Integration of the weak Spectroscopy into CAAL

Timo Straßnick Fabian Ozegowski

March 2024

Abstract

The principle of spectroscopy allows verifying many behavioral equivalences for transition systems simultaneously with a single computation by analyzing the expressiveness of distinguishing formulas for the respective processes. In this work, the weak version of the Spectroscopy Energy Game is implemented for verification purposes and integrated into CAAL, a web application for modeling, visualization, and verification of processes. Furthermore, we provide an overhaul of the user interface in CAAL for the verify view as well as the strong spectroscopy energy game view to improve user-friendliness.

Contents

1 Introduction

While analyzing the behavior of concurrent processes, one key problem is the verification of behavioral equivalences. Those relations between processes specify in what manner processes are similar since there is no universal equivalence but rather many distinct notions of equivalence that characterize distinct behavior. Verifying these notions of equivalence usually has to be done individually with specifically tailored methods. A different approach is proposed in [1], where a logic game is introduced that implicitly builds formulas that describe differences in behavior of two processes to then derive behavioral equivalences from those formulas. This idea is further refined in [2] and for weak notions of equivalence also in [3]. This work mainly refers to the latter paper as we will primarily deal with weak behavioral equivalences.

This work concerns the implementation of the weak spectroscopy for verification purposes into CAAL, a web tool for modeling, visualizing and verifying processes as well as an overhaul of the user interface for the verification view and the strong spectroscopy energy game that was implemented as part of a bachelor's thesis [4].

In the first half, we go over the theory, starting with labeled transition systems and Hennessy-Milner logic. We then provide a brief overview of the spectrum of notions of equivalences, the linear-time-branching-time spectrum to further touch distinguishing formulas and finally the weak spectroscopy energy game. The second half presents our contributions. We will first provide an overview of the base version of CAAL we started from, state our reasoning behind the aspects we wanted to change and then present our results until finally going over some technical aspects.

The final result of this work is available under `https://equivio.github.io/CAAL/`.

2 Theory

The following four chapters present the theory behind the spectroscopy as defined in [3]

2.1 Modeling Processes and Characterizing their Behavior

To analyze the behavior of processes, we first need to have an underlying structure to model these processes. CAAL works with Labeled Transition Systems as a means for that.

Definition 1 (Labeled Transition System)**.**
A Labeled Transition System (LTS) is a Tuple $(\mathcal{P}, \mathcal{A}, \rightarrow)$ where $\mathcal{P}$ is a set of processes, $\mathcal{A}$ a set of actions and $\rightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ a transition relation.

The special action $\tau \in \mathcal{A}$ indicates internal behavior that is unobservable. We use $\twoheadrightarrow$ as shorthand notation for the reflexive transitive closure of these internal steps $\xrightarrow{\tau}{}^{*}$.

We write $p_1 \xrightarrow{a} p_2$ if a process $p_1 \in \mathcal{P}$ can perform an action a that results in process $p_2 \in \mathcal{P}$, i.e. $(p_1, a, p_2) \in \rightarrow$. Furthermore, we define $p_1 \stackrel{a}{\Longrightarrow} p_2$ as the "weak" transition corresponding to $p_1 \twoheadrightarrow p_1' \xrightarrow{a} p_2' \twoheadrightarrow p_2$ if $a \neq \tau$ and $p_1 \twoheadrightarrow p_2$ if $a = \tau$. Ultimately, we use the notation $P_1 \xrightarrow{a} P_2$ with $P_1, P_2 \subseteq \mathcal{P}$ and $a \in \mathcal{A}$ if $P_2 = \{p_2 \in \mathcal{P} \mid \exists p_1 \in \mathcal{P}.\ p_1 \overset{a}{\rightarrowtail} p_2\}$.

To characterize behavior of the modeled processes, we use Hennessy-Milner logic [5], an extension of propositional logic by a modal operator that can be interpreted as an observation. The grammar in 2 provides a set of HML-Formulas that characterize stability-respecting branching bisimilarity.

Definition 2 (Syntax of stability-respecting branching Hennessy-Milner logic)**.**

Stability-respecting branching Hennessy-Milner logic, HML_{srbb}, is inductively defined by the followng grammar starting with φ:

$$
\begin{array}{lll}
\varphi \coloneqq & \langle\varepsilon\rangle\chi & \text{with } \varepsilon \notin \mathcal{A} \text{ signaling no visible action} \\
& \Big| \bigwedge_{i\in I\subseteq\mathbb{N}} \{\psi_i\} & \\
\chi \coloneqq & \langle a\rangle\varphi & \text{with } a \in \mathcal{A}\setminus\{\tau\} \\
& \Big| \bigwedge_{i\in I\subseteq\mathbb{N}} \{\psi_i\} & \\
& \Big| \bigwedge_{i\in I\subseteq\mathbb{N}} \{\psi_i, \neg\langle\tau\rangle \bigwedge \varnothing\} & \text{(stable conjunction)} \\
& \Big| \bigwedge_{i\in I\subseteq\mathbb{N}} \{\psi_i, (a)\varphi\} & \text{(branching conjunction)} \\
\psi \coloneqq & \langle\varepsilon\rangle\chi & \\
& \Big| \neg\langle\varepsilon\rangle\chi &
\end{array}
$$

Definition 3 (Semantics of stability-respecting branching Hennessy-Milner logic)**.**

Let $T = (\mathcal{P}, \mathcal{A}, \to)$ be an LTS and $p \in \mathcal{P}$.
The satisfaction relation $\models \subseteq 2^{\mathcal{P}} \times \mathrm{HML}_{\mathrm{srbb}}$ is defined by the following rules:

$$
\begin{array}{lcll}
p & \models & \langle a\rangle\varphi & \text{if } \exists p'.p' \models \varphi \wedge p \xrightarrow{a} p' \\
p & \models & \langle\varepsilon\rangle\varphi & \text{if } \exists p'.p' \models \varphi \wedge p \twoheadrightarrow p' \\
p & \models & (\tau)\varphi & \text{if } p \models \varphi \vee \exists p'.p' \models \varphi \wedge p \xrightarrow{\tau} p' \\
p & \models & \bigwedge_{i\in I\subseteq\mathbb{N}} \{\psi_i\} & \text{if } \forall i \in I.p \models \psi_i
\end{array}
$$

The notation $(a)\varphi$ is interpreted as $\langle a\rangle\varphi$ if $a \neq \tau$ and correspondingly, $p \xrightarrow{(a)} p'$ is interpreted as $p \xrightarrow{a} p'$ if $a \neq \tau$.

