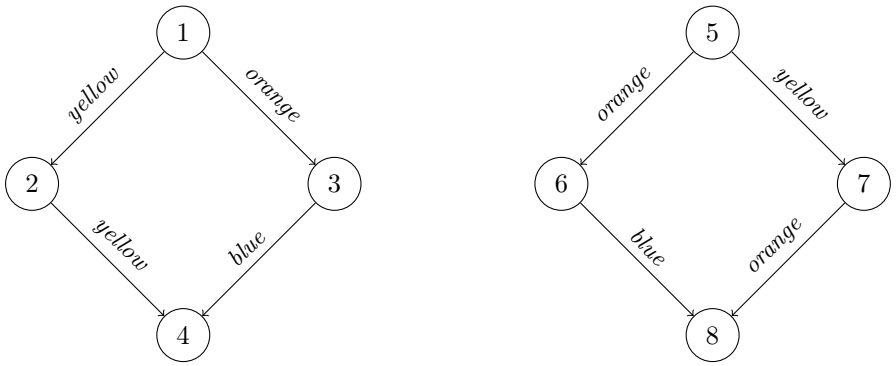


Figure 2.1: Example of an LTS

2.1.2 Calculus of Communication Systems

Another way of describing a computing system is by using the Calculus of Communicating Systems (CCS) language. For this thesis, it is sufficient to introduce only a part of CCS from [AILS07] that contains visible actions, reduced operators and semantic rules, and leaves value passing out.

Definition 2.2 (CCS)**.** Firstly, let us define a set of action names Σ and a set of process names $\mathcal{K}$ (or constants). [AILS07] defines the set of CCS processes $\mathcal{P}$ using the following grammar:

$$P ::= K \ \mid \ \alpha.P \ \mid \ 0 \ \mid \ \sum_{i \in I} P_i$$

where

- $K \in \mathcal{K}$ is a name of a process,
- $\alpha \in \Sigma$ is an action,
- 0 represents a terminated process (has no transitions), and
- I is an index set.

The behavior of the process K is given by defining equation $K \stackrel{\text{def}}{=} P$.

Having the syntactic CCS definition, [AILS07] presents the CCS semantic for a process by a LTS $(\mathcal{P}, \Sigma, \rightarrow)$ with the help of the two inference rules presented below.

$$ACT \ \frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$SUM_k \ \frac{P_k \xrightarrow{\alpha} P'_k}{\sum_{i \in I} P_i \xrightarrow{\alpha} P'_k} k \in I$$

The transitions above the line are called premises and indicate that if they are true then the transitions under the line are also true. Therefore, one only has

to prove that the premises are true to show that the bottom transitions can be performed.

ACT rule has no premises, thus indicates that for a process $\alpha.P$, a transition $\alpha.P \xrightarrow{\alpha} P$ is possible without any further conditions [AILS07].

The SUM_k operator represents an alternative between k behaviors. If the SUM_k is finite, then can be denoted as $P_1 + ... + P_k$. A process $P_1 + P_2$ can show the behavior of P_1 or P_2. The decision is made at the moment when one of the processes makes the first transition. If P_1 makes the first transition then P_2 is discarded (analogously if P_2 performs first) [San11]. SUM_k expresses this behavior by showing that if there is a $P_k \xrightarrow{\alpha} P_k^{'}$ transition, then the whole SUM_k can perform a $SUM_k \xrightarrow{\alpha} P_k^{'}$ transition [AILS07].

From this point, I refer to LTS states as *states* and CCS processes as *processes*. This thesis mostly concentrates on states. However, because processes and states can both be used to describe the same behavior, every further definition that mentions states works analogously with processes.

***Example* 2.1.** State 1 from 2.1 can be described as $1 := yellow.yellow.0 + orange.blue.0$.

2.1.3 Hennessy-Milner Logic

Hennessy-Milner Logic, short HML, is a language used for specification and verification of computing systems[AILS07]. HML builds a foundation for distinguishing two states.

Definition 2.3 (HML)**.** [BN21] denotes the syntax of HML formulas, over a set of actions Σ, as HML[Σ] and defines it inductively as follows:

Observations If $\varphi \in$ HML[Σ] and $\alpha \in \Sigma$, then $\langle\alpha\rangle\varphi \in$ HML[Σ].
Conjunctions If $\varphi_i \in$ HML[Σ] for all i from a set of indexes I, then $\bigwedge_{i\in I} \varphi_i \in$ HML[Σ].
Negations If $\varphi \in$ HML[Σ], then $\neg\varphi \in$ HML[Σ].

An empty conjunction is denoted as T, so called nil-element.

***Example* 2.2.** A formula $\langle yellow\rangle\langle yellow\rangle\mathsf{T}$ is an example of an HML Formula.

2.2 Equivalences

Given two computing systems, their behavior can be described in form of a states or a process. The systems do not necessarily have to show the same behavior, however, even if they are not identical, there still might be some kind of similarity between them. These behavioral similarities are called *notions of behavioral equivalences*. Several notions of behavioral equivalences have been proposed for LTSs/CCS [AILS07]. Note that in this thesis the term notion of behavioral equivalence is reduced to just *equivalence*.

This section defines and explains five equivalences chosen from all equivalences included by Bisping and Nestmann in [BN21].

2.2.1 Trace Equivalence

[AILS07] defines traces as follows:

Definition 2.4 (Traces)**.** Given a state p, a finite sequence $\alpha_1 ... \alpha_k \in \Sigma^*$ with $k \geq 0$ is a trace of p if there exists a sequence of transition $p = p_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_k} p_k$ for some $p_1, \ldots, p_k$. Traces of the state p, written $T(p)$, is a collection of traces in the state p.

$T(p)$ contains all finite sequences of actions that one can observe while interacting with p.

[BFVG04] defines trace equivalence as follows:

Definition 2.5 (Trace equivalence)**.** For two states p and q one writes $p \sqsubseteq_T q$ if $T(p) \subseteq T(q)$. Two states p and q are considered trace equivalent, denoted $p \equiv_T q$, if $p \sqsubseteq_T q$ and $q \sqsubseteq_T p$.

2.2.2 Failure Equivalence

Failures are very similar to traces, but additionally allow one to observe a set of non-executable actions after a trace.

[San11] defines failures as follows:

Definition 2.6 (Failures)**.** A pair (s, Σ') is called a failure, where s is a finite sequence of observations and Σ' a set of actions. A state p has the failure (s, Σ') if:

- $p \xrightarrow{s} p'$
- $p' \xrightarrow{\alpha} \not\rightarrow$, for all $\alpha \in \Sigma'$

for some p'. Failures of the state p, written $F(p)$, is a collection of all failures that belong to the state p.

Definition 2.7 (Failure equivalence)**.** For two states p and q one writes $p \sqsubseteq_F q$ if $F(p) \subseteq F(q)$. Two states p and q are considered failure equivalent, denoted $p \equiv_F q$, if $p \sqsubseteq_F q$ and $q \sqsubseteq_F p$ [BFVG04].

2.2.3 Possible-future Equivalence

Possible-futures follow a similar idea as failures, however with a twist, that after a trace one can observe the set of all possible traces.

I introduce possible-futures as in [Gla90].

Definition 2.8 (Possible-futures)**.** A pair (s, X) is called a possible-future, where s is a finite sequence of observations and X a set of traces. A state p has the possible-future (s, Σ) if:

- $p \xrightarrow{s} p'$
- $X = T(p')$

for some p'. Possible-futures of the state p, written $PF(p)$, collect all possible-futures that belong to the state p.

[BFVG04] introduces possible-future equivalence as follows:

Definition 2.9 (Possible-future equivalence)**.** For two states p and q one writes $p \sqsubseteq_{PF} q$ if $PF(p) \subseteq PF(q)$. Two states p and q are considered trace equivalent, denoted $p \equiv_{PF} q$, if $p \sqsubseteq_{PF} q$ and $q \sqsubseteq_{PF} p$.

2.2.4 Simulation Equivalence

A simulation allows one to determine whether one state can "simulate" a behavior of another state. Whenever there is the state p that "simulates" state q, then q can show at least the same behavior as p.

[San11] defines simulation equivalence as follows:

Definition 2.10 (Simulation equivalence)**.** A relation $R \subseteq S \times S$ is a simulation if, whenever $(p, q) \in R$ then:

- for all states p' and $\alpha \in \Sigma$ with $p \xrightarrow{\alpha} p'$, there is q' such that $q \xrightarrow{\alpha} q'$ and $(p', q') \in R$.

The preorder $\sqsubseteq_S$ relates two states p and q if there is a simulation R with $(p, q) \in R$. Equivalence denoted as $\equiv_S$, induced by $\sqsubseteq_S$, is called a simulation equivalence. For p and q applies $p \equiv_S q$ if $p \sqsubseteq_S q$ and $q \sqsubseteq_S p$.

***Example* 2.3.** For two states 1 and 5 from example 2.1 one can observe:

- that $1 \xrightarrow{yellow} 2$ and there is $5 \xrightarrow{yellow} 7$,
- however $2 \xrightarrow{yellow} 4$ and $7 \not\xrightarrow{yellow}$.

Therefore $(2, 7) \notin R$ and consequently $(1, 5) \notin R$, so the states 2 and 7 are not simulation equivalent.

2.2.5 Bisimulation Equivalence

Bisimulation equivalence is the finest equivalence introduced in [BN21] and thus implies all other equivalences introduced in this thesis. Furthermore, two computing systems that are bisimulation equivalent are considered to have the same behavior [San11].

I introduce formal definition as in [San11].

Definition 2.11 (Bisimulation equivalence)**.** A relation $R \subseteq S \times S$ is a bisimulation if, whenever $(p, q) \in R$, for all $\alpha \in \Sigma$ the following applies:

- For all states p' and $\alpha \in \Sigma$ with $p \xrightarrow{\alpha} p'$, there is q' such that $q \xrightarrow{\alpha} q'$ and $(p', q') \in R$.
- For all states q' and $\alpha \in \Sigma$ with $q \xrightarrow{\alpha} q'$, there is p' such that $p \xrightarrow{\alpha} p'$ and $(p', q') \in R$.

Both bisimulation preorder $\sqsubseteq_B$ and bisimulation equivalence $\equiv_B$ are defined analogously to definitions of simulation preorder and simulation equivalence in the subsection 2.2.4.

More intuitive explanation: a pair of states is considered bisimulation equivalent if all the state pairs that one reaches while making transitions, also show equivalent behavior.

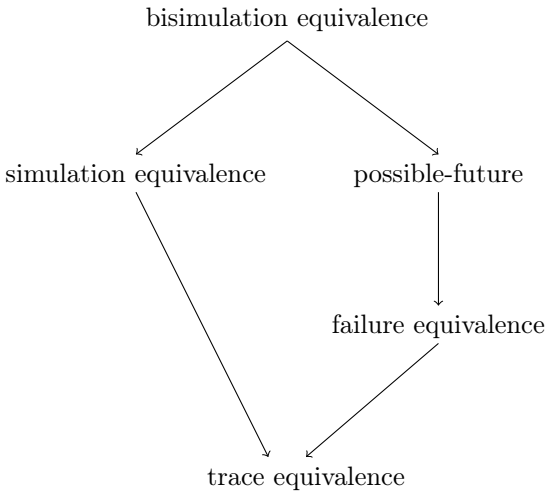
***Example* 2.4.** For the same states as in example 2.3 the following applies.

- $5 \xrightarrow{yellow} 7$ and $1 \xrightarrow{yellow} 2$.
- However, $2 \xrightarrow{yellow} 4$, but $7 \not\xrightarrow{yellow}$.

Therefore $(2, 7) \notin R$ and consequently $(5, 1) \notin R$, so $5 \not\sqsubseteq_B 7$ and thus $5 \not\equiv_B 7$.

Let X_1 be an equivalence, if two states are not X_1-equivalent then they are called X_1-inequivalent. When an equivalence X_1 is called *finer* than an equivalence X_2 (X_2 and X_1 are different equivalences), then it means that X_1-equivalence between two states implies X_2-equivalence between the same states. More intuitively, all pairs of states that are X_1 equivalent are also X_2 equivalent and if a pair of states is X_2-inequivalent then the pair is also X_1-inequivalent. Below is a figure 2.2 from [Gla90] provided to illustrate the dependencies between different equivalences. If two equivalences are connected (also indirectly), then the one above is finer than the one below. For example, if two states are possible-future equivalent, then they are also failure and trace equivalent. On the other hand, if the states are simulation equivalent, then they are not automatically failure equivalent [Gla90].

Figure 2.2: Linear-time−branching-time spectrum



2.3 HML Semantics

In subsection 2.1.3 I have already introduced the HML syntax. Now i would like to introduce HML semantics in order to establish a foundation for distinguishing two states. This section firstly presents HML semantics and then concludes by defining distinguishing formulas.

Definition 2.12 (HML semantics)**.** Given an LTS $L = (S, \Sigma, \rightarrow)$ and a state $p \in S$, [FvGd06] defines the semantics of HML as follows:

- $p \vDash \bigwedge_{i \in I} \varphi_i$ if $p \vDash \varphi_i$ for all $i \in I$,
- $p \vDash \langle \alpha \rangle \varphi$ if $p \xrightarrow{\alpha} q$ and $q \vDash \varphi$ for some $q \in S$, and
- $p \vDash \neg\varphi$ if $p \nvDash \varphi$.

If $p \vDash \varphi$, then we say that state p satisfies the formula φ.

***Example* 2.5.** Given the two states 1 and 5 from figure 2.1, one can observe that
$1 \vDash \langle yellow \rangle \langle yellow \rangle \mathsf{T}$ but $5 \nvDash \langle yellow \rangle \langle yellow \rangle \mathsf{T}$.

Given both definitions of HML syntax and HML semantics, it is possible to distinguish two (bisimulation inequivalent) states with the help of a formula that one state satisfies but the other one does not.

Definition 2.13 (Distinguishing formula)**.** Given an LTS $L = (S, \Sigma, \rightarrow)$, a formula φ distinguishes state $p \in S$ from $q \in S$ if $p \vDash \varphi$ but $q \nvDash \varphi$ [BN21].

Processes p and q do not necessarily need to belong to the same LTS.

If for all $\varphi \in \mathrm{HML}[\Sigma]$ applies, that if $p \vDash \varphi$ then $q \vDash \varphi$ and if $q \vDash \varphi$ then $p \vDash \varphi$, then p and q are bisimulation equivalent [FvGd06]. This means that there is no HML formula that distinguishes p from q or q from p.

***Example* 2.6.** Continuing the example 2.5, the formula $\langle yellow \rangle \langle yellow \rangle \mathsf{T}$ distinguishes state 1 from 5, but not state 5 from 1.

2.4 Linear-time−Branching-time Spectrum

If two states are bisimulation inequivalent, one can develop a formula to distinguish them. However, as already mentioned in section 2.2, even if two states are bisimulation inequivalent, there still might be some kind of other equivalency between them. Assume that two states are X-equivalent, but not bisimulation equivalent. Then there is a distinguishing formula for these two states. However, since the two states are X equivalent, then there should exist a set of reduced HML formulas for each equivalence so that no formula from that set distinguishes the two states. [BN21] introduces a spectrum of observation languages that accomplishes that issue. This section presents definition of observational preorders followed by the definition of observation languages.

[BN21] defines observational preorders as follows:

Definition 2.14 (Observational preorders)**.** Given an LTS $L = (S, \Sigma, \rightarrow)$ and two states $q, p \in S$, a set of observations, written $\mathcal{O}_X \subseteq \text{HML}[\Sigma]$, preorders these two states, $p \preceq_X q$, if there is no formula $\varphi \in \mathcal{O}_X$ that distinguishes p from q. If $p \preceq_X q$ and $q \preceq_X p$, then $p \equiv_X q$ (see section 2.2). If a formula $\varphi \in \mathcal{O}_X$ distinguishes state p from state q, then $p \not\equiv_X q$.

As already mentioned, each equivalence presented in section 2.2 has its own observation language. The linear-time−branching-time spectrum, as presented in [BN21], introduces a lattice of observation languages.

Definition 2.15 (Observation languages)**.** Every observation language $\mathcal{O}_X$ performs trace observations, that is:

- $\mathsf{T} \in \mathcal{O}_X$ and
- if $\varphi \in \mathcal{O}_X$, then $\langle\alpha\rangle\varphi \in \mathcal{O}_X$.