Example 1.
Let us consider the LTS defined by the CCS terms

$$
\begin{aligned}
P_1 &= a.P_2 + b.0 + a.0 \\
P_2 &= a.P_2 + b.0
\end{aligned}
$$

that can be visualized in graph form as seen in 1

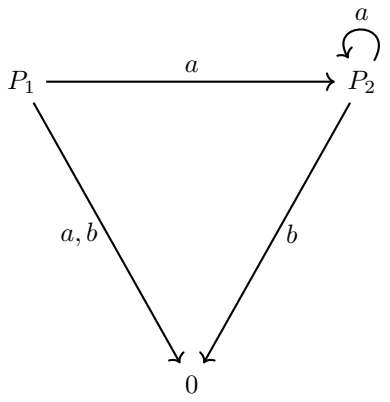

Figure 1: Example of an LTS in graph form

The formula $\varphi_1 \coloneqq \langle a \rangle \langle b \rangle \bigwedge \varnothing$ formalizes the proposition that there exists an action a such that an action b is possible afterwards. The formula is true for both P_1 and P_2, i.e. $P_1 \models \varphi_1$ and $P_2 \models \varphi_1$. For formula $\varphi_2 \coloneqq \langle a \rangle \bigwedge \{\neg \langle a \rangle \bigwedge \varnothing\}$, $\varphi_2 \models P_1$ holds and $\varphi \not\models P_2$ since P_2 can only loop on itself with action a. Therefore, φ_2 is called a "distinguishing formula". These formulas play a major role in deciding which relations apply to a pair of processes and are thus examined further in chapter 2.3.

2.2 Behavioral Equivalences

This chapter provides a minimalistic introduction to the relations that characterize behavior between a pair of processes. The commonly used term "behavioral equivalences" suggests that these relations are equivalence relations, but in this context, "preorder" is the more appropriate term.

Definition 4 (Preorder and equivalence relation)**.**
A preorder is a binary relation R on a set S that fulfills the following two criteria for every $a, b, c \in S$

- $(a, a) \in R$ (reflexivity)
- $(a, b) \in R \land (b, c) \in R \longrightarrow (a, c) \in R$ (transitivity)

An equivalence relation is a preorder that is also symmetric, i.e. $(a, b) \in R \longrightarrow (b, a) \in R$ holds.

2.2.1 Trace Semantics

Let $T = (\mathcal{P}, \mathcal{A}, \rightarrow)$ be an LTS.

Definition 5 (Weak traces)**.**
Let $p_0, ..., p_n \in \mathcal{P}$ be a sequence of processes, $a_1, ..., a_n \in \mathcal{A} \setminus \{\tau\}$ a sequence of actions and $n \in \mathbb{N}^+$ such that for all $0 \leq j < n$ there is a weak transition $p_j \overset{a_{j+1}}{\Longrightarrow} p_{j+1}$. Then, $(a_1, ..., a_n) \in (\mathcal{A} \setminus \{\tau\})^*$ is called a weak trace of $p_0 \in \mathcal{P}$. We denote

$$\text{Traces}_w(p) \coloneqq \{w \in (\mathcal{A} \setminus \{\tau\})^* \mid w \text{ is a weak trace of } p \text{ in } \mathcal{P}\}.$$

as the set of weak traces of p.

Definition 6 (Weak trace inclusion)**.**
Let $p_1, p_2 \in \mathcal{P}$. p_1 is weakly trace-included by p_2 iff $\text{Traces}_w(p_1) \subseteq \text{Traces}_w(p_2)$.

Intuitively, p_1 is weakly trace-included by p_2 if p_2 can at least match all sequences of actions possible from p_1, skipping over all τ-actions.

Example 2.
For this example, we consider the LTS defined by the terms

$$\begin{aligned} P_1 &= \tau.P_1 + a.b.0 \\ P_2 &= a.P_2' \\ P_2' &= b.P_2 + \tau.b.0 \end{aligned}$$

as illustrated in 2.

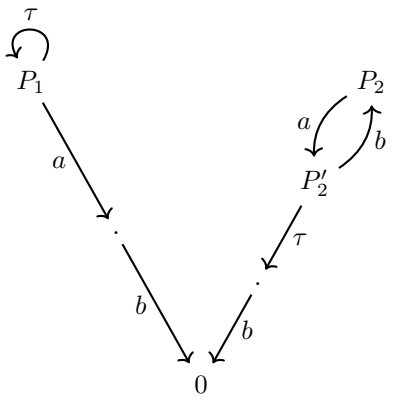

Figure 2: Example of a process P_1 being weakly trace-included by another process P_2.

Here, P_1 is weakly trace-included by P_2 since $\text{Traces}_w(P_1) = \{(a), (a, b)\} \subseteq L((a, b)^* a(\epsilon + b)) = \text{Traces}_w(P_2)$. However, the relation in not symmetric, i.e. $\text{Traces}_w(P_1) = \text{Traces}_w(P_2)$. For symmetry to apply, $\text{Traces}_w(P_2) \subseteq \text{Traces}_w(P_1)$ would also have to hold but $P_2' \xrightarrow{b} P_2$ prevents this.

2.2.2 Weak Simulation

Let $T = (\mathcal{P}, \mathcal{A}, \rightarrow)$ be an LTS.

Definition 7 (Weak simulation)**.**
Weak simulation is a binary relation $R \subseteq \mathcal{P} \times \mathcal{P}$ where the following condition holds for every pair of processes $p_1, p_2 \in \mathcal{P}$ and for all actions $a \in \mathcal{A}$:

- if $(p_1, p_2) \in R$ and $p_1 \xrightarrow{a} p_1'$, then $\exists p_2' \,.\, p_2 \stackrel{a}{\Rightarrow} p_2'$ and $(p_1', p_2') \in R$

p_1 is weakly simulated by p_2 if there exists a simulation R with $(p_1, p_2) \in R$.

Example 3 (Weak simulation)**.**
Let us again consider the LTS from 2. There, P_1 is weakly simulated by P_2 since there exists a relation R_1 with

$$R_1 = \{(P_1, P_2), (b.0, P_2'), (0, 0)\}.$$

In the reverse direction, trying to build a relation R_2 fails. To provide such a relation proving P_2 to be weakly simulated by P_1, one has to start with $R_2 = \{(P_2, P_1)\}$ and continue by adding $(P_2', b.0)$ because of the transition

$P_2 \xrightarrow{a} P_2'$. Since $P_2' \xrightarrow{b} P_2$, the next pair of processes to be added are $(P_2, 0)$. Finally, there is no $P \in \mathcal{P}$ such that $0 \stackrel{a}{\Rightarrow} P$, failing to match $P_2 \xrightarrow{a} P_2'$ and thus violating the criterion for R_2 to be a weak simulation.