In the spectrum there are:

- trace observations $\mathcal{O}_T$: just trace observations,
- failure observations $\mathcal{O}_F$: $\bigwedge_{i\in I} \neg\langle\alpha_i\rangle \in \mathcal{O}_F$,
- possible-futures $\mathcal{O}_{PF}$: $\bigwedge_{i\in I} \varphi_i \in \mathcal{O}_{PF}$ with all $\varphi_i \in \{\neg\psi_i, \psi_i\}$ and $\psi_i \in \mathcal{O}_T$,
- simulation observations $\mathcal{O}_S$: if $\varphi_i \in \mathcal{O}_S$ for all $i \in I$, then $\bigwedge_{i\in I} \varphi_i \in \mathcal{O}_S$, and
- bisimulation observations $\mathcal{O}_B$: $\varphi \in \mathcal{O}_B$ for all $\varphi \in \text{HML}[\Sigma]$.

Each observation language of the spectrum allows different amounts of the syntactic features of HML. [BN21] notes the five observation languages are closed and inductive. Closed in the sense that for a formula $\varphi \in \mathcal{O}_X$, every subformula φ' of φ is also a part of the language $\mathcal{O}_X$. Inductive in the sense that a formula of an observation language $\mathcal{O}_X$ can be built using only formulas with lower syntax tree height, that belong to the same observation language $\mathcal{O}_X$.

***Example* 2.7.** Continuing example 2.5, because the formula $\langle yellow\rangle\langle yellow\rangle\mathsf{T}$ is a part of $\mathcal{O}_T$, the states 1 and 5 are not trace equivalent.

2.4.1 Formula Pricing System

Because of the fact, that each observation language allows a different amount of syntactic features of HML, one can use these features to rate formulas and assign them to observation languages. [BN21] presents a pricing system based on the number of conjunctions, positive deep branches, positive flat branches, negations, negation height, and observations that a formula yields. Below an explanation of each aspect according to [BN21].

- **Conjunctions:** How many conjunctions one runs into when descending down the syntax tree of a formula? Additionally, negations at the exact beginning of a formula or following an observation count as conjunctions.

- **Positive deep branches:** How many positive deep branches appear in every conjunction? Flat branches are subformulas of the form $\langle\alpha\rangle$ or $\neg\langle\alpha\rangle$, all other subformulas are called deep branches.
- **Positive flat branches:** How many positive flat branches appear in every conjunction?
- **Negations:** How many negations one runs into when descending down the syntax tree of a formula?
- **Negation height:** How high can the syntax tree under every negation be?
- **Observations:** How many observations one runs into when descending down the syntax tree of a formula?

With this pricing system, it is possible to compare two formulas and select a "cheaper" one. If the price of a formula φ_1 has lower or equal values in each dimension and at least one entry lower than the price of a formula φ_2, then φ_1 dominates φ_2 (φ_1 is cheaper than φ_2) [BN21]. One can also compare the price of a formula with the boundaries of the observation languages in order to easily define to which observation language does the formula belong. Table 2.1 presents these boundaries showing how many syntactic HML-features each observation language can use at most.

Table 2.1: Dimensions of observation expressiveness [BN21]

Observation language	*Conjunctions*	*Positive deep br.*	*Positive flat br.*	*Negations*	*Negation height*	*Observations*
trace $\mathcal{O}_T$	0	0	0	0	0	∞
failure $\mathcal{O}_F$	1	0	0	1	1	∞
possible-future $\mathcal{O}_{PF}$	1	∞	∞	1	∞	∞
simulation $\mathcal{O}_S$	∞	∞	∞	0	∞	∞
bisimulation $\mathcal{O}_B$	∞	∞	∞	∞	∞	∞

As you can see in table 2.1 $\mathcal{O}_T$ and $\mathcal{O}_F$ both allow neither positive deep branches nor positive flat branches. For remaining observation languages both dimensions (positive deep br. and positive flat br.) are unlimited. Therefore there is no need to distinguish between deep or flat branches in order to verify whether a formula is a part of one of the observation language. Thus, I can reduce the dimensionality of the pricing system by merging positive deep branches and positive flat branches dimensions into one positive branches dimension. This results in table 2.2.

Table 2.2: Reduced dimensions of observation expressiveness

Observation language	*Conjunctions*	*Positive br.*	*Negations*	*Negation height*	*Observations*
trace $\mathcal{O}_T$	0	0	0	0	∞
failure $\mathcal{O}_F$	1	0	1	1	∞
possible-future $\mathcal{O}_{PF}$	1	∞	1	∞	∞
simulation $\mathcal{O}_S$	∞	∞	0	∞	∞
bisimulation $\mathcal{O}_B$	∞	∞	∞	∞	∞

If a formula does not exceed the limits of an observation language, then the formula *fits* into the language.

***Example* 2.8.** Continuing example 2.7, the formula $\langle yellow \rangle \langle yellow \rangle \mathsf{T}$ has the price $(0, 0, 0, 0, 2)$ and therefore fits into the $\mathcal{O}_T$ language, thus all pairs of states that this formula distinguishes are not trace equivalent.

Chapter 3

The Game Theory

This chapter firstly introduces the reader to the spectroscopy game and associated terms like plays and wins, strategies, winning regions, and winning strategy graphs. The second section explains how to create HML Formulas from the spectroscopy game.

3.1 The Spectroscopy Game

The idea of combining logic with game theory is being in development for over the past sixty years, with its foundation in Gale-Stewart-style games [Ber08]. The fundamental rules are the following: two players, attacker and defender, compete against each other in a perfect information game. The game has exactly one winner and one loser. Let us introduce the spectroscopy game and few related definitions.

The Spectroscopy game from [BN21] makes the attacker choose a strategy of attacking. The defender plays non-deterministically by moving to multiple states at once. The attacker may challenge the defender by making him pick exactly one state, of all the states where the defender is currently in, from which the defender then continues playing.

Definition 3.1. Given an LTS $L = (S, \Sigma, \rightarrow)$, the M-labeled spectroscopy game, according to [BN21], is defined as $\mathcal{G}^S_\Delta[g_0] = (G, G_d, \rightarrowtail, g_0)$ with $M = \{\neg, \wedge, *, \langle\alpha\rangle\}$, and consists of:

- attacker positions $(p, Q)_a \in G_a$ with $p \in \mathcal{P}, Q \in 2^{\mathcal{P}}$, $G_a = G \setminus G_d$ and
- defender positions $(p, Q)_d \in G_d$,

and four kinds of moves:

- observation moves $(p, Q)_a \overset{\langle\alpha\rangle}{\rightarrowtail} (p', \{q' \mid \exists q \in Q. q \xrightarrow{\alpha} q'\})_a$ if $p \xrightarrow{\alpha} p'$,
- conjunct challenges $(p, Q)_a \overset{\wedge}{\rightarrowtail} (p, Q)_d$,
- conjunct answers $(p, Q)_d \overset{*}{\rightarrowtail} (p, \{q\})_a$ if $q \in Q$, and
- negation moves $(p, \{q\})_a \overset{\neg}{\rightarrowtail} (q, \{p\})_a$.

The negation move is only possible when $|Q| = 1$. Bisping and Nestmann [BN21] have proven that the spectroscopy game $\mathcal{G}_\Delta[(p, \{q\})_a]$ is won by the defender precisely if p and q are bisimulation equivalent.

All further definitions in this section are from[BN21].

Definition 3.2 (Plays and wins)**.** Given a game $\mathcal{G}_\Delta[g_0]$, the path $g_0 g_1 ... \in G^\infty$ with $g_i \rightarrowtail g_{i+1}$ is called a play of $\mathcal{G}_\Delta[g_0]$. If a play is infinite, then the defender wins. If a finite play $g_0 ... g_n \not\rightarrowtail$ is stuck, then the player that cannot perform further moves (is stuck) loses. In other words, defender wins if $g_n \in G_a$, and attacker wins if $g_n \in G_d$.

Definition 3.3 (Strategies and winning strategies)**.** A set of moves $F \subseteq \rightarrowtail$ is called a (positional, non-deterministic) strategy. When a player follows a strategy F (fairly picks elements of F) and wins the game, then F is called a winning strategy for that player. The player that has a winning strategy for a game $\mathcal{G}_\Delta[g_0]$ wins $\mathcal{G}_\Delta[g_0]$.

Definition 3.4 (Winning regions)**.** The set $W_a \subseteq G$ of all positions g where the attacker wins $\mathcal{G}_\Delta[g]$ (there is a winning strategy for the attacker) is called the attacker winning region (analogously for defender winning region W_d).

As stated in [BN21], all spectroscopy games are determined. This means that every position $g \in G$ is included in either W_a or W_d, or one could also say $W_a \cup W_d = G$.

Definition 3.5 (Winning strategy graph)**.** From an attacker winning region and an initial position $g_0 \in W_a$, one can build the attacker winning graph F_a, which is a subset of the $\rightarrowtail$-graph, by following all $\rightarrowtail$-edges that do not lead out of W_a.

3.2 From the Spectroscopy Game to Cheapest Distinguishing Formulas

The spectroscopy game allows us to collect all distinguishing formulas for two states. One can compare the formulas with observation languages to retrieve the coarsest equivalence for the two states. In order to achieve this aim, the last section of chapter 3 first explains how to build distinguishing formulas from attacker strategies and concludes by defining a way to compare multiple distinguishing formulas for the purpose of receiving the cheapest one.

3.2.1 Distinguishing Formulas

Given a spectroscopy game, the attacker can develop (if possible) multiple different strategies on how to win the game. However, strategies are sets of game positions, not formulas. Therefore [BN21] presents a way of constructing distinguishing formulas out of the attacker strategies.

Definition 3.6 (Strategy formulas)**.** Given a spectroscopy game $\mathcal{G}_\Delta$ and an attacker strategy $F \subseteq (G_a \times M \times G)$ for the game, [BN21] inductively defines the set of strategy formulas, written $\mathbf{Strat}_F(g_a)$, as follows:

- If $\varphi \in \mathbf{Strat}_F(g'_a)$ and $(g_a, \langle\beta\rangle, g'_a) \in F$, then $\langle\beta\rangle\varphi \in \mathbf{Strat}_F(g_a)$,

- if $\varphi \in \mathbf{Strat}_F(g'_a)$ and $(g_a, \neg, g'_a) \in F$, then $\neg\varphi \in \mathbf{Strat}_F(g_a)$, and
- if $\varphi_{g'_a} \in \mathbf{Strat}_F(g'_a)$ for all $g'_a \in I = \{g'_a \mid g_d \overset{*}{\rightarrowtail}_\Delta g'_a\}$, and $(g_a, \wedge, g_d) \in F$, then $\bigwedge_{g'_a \in I} \varphi_{g'_a} \in \mathbf{Strat}_F(g_a)$.

It is important to understand the intuition behind the last part of the definition above. If there is a conjunct move, which is a part of the attacker strategy, and there are multiple conjunct answers possible, then one joins strategy formulas of every conjunct answer into one conjunction. In the subsection 4.3.5 I will come back to this and explain why is it important and what impacts does it have on my implementation of the spectroscopy game.

***Example* 3.1.** Let me introduce a new state $p_1 := a.0$ and a set $Q_1 := \{(a.0 + b.0), (a.0 + c.0)\}$. And let us assume that the attacker performs first the conjunct challenge move $(p_1, Q_1)_a \overset{\wedge}{\rightarrowtail} (p_1, Q_1)_d$ and then for every conjunct answer possible beats the defender with the following two sequences of game moves $(p_1, \{a.0 + b.0\})_a \overset{\neg}{\rightarrowtail}\overset{b}{\rightarrowtail}\overset{\wedge}{\rightarrowtail}$ and $(p_1, \{a.0 + c.0\})_a \overset{\neg}{\rightarrowtail}\overset{c}{\rightarrowtail}\overset{\wedge}{\rightarrowtail}$. The resulting attacker winning strategy gives rise to following strategy formula $\bigwedge\{\langle b\rangle\mathsf{T}, \langle c\rangle\mathsf{T}\}$.

Bisping and Nestmann proved the following lemma regarding the attacker winning regions and distinguishing formulas.

Lemma 1. Given a spectroscopy game $\mathcal{G}_\Delta$ and the attacker winning region W_a of the game, every $\varphi \in \mathbf{Strat}_{F_a}((p, \{q\})_a)$ distinguishes p from q.

3.2.2 Cheapest Distinguishing Formulas

The attacker in the spectroscopy game can develop multiple different winning strategies. Different strategies give birth to different distinguishing formulas. [BN21] mentions, that because of potential cycles in the attacker winning strategy graph F_a, $\mathbf{Strat}_{F_a}$ will most likely include infinitely many formulas. Therefore Bisping and Nestmann only focus on the smallest formulas and the ones that belong to the smallest observation language from the spectrum. The following algorithm, created by Bisping and Nestmann, computes a set of cheapest distinguishing formulas for two states.

Algorithm 1 Spectroscopy Procedure [BN21]

procedure GAME_SCPETROSCOPY(S, p_0, q_0):
  $\mathcal{G}^S_\Delta = (G, G_a, \rightarrowtail) \leftarrow$ construct_spectroscopy_game(S)
  $W_a \leftarrow$ compute_winning_region($\mathcal{G}^S_\Delta$)
  if $(p_0, \{q_0\})_a \in W_a$ **then**
    $F_a \leftarrow$ winning_graph($\mathcal{G}^S_\Delta, W_a, (p_0, \{q_0\})_a$)
    strats[]$\leftarrow \emptyset$
    todo $\leftarrow [(p_0, \{q_0\})_a)]$
    while todo $\neq$ [] **do**
      g $\leftarrow$ todo.dequeue()
      sg $\leftarrow$ strats[g]
      if sg $=$ *undefined* **then**
        strats[g] $\leftarrow \emptyset$
      gg$' \leftarrow \{g' | (\mathrm{g}, \bullet, g') \in F_a \wedge \mathrm{strats}[g'] = \textit{undefined}\}$
      if gg$' = \emptyset$ **then**
        sg$' \leftarrow$ prune_dominated(Strat$'_{F_a,\mathrm{strat}}$(g))
        if sg $\neq$ sg$'$ **then**
          strats[g] $\leftarrow$ sg$'$
          todo.enqueue_each_end($\{g^* | (g^*, \bullet, \mathrm{g}) \in F_a \wedge g^* \notin \mathrm{todo}\}$)
      else
        todo.enqueue_each_front(gg$'$)
    return strats($(p_0, \{q_0\})_a$)
  else
    $R \leftarrow \{(p, q) | (p, \{q\})_a \in G_a \setminus W_a\}$
    return R

For two states p_0 and q_0 the algorithm constructs the spectroscopy game $\mathcal{G}_\Delta$ starting with the initial game position $(p_0, \{q_0\})_a$ and then computes the attacker winning region W_a. If $(p_0, \{q_0\})_a \notin W_a$, then it means that the two states are bisimulation equivalent and the algorithm terminates returning bisimulation relation. If however $(p_0, \{q_0\})_a \in W_a$, then the algorithm computes the attacker winning strategy graph F_a. Using F_a, the pricing system for formulas, and the strategy formulas definition, the procedure computes the set of cheapest distinguishing formulas that distinguish p_0 from q_0 [BN21].

Chapter 4

The Spectroscopy Invaders

As a practical background for my thesis, I have implemented the spectroscopy game as an educational web browser game called "The Spectroscopy Invaders". This chapter presents the game design and explains the most significant choices I have made during the implementation process. "The Spectroscopy Invaders" game helps a player to understand the idea behind the spectroscopy game.

To give a suitable overview of "The Spectroscopy Invaders" game, this chapter first explains the implementation process of the spectroscopy procedure. The second section describes techniques of good game design. The third section presents the video game design and compares it to other similar video games. The last section deals with the problem of converting attacker's choices into a formula.

4.1 The Spectroscopy Procedure

In "The Spectroscopy Invaders", I have implemented the spectroscopy procedure due to its ability to compute the cheapest distinguishing formulas. However, one can keep some of the computations that the algorithm has made and use them for further purposes. Therefore, in this section, I will first describe how I have implemented the spectroscopy procedure and then explain for what further purposes I have used it.

4.1.1 Implementation

To implement the spectroscopy procedure I have followed the steps in the algorithm 1. To create the spectroscopy game $\mathcal{G}_\Delta$ I calculate possible sequences of game moves. To prevent unnecessary loops while creating sequences of game moves, following scenarios are not taken into account:

- Two or more negation moves in a row.
- Conjunct challenge for game positions $(p, Q)_a$ where $|Q| = 1$.
- Visiting the same game position more than once in a single sequence.