2.2.3 The Linear-Time-Branching-Time Spectrum

The graph in Figure 3 depicts the linear-time-branching-time spectrum that was originally proposed by van Glabbeek in [6] and further adapted in [3] to visualize the whole of behavioral equivalences relevant for the weak spectroscopy energy game hierarchically. The directed edges of the graph can be interpreted as implications, meaning that if there is an edge from relation x to relation y and x applies to a pair of processes, one can conclude that y also applies. Therefore, stability-respecting branching bisimilarity contitutes the finest, most strict relation whereas trace inclusion represents the broadest, most general behavioral equivalence.

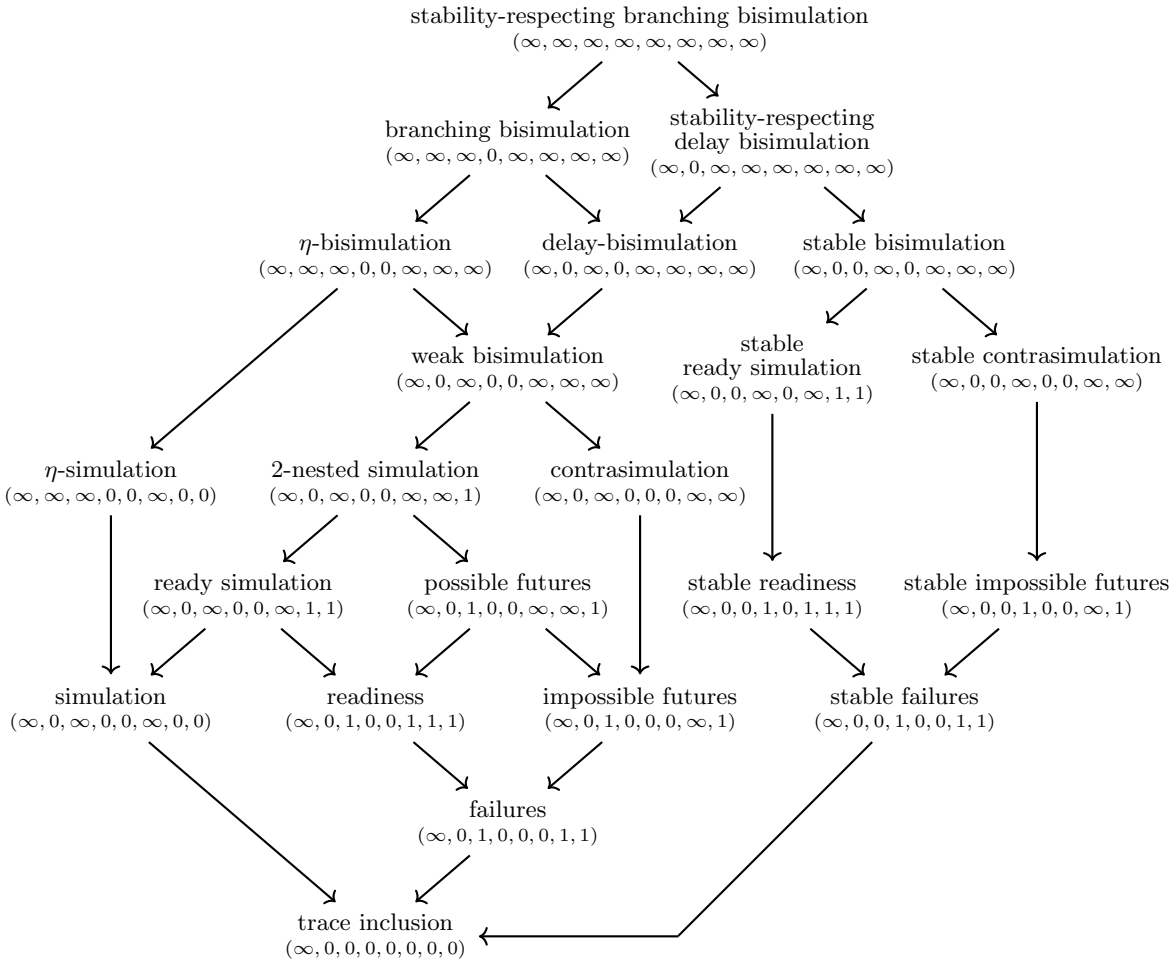

Figure 3: Hierarchy of behavioral equivalences

2.3 Distinguishing Formulas

In this chapter, we will examine the concept of distinguishing formulas and how to quantify them as an interpretation of a price or cost in order to make conclusions about which behavioral equivalences apply to the pair of processes that are being distinguished.

Definition 8 (Distinguishing formula)**.**
Let $T = (\mathcal{P}, \mathcal{A}, \rightarrow)$ be an LTS and $p_1, p_2 \in \mathcal{P}$. A formula $\varphi \in \mathrm{HML}_{\mathrm{srbb}}$ is a

distinguishing formula that distinguishes p_1 from p_2 if $p_1 \models \varphi$ and $p_2 \not\models \varphi$.

2.3.1 "Cost" of Distinguishing Formulas

Before we can examine how one can break down distinguishing formulas to evaluate the corresponding price or cost, we first need to have a structure that can represent these costs. For that, we use energies.

Definition 9 (Energies)**.**
An energy is an eight-dimensional vector $e \in E = (\mathbb{N} \cup \{\infty\})^8$.

Each dimension of an energy represents the depth of an operator in a certain way. The term "depth" means that we count occurrences of a specific operator along a path from root to leaf of the tree that is spanned by the syntax of the formula to be examined. For every component and every operator respectively, we take the highest number of occurences out of all those paths. The dimensions correspond to:

1. Depth of observations
2. Depth of branching conjunctions
3. Depth of non-stable conjuncions
4. Depth of stable conjunctions
5. Depth of conjunctions not preceded by $\langle\epsilon\rangle$
6. Depth of observations of positive clauses in conjunctions
7. Depth of observations of negative clauses in conjunctions
8. Depth of negations

2.3.2 Languages of Distinguishing Formula corresponding to Behavioral Equivalences

Now that we have a metric of how complex a distinguishing formula is, i.e. how much "syntactical effort" is required to tell two processes apart, we can formulate languages of formulas that correspond to behavioral equivalences. Figure 3 provides an upper bound for each relation, meaning that to tell two processes apart that are not in a certain relation, we can provide a distinguishing formula that at max has a cost of the upper bound of that particular relation.