The sequences of game moves are saved in form of a tree. The nodes are represented by game positions, the root node is the initial game position and a

path from the root to any node is a sequence of game moves. I will refer to this tree as the *spectroscopy tree*. Every node that is a part of the attacker winning region is marked as winning.

If the states are bisimulation equivalent, then instead of returning the bisimulation relation, my implementation of the spectroscopy procedure will simply return an empty list.

4.1.2 Further Purposes

After the spectroscopy procedure terminates the spectroscopy tree is kept. For any game position, one can traverse the tree in order to find the node representing a given game position and verify whether it is part of the attacker winning region or not. This feature will be implemented as a feedback mechanics, but I will come back to it in subsection 4.3.8.

4.2 Efficient Learning from Video Games

The goal was to implement an educational game that combines learning with the design and mechanics of computer games. Murphy presents techniques of game design [Mur12] to guide a game developer on how to ensure that a player learns new skills and receives an exciting game experience at the same time.

4.2.1 Techniques of Game Design

[Mur12] introduces seven abstract techniques of game design. The techniques tell a game designer what is needed to ensure that players efficiently learn all the skills that the game has to offer. For each technique, I will introduce its concept as in [Mur12].

The first technique is *flow*. Flow is often said to be the fundamental reason why people play games and is described as an "optimal human experience" or "engagement". Game designers have the task to keep players in flow for as long as possible. To ensure flow in a game, the designer has to fulfill four requirements, that are:

- *Clear tasks*: Players can understand the assignments.
- *Feedback*: The game gives feedback about own choices and the progression towards goals.
- *Balanced, attainable goal*: The tasks have to be challenging but not too hard.
- *Concentration*: The game should not distract the player from finishing the tasks.

Furthermore, the difficulty should increase along with time and the skills that the players have already learned.

The second technique is *feedback*, which is also a part of flow. Players should receive feedback that guides them to achieve goals by informing them about the progress and the quality of their actions.

Simplicity requires the game designer to make the user interface, the goals/tasks, feedback, game mechanics, screen layout, and the rules and instructions "as simple as possible, but not simpler"[Mur12]. Simplicity makes the game less complex, making the goals/tasks easier to understand and allowing players to correlate their choices and actions to the feedback.

Immersion and engagement technique makes use of visual and audio methods like graphics or music to interest the players in the game's story (called immersion) and introduces riddles/puzzles engaging players to solve them.

Choice/involvement provides players with "meaningful" choices during the game. However, one has to balance between not enough and too many choices, because players facing too many options have trouble comparing them and making the right decision.

Practice technique introduces the use of practice and repetition as a part of the game. The game designer creates sequences of challenges with increasing difficulty and variety that players have to overcome. Players learn new skills until they learn everything the game has to offer.

The last technique is called *fun*. [Mur12] defines fun as "the positive feelings that occur before, during, and after a compelling flow experience". This technique states, that the game should arouse as many as possible of the following feelings: delight, engagement, enjoyment, cheer, pleasure, entertainment, satisfaction, happiness, control, and mastery of the material.

4.3 Game's Design

This section describes the design of "The Spectroscopy Invaders" game, explains how all seven techniques of game design were implemented and compares "The Spectroscopy Invaders" to other similar video games.

4.3.1 Similar Video Games

During further sections, "The Spectroscopy Invaders" will be repetitively compared with two other similar video games that are also based on the concept of equivalence games. The first game has been implemented by Dominik Andre Peacock as a part of his bachelor's thesis titled "Process equivalences as a video game" [Pea20]. The second game, named "One Step Too Far" [EKH+], has been created by 8 students of the Technical University of Berlin as a part of the programming project "Models of Dynamic Systems".

Figure 4.1: Level 1 in Peacock's game

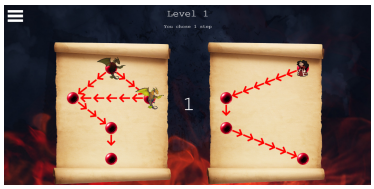


Figure 4.2: Level 1 in "One Step Too Far"

4.3.2 Game World

A player entering "The Spectroscopy Invaders" finds him/herself in a fantasy world and plays as a groom that has to defeat evil elves. The main menu consists of five segments, each includes different graphics and implements four levels. The plot and colorful graphics are there to keep the player immersed, and arouse positive emotions like enjoyment and pleasure whereby the techniques of immersion and engagement, and fun are partly realized. However, I have limited the amount of graphics on the screen and the complexity of the plot to a minimum. As a result the player is not distracted and therefore I have accomplished simplicity and concentration requirements in flow.

The game developed by Peacock [Pea] also welcomes the player with main menu that includes graphics and allows him/her to enter various levels divided into 5 segments. The attacker in Peacock's game is represented by a blue robot, and the defender by a red robot. The second game "One Step Too Far" firstly introduces the player to the plot and explains the basic mechanics of the game. Then the player can choose one of the 20 levels divided into four segments. The attacker there is represented by a dragon figure and the defender by a princess.

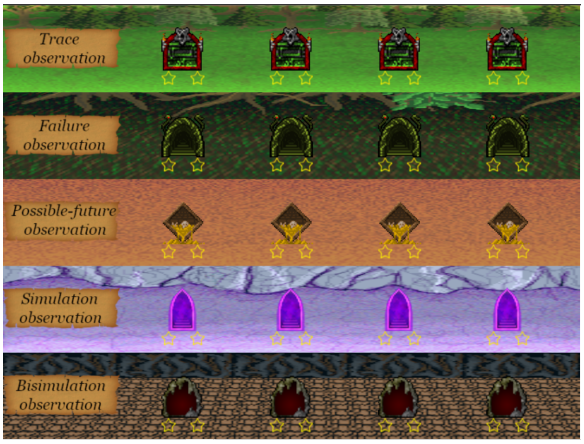


Figure 4.3: Main menu in "The Spectroscopy Invaders"

4.3.3 Levels and Their Complexity

In the theory, the spectroscopy game is a combination of game positions. Each game position is a configuration of states belonging to an LTS. The game starts with an initial position $(p_0, \{q_0\})_a$, where p_0 and q_0 are two different states.

In "The Spectroscopy Invaders", clicking on a level in the main menu opens a battleground where the player competes in the implementation of the spectroscopy game on two graphs. Both graphs consist of blue hexagons, referred to as *fields*, and represent an LTS with fields being the states. A field with no outgoing arrows is called a *closed field*. The attacker is represented by a groom figure, called *hero*, and the defender by elves figures. The figures can stand on the fields. The arrows show to which fields a figure can move. These arrows represent possible transitions in an LTS where different colors stand for different actions. Because the defender can be simultaneously in multiple fields, there

can be multiple elf figures at the time. The elves are called *enemies*. I refer to the fields where the hero is placed as the *hero's field* and to the fields where the enemies are as *enemies' fields*.

Following the technique of simplicity, I have decided to reduce the complexity of each level. Each field can have at most 3 outgoing arrows and graphs created by fields in each level do not include cycles. Furthermore, any two fields on the game board are connected with no more than one arrow. There are short instructions provided for each segment. Thus, I have created small and simple levels for the player and therefore improved the simplicity.

Playgrounds in "One Step Too Far" and in Peacock's game also consist of two separate graphs. "One Step Too Far" also uses colorful arrows while Peacock's arrows, instead of having different colors, have different shapes.

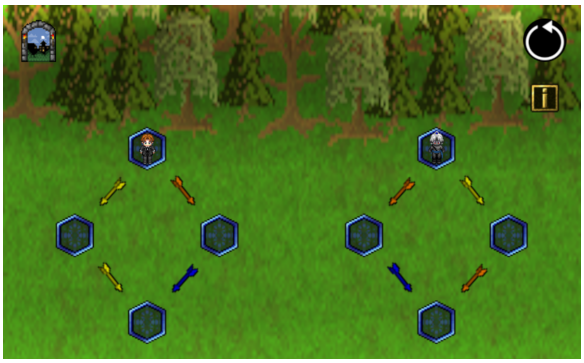


Figure 4.4: Level 1 in "The Spectroscopy Invaders" (represents the LTS from figure 2.1)

4.3.4 Player as Attacker

In the spectroscopy game, the attacker and the defender make moves according to the rules introduced in def. 3.1. However, the attacker gets to perform most moves, while the defender can only answer to the conjunct challenge. Besides that, the attacker's moves are responsible for building distinguishing formulas according to def. 3.6. Therefore, following the techniques of choice/involvement, the player should make meaningful decisions. Thus, the player takes over the role of the attacker. In contrast to the theoretical spectroscopy game, where the defender performs the conjunct answer move, "The Spectroscopy Invaders" does not provide game activities for the defender.

In Peacock's game, the player takes over the role of the attacker. In "One Step Too Far" the player can experience the game from a very different perspective by playing as the defender. However, before a level starts, the player has to inspect the graphs and give a number of attacker moves to mimic.

4.3.5 Game Activities

The spectroscopy game from [BN21] presents four moves. In this thesis, I refer to the implementations of these moves as *game activities*.

Observation Activity

Following the def. 3.1, Bisping and Nestmann introduce observation move as following. Given an attacker position $(p, Q)_a$, for every possible transition $p \xrightarrow{\alpha} p'$ there exist an observation move $(p, Q)_a \xrightarrow{\alpha} (p', Q')_a$ where Q' contains all states that are possible to reach from any state in Q performing one transition that executes the action α.

In "The Spectroscopy Invaders", observation activity can be performed by clicking on any field that has an ingoing arrow pointing from the hero's field. After the execution of the observation activity, the hero moves to the chosen state. Every enemy, whose field has outgoing arrows with the same color as the arrow along which the hero has moved, moves to the fields to which the arrows are pointing. If an enemy has not moved, he is defeated. To indicate that an enemy has been defeated, his figure starts shaking and flashing for a short time and then disappears. It might happen that one enemy has to move to more than one field. In that case, the enemy is split into multiple enemies and each moves to one of the fields. It is also possible that two or more enemies move simultaneously to the same field. In that case, the enemies merge into one single enemy.

Peacock provides a similar solution to my observation activity. However, in his game, the red robot can be in only one place at a time. Therefore, if there is more than one possible field to which the red robot can move, as an answer to the blue's robot movement, the red robot has to choose exactly one using a decision algorithm. Spectroscopy game allows the defender to move simultaneously to multiple fields, therefore I do not have to implement a decision algorithm for observation activity. In "One Step Too Far" there is an algorithm that selects where to move the dragon next. Then, the princess has to answer by choosing an equivalent transition from the field where she is currently placed at.

Figure 4.5: Before the execution of the observation activity

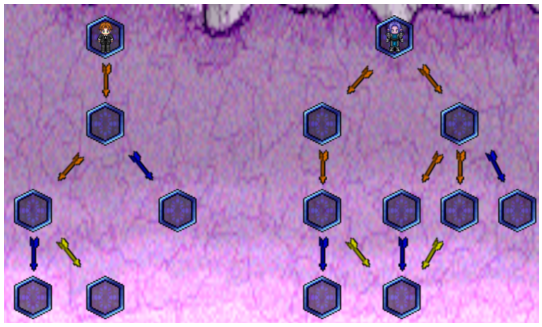
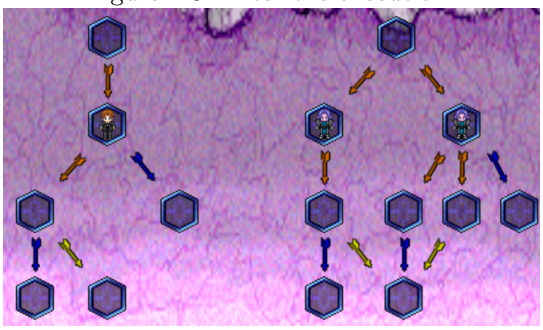


Figure 4.6: After the execution



Negation Activity

The second move in the spectroscopy game is the negation move. Given an attacker position $(p, \{q\})_a$, there exists a negation move $(p, \{q\})_a \overset{\langle\neg\rangle}{\rightarrowtail} (q, \{p\})_a$, when Q consists of exactly one state q. As a result, the states are swapped.

In "The Spectroscopy Invaders", there exists a button between the two graphs that allows the player to perform negation activity. The button is active only if there is exactly one enemy on the board and when pressed, the hero and the only enemy swap their positions (fields).

[Pea] also implements a swapping mechanism. The button between the two graphs makes the blue robot and the red robot swap their positions. Since it is the attacker that performs the negation action, there is no button in "One Step Too Far" to initiate the swapping process. Instead, the algorithm that chooses where to move the dragon next has to consider swapping positions with the princess as a possibility for the next move.

Figure 4.7: Before the execution of the negation activity

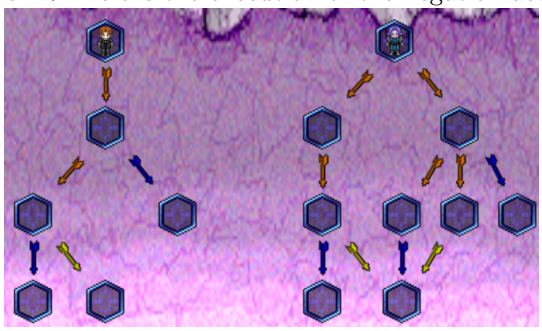
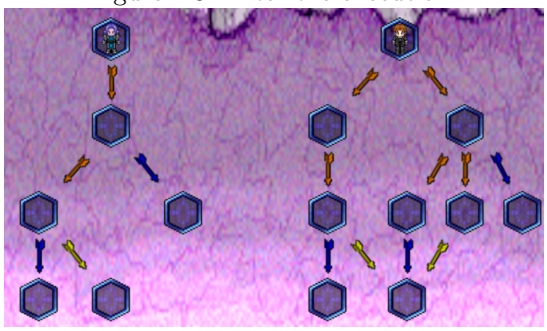


Figure 4.8: After the execution



Conjunct Activity

The last two moves in the spectroscopy game are conjunct challenge and conjunct answer. For any attacker position $(p, Q)_a$, conjunct challenge results in defender position $(p, Q)_d$ where the defender answers $(p, Q)_d \rightarrowtail^{*} (p, \{q\})_a$ picking exactly one state $q \in Q$ and the rest $Q \setminus q$ states are discarded. The spectroscopy game continues from the attacker position $(p, \{q\})_a$.

Back to example 3.1. Let us assume that the attacker first plays conjunct challenge $(a.0, \{(a.0+b.0), (a.0+c.0)\})_a \overset{\wedge}{\rightarrowtail} (a.0, \{(a.0+b.0), (a.0+c.0)\})_d$. The defender can answer with two conjunct answers:

- $(a.0, \{(a.0 + b.0), (a.0 + c.0)\})_d \rightarrowtail^{*} (a.0, \{a.0 + b.0\})_a$ or
- $(a.0, \{(a.0 + b.0), (a.0 + c.0)\})_d \rightarrowtail^{*} (a.0, \{a.0 + c.0\})_a$.

For simplicity, let us define $s_1 = (a.0, \{a.0 + b.0\})_a$ and $s_2 = (a.0, \{a.0 + c.0\})_a$.

The reason why I have mentioned both moves together is because "The Spectroscopy Invaders" does not provide any defender algorithm that selects "the best" $q \in Q$, therefore, "The Spectroscopy Invaders" combines both moves forcing the player to play $(p, \{q\})_a$ for each $q \in Q$. The explanation behind it is the following. In the spectroscopy procedure, the algorithm simulates the spectroscopy game for all conjunct answers and saves every attacker move in F. Then, the procedure calculates strategy formulas $Strat_F(s_1) = \neg\langle b\rangle$ and $Strat_F(s_2) = \neg\langle c\rangle$. According to def. 3.12, the procedure calculates $Strat_F(s_1, s_2) = \bigwedge\{\neg\langle b\rangle, \neg\langle c\rangle\}$ as a conjunction of $Strat_F(s_1)$ and $Strat_F(s_2)$.

Assume that the defender plays conjunct answer $(a.0, \{(a.0 + b.0), (a.0 + c.0)\})_d \rightarrowtail^{*} s_1$. To calculate $Strat_F((a.0, \{(a.0 + b.0), (a.0 + c.0)\})_a)$ one needs $Strat_F(s_1)$ and $Strat_F(s_2)$. However, the attacker plays further only from s_1 therefore $Strat_F(s_2)$ can not be calculated. Modifying def. 3.6 to only add to the conjunction the game position that was chosen by the defender would result in $Strat_F((a.0, \{(a.0 + b.0), (a.0 + c.0)\})_a) = \bigwedge\{\neg\langle b\rangle\}$. However $\bigwedge\{\neg\langle b\rangle\}$ does not distinguish $a.0$ from $(a.0 + b.0) + (a.0 + c.0)$. Thus it can be observed, that playing the spectroscopy game for one conjunct answer on creates formulas that do not distinguish the initial states. That is why I have decided that the hero has to defeat every enemy.