Definition 10 (Languages of behavioral equivalences)**.**
Let R be a relation from Figure 3 with corresponding upper bound e_R. The Language $\mathcal{O}_R \subseteq \mathrm{HML}_{\mathrm{srbb}}$ is defined as the set of formulas $\varphi \in \mathrm{HML}_{\mathrm{srbb}}$ whose costs are component-wise lower or equal than e_R.

Corollary 1.
Let $p_1, p_2 \in \mathcal{P}$ and let R be a relation. If there exists no $\varphi \in \mathcal{O}_R$ that distinguishes p_1 from p_2, then $(p_1, p_2) \in R$.

2.4 The Weak Spectroscopy Energy Game

In this chapter, we address the weak [1] spectroscopy energy game. The concept of this energy game is for the attacker to implicitly contstruct distinguishing formulas that have a cost that is less than a predefined energy budget limiting their expressiveness. The defender on the other hand tries to maximize those costs to ultimately exceed the provided budget.

Before we define the game itself, we first need to define how energies can be modified:

Definition 11 (Energy updates)**.**
An energy update is an eight-dimensional vector $u = (u_1, ..., u_8)$ where each component u_i is either one of two forms:

- relative: $u_i \in \{-1, 0\}$
- min-selection: $u_i \in min_D$ where $D \subseteq \{1, ..., 8\}$

We call the set of all energy updates Up and the function applying an update to an energy upd with $upd : E \times Up \to E$. For $upd((e_1, ..., e_8), (u_1, ..., u_8)) = (e'_1, ..., e'_8)$, e'_i is defined as $e_i + u_i$ if u_i is of relative form or $min_{d \in D} e_d$ if u_i is of min-selection form. Any update that would result in a negative component is defined to be undefined.

Definition 12 (The weak spectroscopy energy game)**.**
Let $T = (\mathcal{P}, \mathcal{A}, \to)$ be the LTS to be played on. The weak spectroscopy game $\mathcal{G}_w = (G, G_D, \rightarrowtail, w, g_o, e_0)$ consists of

- a set of game positions G partitioned into defender positions G_d and attacker positions $G \setminus G_d$,
- a move relation $\rightarrowtail \subseteq G \times G$,
- a weight funtion $w : \rightarrowtail \to Up$,
- a starting position $g_0 \in G$ and
- a starting energy budget $e_0 \in E$.

[1] The version of the spectroscopy energy game relevant for this work is called "weak" since it features "weak" transitions accounting for internal activity as opposed to the version introduced in [2]

We write $g \overset{u}{\rightarrowtail} g'$ for $g \rightarrowtail g'$ and $w(g, g') = u$.

The game positions can be categorized into the following distinct types where the subscript a and d imply attacker and defender positions, respectively and $p, p', q \in \mathcal{P}$, $Q, Q_a \in 2^{\mathcal{P}}$ and $a \in \mathcal{A}$:

- general attacker positions $(p, Q)_a$
- attacker delayed positions $(p, Q)_a^\epsilon$
- attacker clause positions $(p, q)_a^\wedge$
- defender conjunction positions $(p, Q)_d$
- defender stable conjunction positions $(p, Q)_d^s$
- defender branching positions $(p, a, p', Q, Q_a)_d^\eta$
- attacker branching positions $(p, Q)_a^\eta$

The game move relation $\rightarrowtail$ is composed of several move types:

- delay $(p, Q)_a \overset{(0,0,0,0,0,0,0,0)}{\rightarrowtail} (p, Q')_a^\epsilon$, with $Q \twoheadrightarrow Q'$
- procrastination $(p, Q)_a^\epsilon \overset{(0,0,0,0,0,0,0,0)}{\rightarrowtail} (p', Q)_a^\epsilon$, with $p \xrightarrow{\tau} p'$ and $p \neq p'$
- observation $(p, Q)_a^\epsilon \overset{(-1,0,0,0,0,0,0,0)}{\rightarrowtail} (p', Q')_a$, with $p \xrightarrow{a} p', Q \xrightarrow{a} Q'$ and $a \neq \tau$
- finishing $(p, \varnothing)_a \overset{(0,0,0,0,0,0,0,0)}{\rightarrowtail} (p, \varnothing)_d$
- early conjunction $(p, Q)_a \overset{(0,0,0,-1,0,0,0)}{\rightarrowtail} (p, Q)_d$, with $Q \neq \varnothing$
- late instable conjunction $(p, Q)_a^\epsilon \overset{(0,0,0,0,0,0,0,0)}{\rightarrowtail} (p, Q)_d$
- conjunction answer $(p, Q)_d \overset{(0,0,-1,0,0,0,0,0)}{\rightarrowtail} (p, q)_a^\wedge$, with $q \in Q$
- positive clause $(p, Q)_a^\wedge \overset{(min_{\{1,6\}},0,0,0,0,0,0,0)}{\rightarrowtail} (p, Q')_a^\epsilon$, with $\{q\} \twoheadrightarrow Q$
- negative clause $(p, Q)_a^\wedge \overset{(min_{\{1,7\}},0,0,0,0,0,0,-1)}{\rightarrowtail} (q, Q')_a^\epsilon$, with $\{p\} \twoheadrightarrow Q$ and $p \neq q$
- late stable conjunction $(p, Q)_a^\epsilon \overset{(0,0,0,0,0,0,0,0)}{\rightarrowtail} (p, Q')_d^s$, with $Q' = \{q \in Q \mid \neg\exists q' \in \mathcal{P}.\ q \xrightarrow{\tau} q'\}$ and $\neg\exists p' \in \mathcal{P}.\ p \xrightarrow{\tau} p'$
- conjunction stable answer $(p, Q)_d^s \overset{(0,0,0,-1,0,0,0,0)}{\rightarrowtail} (p, q)_a^\wedge$, with $q \in Q$
- empty stable conjunction answer $(p, Q)_d^s \overset{(0,0,0,-1,0,0,0,0)}{\rightarrowtail} (p, Q)_d$, with $Q = \varnothing$