In order to perform the conjunct challenge activity in "The Spectroscopy Invaders", the player has to push the button. After clicking it, fields where the hero and enemies are currently placed at turn red. For each enemy and the

hero, I save their red marked positions. I call these positions *conjunct fields* and the current enemies *conjunct enemies*. After playing the conjunct challenge one enemy is chosen, its figure remains unchanged, however, all other enemies' figures become less visible. From this point, the player plays only against the chosen enemy (the other enemies do not move). If the player loses against the chosen enemy, the level is lost. However, if the player defeats the chosen enemy, the conjunct field of the enemy turns blue again, the hero goes back to his conjunct field and a new enemy is chosen. The process is repeated for every conjunct enemy. If the player defeats the last conjunct enemy, the hero does not move back to his conjunct challenge and the level is won by the player. The order in which the enemies are chosen is not important for the purposes of "The Spectroscopy Invaders".

There is one special case that might occur after the conjunct activity. While playing against one of the conjunct enemies, the enemy might move to multiple fields (as an answer to observation activity). As a result, the enemy is split into multiple enemies and the player has to defeat each of them in order to defeat the original enemy. Therefore, the player might have to play a second, 2-nested, conjunct activity. The 2-nested conjunct activity works analogically to the first one. Only the current enemies' fields and the current hero's field are marked as conjunct fields (with a different color) for the second conjunct activity, without influencing the conjunct fields for the first conjunct activity. Using the description above it is possible to add further n-nested conjunct activities (for $n > 2$) and the player is free to perform them.

In both games [Pea] and [EKH$^+$], a counterpart to conjunct activity is the moment where the attacker has moved (like in observation activity) and the defender is forced to pick a response. However, the attacker does not have to win for every defender's response.

Figure 4.9: Conjunct activity

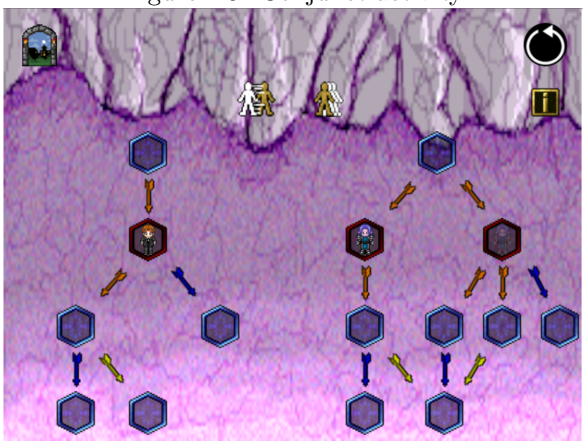
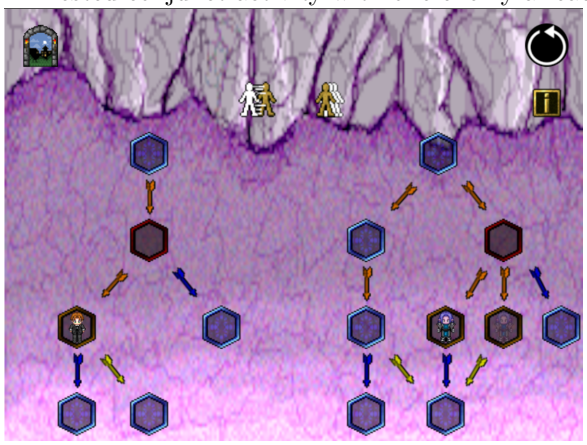


Figure 4.10: 2-nested conjunct activity with one enemy already defeated



4.3.6 Game Over

The goal of the attacker in the spectroscopy game is to lead to a game position $(p, Q)_a$ where p is any state but Q is an empty set. From this position, the attacker can play a conjunct challenge move resulting in $(p, Q)_d$ position. Since Q is an empty set, the defender can not play the conjunct answer move and therefore is stuck and loses. However, if the attacker can no longer play the observation move and both the negation move and the conjunct challenge move do not bring any progress (the attacker creates possibly infinite plays) the defender wins.

Whenever the player's choices in "The Spectroscopy Invaders" lead to a situation where there are no enemies left on the board, the player wins immediately without having to perform any additional game activities. The player loses when he and at least one enemy are in closed fields. No further observation activities are possible and neither the negation activity (if there is one enemy left) nor the conjunct activity (if there are two or more enemies) will bring any progress.

The player in Peacock's game wins when the defender can not mimic the player's last transition and loses when the player is stuck. The player in "One Step Too Far" wins if the number of moves, given at the beginning of a level, has been mimicked. Otherwise, the player loses.

4.3.7 Tasks and Game's Segments

As already mentioned, "The Spectroscopy Invaders" is divided into five segments. All segments consist of four levels. Each level is a separate small "puzzle" where the player and one enemy start in two separate fields. To solve the puzzle, the player has to destroy all enemies. As the technique of immersion and engagement requires, these "puzzles" keep the player engaged and provide straightforward tasks and thus satisfy the requirement of the clear task in flow.

The levels in every segment are ordered from easiest to hardest, and to unlock a harder level one has to solve the previous one first. This ensures good flow. Note that the difficulty is based on the complexity of the graphs (number of field and arrows) and the complexity of the combination of game activities needed

to defeat the enemies. Each segment represents one observation language and allows different amounts of game activities. Four levels for each segment allow one to practice and thus better understand the segment's mechanics whereby I have implemented the practice technique. If I would introduce more observation languages in my work, I would have to implement additional segments. However, too many levels might bore the player and stop him/her from finishing the whole game. That is why I have decided to only introduce 5 languages whereby I have chosen trace, simulation and bisimulation because they are some of the most popular equivalences (simulation and bisimulation are also part of both "One Step Too Far" and Peacock's game). The two remaining equivalences were chosen so that it is possible to reduce the dimensionality of the formula price system (see subsection 2.4.1).

The segments are ordered from the easiest to the hardest one:

1. Trace observations
2. Failure observations
3. Possible-future observations
4. Simulation observations
5. Bisimulation observations

In further segments, I have created levels that require the player to play complex combinations of game activities. The ascending difficulty order and unlocking system build a foundation for good flow, engagement in the game, and give a sense of progression providing improvements in my implementation of immersion and engagement, and feedback techniques.

In Peacock's game and in "One Step Too Far" each segment, just as in my implementation, represents different problems, allows different moves and requires different strategies.

4.3.8 Feedback

To implement the technique of feedback, "The Spectroscopy Invaders" includes several feedback mechanics.

The mechanics that turns the enemies' states red and blue again, after the conjunct activity, allows the player to track the progress in defeating each enemy.

Whenever the player performs a move, I can easily transform the current configuration of hero's field and enemies' fields into a game position (since every field represents an LTS state). With the game position I can traverse the spectroscopy tree and determine whether there is a winning strategy for the player. If no, then the restart button starts pulsing. When the player however does not notice his/her no-win situation and continues playing, a window shows telling the player that there is no possibility to win the level anymore.

Every time the player wins a level, an algorithm evaluates the player's choices and rewards him/her with zero, one, or two stars. To understand the logic behind the rewarding system, the process of rating player's choices has to be explained. Therefore, I will come back to the two-stars rewarding system in section 4.4.3.

Collected stars are transferred to the main menu where one can observe how many stars were collected at each level so far. This together with the levels' unlocking system gives the player an overview of the whole game's progress.

Peacock has also implemented mechanics that indicates that there is no possibility to win from the current game's configuration. "One Step Too Far" does not include such a feedback mechanic, however, both games introduce a similar star-rewarding system. Additionally, "One Step Too Far" allows the players to ask for hints on what is the maximal number of steps that the princess can mimic.

4.4 The Process of Rating Player's Choices

Every time the player wins a level, his/her performance is rated. In order to achieve this, I have to convert the player's choices into something one is capable of rating. One possibility is to convert the combination of game activities played by the player into an HML formula, for which a pricing system has already been introduced in section 2.4.1. Having the price of the HML formula one can easily determine whether the formula fits into the level's observation language. Furthermore, since graphs in every level represent an LTS with its transition relation being defined by the colorful arrows, one can use the spectroscopy procedure to calculate the cheapest distinguishing formulas for the states represented by the initial hero's field and enemy's field in the level. Having a set of these formulas I can even better evaluate the HML formula constructed from the player's choices.

4.4.1 Converting Game Activities into Formula

I develop an HML formula from the game activities performed by the player. The process begins with an empty formula, which is expanded for every game activity played by the player. For every observation activity, an observation $\langle\alpha\rangle$ is added at the end of the formula, where α is equal to the color of the arrow along which the player and the enemies have moved. Whenever the player performs a negation activity, $\neg$ is added at the end of the formula. Whenever a conjunct activity is played, $\bigwedge$ and an opening curly bracket "{" are attached. If the player defeats one of the conjunct enemies the nil-element T and a comma are added. If the defeated enemy was also the last conjunct enemy for the current conjunct activity, the nil-element (no comma needed) is attached and the conjunction is closed by adding a closing curly bracket "}". The formula resulting from the player's choices is called a *player's formula*.

The player's formula is used as a tool to rate player's choices. Originally, the player's formula was supposed to be displayed on the game board. It would help the player to even better understand what impacts does his choices have. However, due to the complexity of this task I have decided to give up on this idea.

4.4.2 Comparing the Results

Since it is now possible to build formulas from the attacker's choices, one can compare the player's formula with the result of the spectroscopy algorithm using

the pricing system from section 2.4.1. However, the player's formula has to be slightly adjusted before comparing it with the algorithm's result. Thus, let $\bigwedge_{i \in I} \varphi_i$ be any conjunction in the player's formula. If for any subformula φ_i applies, that φ_i is equal to φ_j, for $j \in I$ and $j \neq i$, then φ_i is removed from the conjunction.

This reduction is needed because the reduced player's formula is compared with the result of the spectroscopy procedure, which also deletes duplicates.

4.4.3 Rewarding System

Every time the player successfully finishes a level in "The Spectroscopy Invaders", he/she receives feedback on the performance. Both [Pea] and [EKH$^+$] reward players with one star, if they successfully solve a level, but the player can earn additional (up to 2) stars, depending on the number of moves they made in this level. However, I rate the player's performance based on the price of the reduced player's formula. Thus the feedback can not be estimated based on the number of moves alone. Additionally, each segment represents a different observation language and the reduced player's formula should fit into the observation language (see table 2.2).

I have developed a 2-stars rewarding system that gives:

- 2 stars if the reduced player's formula is one of the cheapest distinguishing formulas for the level,
- 1 star if the reduced player's formula is not one of the cheapest distinguishing formulas for the level but fits into the observation language represented by the level, and
- 0 stars if the reduced player's formula does not fit into the observation language represented by the level.

The reason why I have decided to use 2 stars instead of 3 is that I do not want to reward the player with 1 star for winning with a very expensive formula. I provide small instructions for each segment in "The Spectroscopy Invaders". These instructions tell the player what activities (and in what amount) can be used, so that the player's formula fits into the observation language given by the game's segment.

Figure 4.11: a level won with a perfect score (2 stars) in “The Spectroscopy Invaders”

Chapter 5

Conclusion

The goal of this thesis was to adjust the spectroscopy game and thus make it playable for a human player as the attacker.

5.1 Summary

I have implemented the spectroscopy game in such a way that the player can take over the role of the attacker. In order to achieve that I had to make the decision whether there should be a defender algorithm that chooses the best conjunct answer possible, or should the player defeat the defender for every possible answer. Finally, I have decided to implement the second scenario, where the player has to defeat the defender for every conjunct answer possible. As a result, there is no more defender algorithm needed. Furthermore, the defender does not perform any move at all. However, the formula built from the player's choices distinguishes the state represented by the initial hero's field from the state represented by the initial enemy's field.

The game board in "The Spectroscopy Invaders" at every level is designed in such a way so that it looks like an actual LTS. Therefore, the player playing "The Spectroscopy Invaders" can imagine it as making transitions on the LTS.

Based on how many stars does the player get after defeating the enemies, he/she can comprehend whether the choices made during the game were optimal or not.

Because of the different 5 segments in "The Spectroscopy Invaders" the player can learn what are the typical moves needed to distinguish processes that are trace, failure, possible-future, simulation, or bisimulation inequivalent. I believe that the increasing difficulty and different strategies needed to win levels in each segment will make the video game more interesting and will encourage the player to finish all levels. The players that have successfully finished the whole video game should have a good overview of how does the spectroscopy game work.

5.2 Further Work

In this thesis the player plays as the attacker, however, it might be interesting to see how "The Spectroscopy Invaders" could work from the other perspective,

like in the “One Step Too Far” game. Then, I would have to ask myself how to develop a video game where the player could still make meaningful choices even though his moves would be reduced to only conjunct answers.

One could make an effort to develop a defender algorithm that would pick the most optimal (for the defender) conjunct answer for every conjunct challenge played by the attacker. The player would not have to defeat each enemy, but just the chosen one. However, in that case, one would have to answer the question, how could one build a distinguishing formula if the player defeats only one enemy.

In this work, I have not looked at weak equivalences like weak simulation or weak bisimulation. Therefore, it could be interesting to further develop the paper [BN21] by creating a spectroscopy game for weak equivalences. Then, one could try to implement it as a video game.

Every level in “The Spectroscopy Invaders” is predefined, however, I could allow the player to create their own levels (graphs) and add them to the video game. In that case, the player would have to decide for him-/herself which game activities (and how many of them) are required to defeat all enemies so that the resulting player’s formula remains as cheap as possible.

In “The Spectroscopy Invaders”, I use the formula built from the player’s choices only to rate the player’s performance. However, I do not allow the player to see the formula. Therefore, “The Spectroscopy Invaders“ game could be further extended with a mechanic that visualizes the formula as it is being expanded.

Bibliography

[AILS07] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive systems: modelling, specification and verification.* cambridge university press, 2007.

[Ber08] Dietmar Berwanger. Infinite coordination games. In *International Conference on Logic and the Foundations of Game and Decision Theory*, pages 1–19. Springer, 2008.

[BFVG04] Bard Bloom, Wan Fokkink, and Rob J Van Glabbeek. Precongruence formats for decorated trace semantics. *ACM Transactions on Computational Logic (TOCL)*, 5(1):26–78, 2004.

[BN21] Benjamin Bisping and Uwe Nestmann. Deciding all behavioral equivalences at once: A game for linear-time–branching-time spectroscopy, 2021. `arXiv:2109.15295`.

[dFERP09] David de Frutos Escrig, Carlos Gregorio Rodríguez, and Miguel Palomino. On the unification of process semantics: Equational semantics. *Electronic Notes in Theoretical Computer Science*, 249:243–267, 2009.

[EKH+] Johanna England, Julian Dschana Kremb, Lucas Hoschar, Lin Fun Cheung, Duy Khang David Dinh, Fabian Stroschkea, Iliya Velev, and Fabian Onur Willner. One step too far. `https://pr.mtv.tu-berlin.de/showcase/one-step-too-far/`. Browser game, Accessed: 2021-11-15.

[FvGd06] Wan Fokkink, Rob van Glabbeek, and Paulien de Wind. Compositionality of hennessy–milner logic by structural operational semantics. *Theoretical Computer Science*, 354(3):421–440, 2006. Foundations of Computation Theory (FCT 2003). URL: `https://www.sciencedirect.com/science/article/pii/S0304397505008819`, `doi:https://doi.org/10.1016/j.tcs.2005.11.035`.

[Gla90] RJ van Glabbeek. The linear time-branching time spectrum. In *CONCUR'90 Theories of Concurrency: Unification and Extension.* 1990.

[Kor91] Henri Korver. Computing distinguishing formulas for branching bisimulation. In *International Conference on Computer Aided Verification*, pages 13–23. Springer, 1991.

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

[Pea] Dominik Andre Peacock. Weak process equivalences. `https://www.concurrency-theory.org/rvg-game/`. Browser game, Accessed: 2021-11-15.

[Pea20] Dominik Andre Peacock. *Process equivalences as a video game*. Bachelor's thesis, Berlin Institute of Technology, 2020.

[San11] Davide Sangiorgi. *Introduction to bisimulation and coinduction*. Cambridge University Press, 2011.