- branching conjunction $(p, Q)_a^{\epsilon} \overset{(0,0,0,0,0,0,0,0)}{\rightarrowtail} (p, a, p', Q \setminus Q_a, Q_a)_d^{\eta}$, with $p \xrightarrow{a} p'$, $Q_a \neq \varnothing$ and $Q_a \subseteq Q$
- branching answer $(p, a, p', Q, Q_a)_d^{\eta} \overset{(0,-1,-1,0,0,0,0,0)}{\rightarrowtail} (p, q)_a^{\wedge}$, with $q \in Q$
- branching observation $(p, a, p', Q, Q_a)_d^{\eta} \overset{(min_{\{1,6\}},-1,-1,0,0,0,0,0)}{\rightarrowtail} (p', Q')_a^{\eta}$, with $Q_a \xrightarrow{(a)} Q'$
- branching accounting $(p, Q)_a^{\eta} \overset{(-1,0,0,0,0,0,0,0)}{\rightarrowtail} (p, Q)_a$

Definition 13 (Plays, energy levels and winning conditions)**.**
A play of $\mathcal{G}_w$ is a path $p = (g_0, g_1, ...) \in G^*$ where $g_i \rightarrowtail g_{i+1}$. For every round i of a play, there is an energy level $EL_p(i)$ that is recursively defined as $EL_p(0) = e_0$ and $EL_p(i+1) = upd(EL_p(i), u_i)$.

If at least one component of the energy level would become negative at any point in a play, i.e. $upd(EL_p(i), u_i)$ is undefined for any round i, then the defender wins that play. Infinite plays are also won by the defender. If there is a position g in a play where no further move is possible, then the player associated with that move loses, i.e. the attacker wins if $g \in G_d$ and the defender wins otherwise.

Definition 14 (Winning budgets)**.**
A strategy is a map from plays to game positions reachable from the last position of each play. If all of those last positions are exclusively attacker positions or defender positions, we call that strategy "attacker strategy" or "defender strategy", respectively. We call a strategy a winning strategy for a player if all plays carried out according to that strategy are winning for that player. A player having a winning strategy for $\mathcal{G}[g_0, e_0]$ means that said player wins $\mathcal{G}$ with starting energy budget e_0 from position g_0, if played perfectly even by both sides. We denote $Win_a(g) \coloneqq \{e \in E \mid \text{attacker wins} \mathcal{G}[e_0, g_0]\}$ as the set of attacker winning budgets.

For every energy $e' \in E$ that is greater in every component than some $e \in Win_a(g)$, e must also be included in $Win_a(g)$. Therefore, $Win_a(g)$ is either the empty set, namely in the case of a non-existent attacker winning strategy or $\mid Win_a(g) \mid = \infty$. Since we are only interested in minimal winning budgets that define the bounds winning, we additionally define $Win_a^{min}(g)$ $\coloneqq \{e \in Win_a(g) \mid \neg \exists e' \in Win_a(g).\ e'$ is less than or equal in every component of $e \ \wedge e \neq e'\}$.

2.4.1 Computing behavioral equivalences

With the proposition from 1 we can conclude that two processes $p_1, p_2 \in \mathcal{P}$ must be in a relation R if $e_R \notin Win_a((p_1, \{p_2\})_a)$ where e_R represents the upper bound of cost of the formulas in $\mathcal{O}_R$. To verify multiple behavioral equivalences $R_1, ..., R_n$ at once, one consequently has to compute the minimal attacker winning budgets $Win_a^{min}((p_1, \{p_2\})_a)$ to avoid computing for every e_{R_i}, $i \notin \{1, ..., n\}$, if $e_{R_i} \in Win_a((p_1, \{p_2\})_a)$, i.e. to avoid computing wether the attacker loses $\mathcal{G}[e_{R_i}, (p_1, \{p_2\})_a]$.

Definition 15 (Computation of minimal winning budgets)**.**
The minimal attacker winning budgets $Win_a^{min}(g)$ for a game postion g are inductively characterized as follows:

- the attacker wins with any budget where the defender has no possible move: $\{0\}^8 \in Win_a^{min}(g)$ with $g \in G_d$ where $\neg\exists g' \in G.\ g \rightarrowtail g'$
- the attacker wins if the defender has possible moves g_i' left with $g \overset{u_i}{\rightarrowtail} g_i'$, $g \in G_d$ and the attacker has a budget not lower than any budget that would be needed for winning after any of the defender's moves: $(\forall i.\ upd(e, u_i) \in Win_a(g_i')) \longrightarrow e \in Win_a(g)$
- the attacker wins if they have a move whose resulting energy budget is also a winnig budget for the resulting position: If $g \in G \setminus G_d,\ g \overset{u}{\rightarrowtail} g'$ and $upd(e, u) \in Win_a(g')$, then $e \in Win_a(g)$

The minimal attacker winning budgets are thus computed in reverse to the direction the energy game is being played, i.e. from positions where the defender is stuck successively to the starting position. Therefore we also need to compute the energy updates in reverse.

Definition 16 (Inverse updates)**.**
The inverse of the update function upd, is defined as $upd^{-1}(e', u) = sup(\{e\} \cup \{e_{min}(i) \mid \exists i \in \{1, ..., 8\}.\ u_i = min_{\{x_i, y_i \mid x_i, y_i \in \{1,...,8\} \wedge x \neq y\}}\})$ where $e_i = e_i' - u_i$ if $u_i \in \{-1, 0\}$ and $e_{min}(i)_{x_i} = e_{min}(i)_{y_i} = e_i'$ if $u_i = min_{\{x_i, y_i\}}$ und $e_{min}(i)_k = 0$ for $y_i \neq k \neq x_i\ \forall i$.

Now, we can compute the minimal attacker winning budgets with the algorithm in 1 proposed in [2]:

The algorithm initializes, according to 15, a to-do list with all game states where the defender is stuck, sets the associated attacker winning budgets to

Input: $\mathcal{G}_w = (G, G_D, \rightarrowtail, w)$

1 $Win_a \coloneqq [g \mapsto \{\} \mid g \in G]$
2 todo $\coloneqq \{g \in G_d \mid g \not\rightarrowtail\}$
3 **while** todo $\neq \varnothing$ **do**
4 $\quad g \coloneqq$ some $x \in$ todo
5 $\quad$ todo $\coloneqq$ todo $\setminus \{g\}$
6 $\quad$ **if** $g \in G \setminus G_d$ **then**
7 $\quad\quad newWin_a \coloneqq \min(Win_a[g] \cup \{\mathrm{upd}^{-1}(e', u) \mid g \overset{u}{\rightarrowtail} g' \wedge e' \in Win_a[g']\})$
8 $\quad$ **else**
9 $\quad\quad G' \coloneqq \{g' \mid g \overset{u}{\rightarrowtail} g'\}$
10 $\quad\quad$ options $\coloneqq \{(g', \mathrm{upd}^{-1}(e', u)) \mid g \overset{u}{\rightarrowtail} g' \wedge e' \in Win_a[g']\}$
11 $\quad\quad$ **if** $G' \subseteq$ dom(options) **then**
12 $\quad\quad\quad newWin_a \coloneqq \min(\{\sup_{g' \in G'} strat(g') \mid strat \in (G \to E \wedge \forall g'.\, strat(g') \in \text{options}(g'))\})$
13 $\quad\quad$ **else**
14 $\quad\quad\quad newWin_a \coloneqq \varnothing$
15 $\quad$ **if** $newWin_a \neq Win_a[g]$ **then**
16 $\quad\quad Win_a[g] \coloneqq newWin_a$
17 $\quad\quad$ todo $\coloneqq$ todo $\cup \{g_{\text{pre}} \mid \exists u.\, g_{\text{pre}} \overset{u}{\rightarrowtail} g\}$
18 $Win_a^{min} \coloneqq Win_a$
19 **return** Win_a^{min}

Algorithmus 1: Algorithm for computing minimal attacker winning budget by propagating budgets from positions where the defender is stuck successively to game positions farther away from those leaf nodes in the game graph. Taken and adapted from [2].

0^8, and iteratively processes the states. New and previously explored game states are repeatedly added to the list whenever a new opportunity to win in that state is found. In line 7, the inversely updated budgets are stored as provisional winning budgets of attack positions. In line 11, the attacker winning budgets are only updated if there are winning budgets in all subsequent positions. If so, the minima of the suprema of all possible combinations of winning budgets representing each subsequent position are stored as new winning budgets (line 12). This process continues until the to-do list no longer contains any elements. Ultimately, we can determine which behavioral equivalences apply to the two processes declared in the input from the set of winning budgets of the starting position.

3 Implementation

This section will cover the practical part of this project. After establishing the basis upon which the implementation was constructed and introducing initial goals, the results will be presented after which the code structure and some technical points of interest will be highlighted.

3.1 Baseline

The basis of this project's implementation was an extended version of CAAL, a web-tool offering modeling, visualization and verification of concurrent processes which was created as part of a master thesis and extended as part of a bachelor thesis. This extended version can be found under `https://fabian-o01.github.io/CAAL/`.

The web application's contents are split into four different parts. Initially, users are presented with the edit page as seen in figure 4 and cannot access any of the other three pages. Here, CCS and TCCS processes can be specified using the syntax found upon pressing the syntax button on the right of the page header. Alternatively projects can be created, stored, loaded or exported using the project drop down menu located on the left of the page header. Once valid processes have been specified in the edit page, the other three pages are unlocked and can be accessed via the page header.

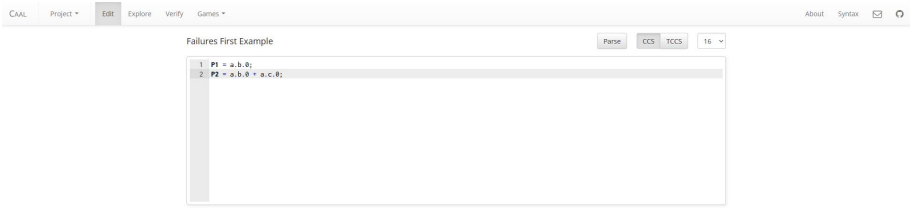

Figure 4: Edit page

First of these pages is the explorer which is displayed in figure 5. Here, the LTS related to the specified processes can be displayed as a graph, where the current position in the LTS is highlighted in red. Changing the currently displayed process or its display options is done at the top of the page. While the majority of the page's space is used for the canvas containing the graph, the bottom contains possible transitions from the current state. Choosing a transition updates the coloring of the LTS graph to reflect the new state. To

make the selection of transitions as intuitive as possible, hover effects highlight the corresponding edge of the graph.

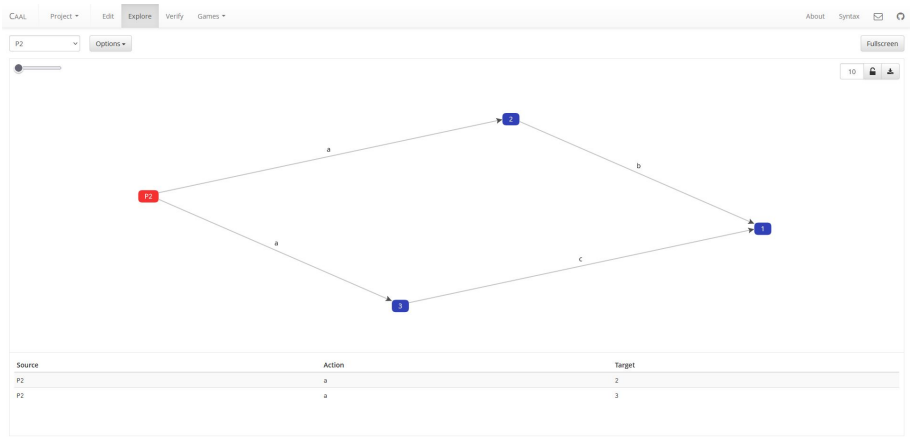

Figure 5: Explore page

The second page contains the verifier (figure 6), which allows users to check processes for different equivalences and preorders by adding a corresponding property. Supported relations can be seen in table 1. While all of these can be added individually, an option to add all relations covered by the strong spectroscopy at once is available by choosing the "BJN-Algorithm" option from the relation drop down menu. Relations added by this option can be verified simultaneously using the "Verify with BJN-Algorithm"-Button. Alternatively, the verifier can be used for model-checking of a process, for which a HML formula has to be stated.

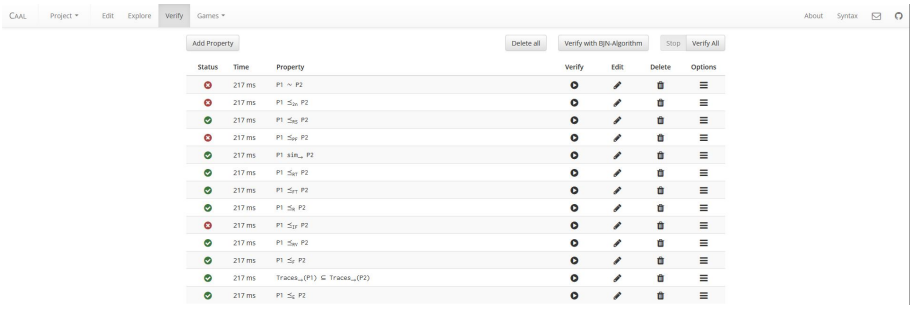

Figure 6: Verify page

Once the desired property has been selected it is added to the property-list

supported relation	strong	weak	equivalence checking
bisimulation	✓	✓	
two-nested simulation	✓		
ready simulation	✓		
possible futures	✓		
simulation	✓	✓	✓
readiness traces	✓		
failure traces	✓		
readiness	✓		
impossible futures	✓		
revivals	✓		
failures	✓		
trace inclusion	✓	✓	✓
enabledness	✓		

Table 1: Supported relations of extended-CAAL

of the verifier page. Until an added property has been checked, its unchecked status is displayed using an icon in the status column of the property-list. The verification can be done individually or collectively and results in marking properties as either correct or incorrect by updating their icon and writing the time used for computation into their time column. Besides options for editing and deleting which are available upon creation, verified properties feature the option to generate a distinguishing formula if possible or to play a supported game.

Said games are implemented on the third page of CAAL as seen in figure 7. Accessing the game page via the options tag of verifier-properties loads the corresponding configuration, while manual access requires the configuration to be adjusted. The game page contains three different games which can be selected via a drop down menu in the page header: The equivalence games consisting of simulation and bisimulation games, a HML game and the spectroscopy energy game for strong relations. These three game-types each have individual views, which share a common structure. While configurations of the games such as choosing the attacking or defending position or restarting the game are accessed at the top of the page, its middle displays current game states in the form of graphs. In order to make a move players choose a desired action from the list of possible moves on bottom right. The chosen move is logged on the bottom left of the page after which the opponent chooses an optimal move, which gets logged as well and leads to the next turn of the players, repeating until one side wins.

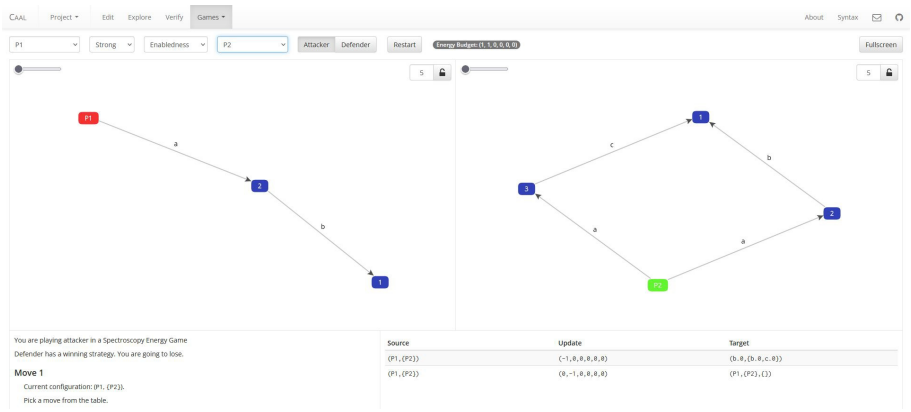

Figure 7: Spectroscopy energy game page

3.2 Vision

The main goals of this project were the integration of the weak spectroscopy into the CAAL web-tool and an overhaul to the game view adjusted for spectroscopy energy games. These goals entailed an evaluation of the code base to determine which parts provided by the extended version of CAAL could be utilized and what was to be adjusted or added.

The inclusion of the weak spectroscopy would grant users access to the verification of new relations included in figure 3 and therefore increase the verification functionality of CAAL. To ensure desired behavior of the verifier, test cases had to be devised and implemented. Conveying both the hierarchies of covered relations and the spectroscopy's beneficial quality of simultaneous verification for multiple relations in a simple and intuitive manner posed as another objective of this project. To achieve this, the previously disjunct spectroscopy properties were to be joined into one property and needed to provide some form of visualization for both its strong and weak version.

Goal for the spectroscopy energy game was shifting the move selection from the list on the bottom right to the displayed process-graph by allowing users to interact directly with edges and vertices. This would eliminate the need to regularly compare listed game moves to the graph view resulting in a more intuitive understanding of the game and an overall improved user experience. Additionally, the current game state representation using two graphs was to be merged into one singular graph to further simplify the game's representation.

3.3 Results

The implementation of the vision presented in the previous chapter can be found under `https://equivio.github.io/CAAL/`. Modifications to the web-tool are focused on the verification and spectroscopy energy game pages.

The verify-page was mainly revised in regard to the spectroscopy-verification. Verification of all relations contained in the adapted linear-time–branching-time spectrum from figure 3 has been implemented and can be accessed individually by adding corresponding properties as before. This increases the number of relations users can verify using CAAL from 16 to 35. The property selection for simultaneous verification of multiple relations using the spectroscopy has been moved and is no longer found in the drop down menu next to other individual relations under the term "BJN-Algorithm". It can now be accessed via a dedicated area in the property creation next to individual verification and model checking as seen in figure 8. There, users can either add a verification property for the initially existing spectroscopy for strong relations or use its weak counterpart which has been implemented and is available for verification. While the generation of distinguishing formulas is still supported for the strong spectroscopy, the weak version only possesses a framework for this functionality and currently generates the universally true property as a placeholder.

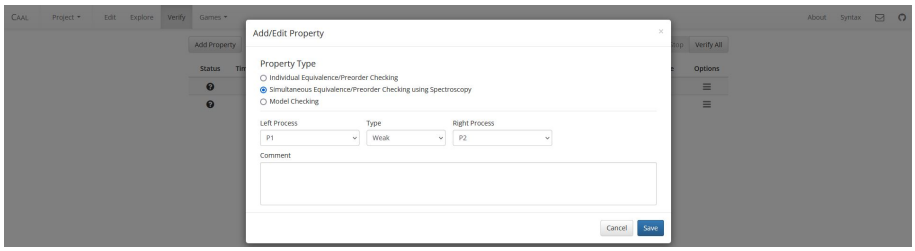

Figure 8: Spectroscopy property creation

Spectroscopy properties are now implemented as singular properties in contrast to the addition of multiple properties, one for each relation featured in the spectroscopy version. This allows for the property to visually represent the hierarchy of included relations by piling them up similar to bricks of a wall in accordance with the adapted linear-time–branching-time spectrum. Additionally, this change eliminated the need for a dedicated button for spectroscopy verification since their contained relations are no longer split across

multiple properties. Upon verification of the property, every applying relation is highlighted in green, while others are highlighted in red. Due to the hierarchical structure of the visualization, not applying relations will never have applying relations "stacked" on top of them. This coloring of the hierarchy depiction replaces the icon in the status column used for individual properties. An example of this practice can be found in figure 9. To prevent unwanted behavior of the new weak spectroscopy verification, test cases covering its verifying functionality have been implemented. The next chapter 2.4 Technical Aspects will present these in greater detail.

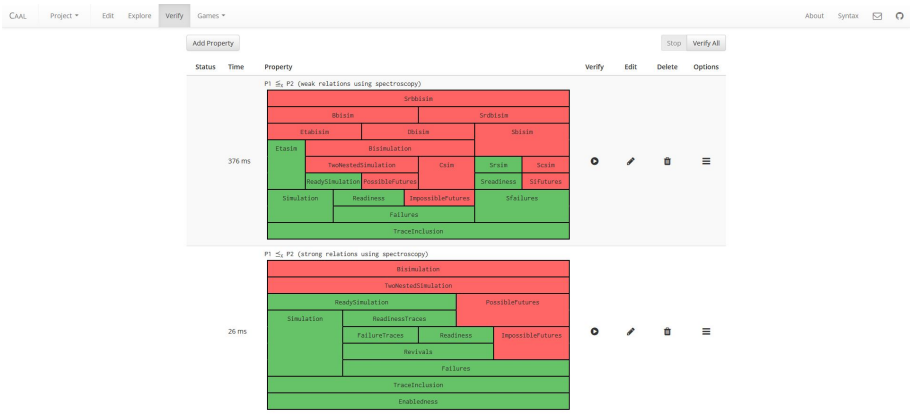

Figure 9: Exemplary spectroscopy properties

Verified properties for the strong spectroscopy allow users to switch to the spectroscopy energy game page. Since there are no longer individual properties for every contained relation, using this functionality loads the game page with all corresponding options with the relation set to bisimulation. The spectroscopy energy game page hosts the strong spectroscopy energy game, which is similar to the weak spectroscopy energy game but is designed for strong relations. As before, the top of the page is dedicated to adjustable options and game specific information. Here, users can adjust the relation, choose processes, decide to play as the attacker or defender and restart the game. Additionally, the currently remaining energy budget is displayed here. Similar to the weak spectroscopy energy game the available budget at the start of a game is determined by the chosen relation, but differing from its weak version, the strong spectroscopy energy game is defined with six dimensions for energies. Divergent from the

previous version of the game page, users are presented with a singular game graph in the middle of the screen. The lower part of the page is reserved for the game log, which behaves as before, logging player moves, moves of the opponent, the current game position and the result of a game once one side wins. The move table which was located on the bottom right has been removed. Instead of relying on a table, users can now make moves by interacting with the displayed game graph and an additional "Confirm Challenge" button.

To present the state identification and move selection for the strong spectroscopy game using the overhauled game view, we will first present a rough outline of said game. A detailed introduction can be found under [2]. In contrast to its weak version, there are only three game positions for the strong spectroscopy energy game:

- general attacker position $(p, Q)_A$
- attacker clause position $(p, q)^\wedge_A$
- defender conjunction position $(p, Q, Q_*)_D$

The components of these positions are marked using the following color-scheme in the graph: Processes not reached by the depth specified at the top right of the graph canvas are greyed out and all those reached but not contained in the current game state are blue. For all three game positions, processes labeled with a p are highlighted in red. All $q \in Q$ of general attacker positions and defender conjunction positions are colored green. The process q of attacker conjunction positions is dyed teal and all $q_* \in Q_*$ of defender conjunction positions are purple.

This coloring lets players identify the three game states with more intuitively: While playing the attacking side, a teal-colored process infers an attacker clause position, while its absence infers a general attacker position. Since the defending side only has one position it does not need to be uniquely distinguishable.

The strong spectroscopy energy game consists of a total of six game moves:

- observations $(p, Q)_A \overset{(-1,0,0,0,0,0)}{\rightarrowtail} (p', Q')_A$,
  with $p \xrightarrow{a} p', Q' = \{q' \in P | \exists q \in Q . q \xrightarrow{a} q'\}$
- conjunction challenges $(p, Q)_A \overset{(0,-1,0,0,0,0)}{\rightarrowtail} (p, Q \backslash Q_*, Q_*)_D$, with $Q_* \subseteq Q$
- conjunction revivals $(p, Q, Q_*)_D \overset{(min_{\{1,3\}},0,0,0,0,0)}{\rightarrowtail} (p, Q_*)_A$, with $Q_* \neq \varnothing$
- conjunction answers $(p, Q, Q_*)_D \overset{(0,0,0,min_{\{3,4\}},0,0)}{\rightarrowtail} (p, q)^\wedge_A$, with $q \in Q$

- positive decisions $(p,q)^{\wedge}_{A} \overset{(min_{\{1,4\}},0,0,0,0,0)}{\rightarrowtail} (p,\{q\})_A$
- negative decisions $(p,q)^{\wedge}_{A} \overset{(min_{\{1,5\}},0,0,0,0,-1)}{\rightarrowtail} (q,\{p\})_A$, with $p \neq q$

Playing the attacking side, players can choose one of two moves if they find themself in a general attacker position: Either an observation move can be performed by clicking on a viable action label, or players can choose to initiate a conjunction challenge. This is done by selecting those processes of Q, which the player wants to move to Q_*, by clicking on the respective node of the graph. Selecting a process of Q in this manner turns the process purple and clicking on it again reverses this selection. After the player is satisfied with their selection, the conjunction challenge move can be executed by pressing the corresponding "Confirm Challenge"-button on the top right of the graph canvas. From an attacker clause position players can proceed by selecting either the process currently marked as p to make a positive decision move or by selecting the process marked as q to make a negative decision move. The selected process becomes the process p of the following general attacker position. The defending side can select any process $q_* \in Q_*$ to carry out a conjunction revival move or select the specific process $q \in Q$ for a conjunction answer move.

To further help users intuitively maneuver the spectroscopy energy game, hovering over a move highlights the changes to the remaining energy budget it would entail in red.

3.4 Technical Aspects

The codebase of the extended version of CAAL found under `https://github.com/Fabian-001/CAAL`, which serves as a base for this project, is mainly implemented using the high level programming languages TypeScript and JavaScript. Additionally, HTML and CSS are used to create the user interface of CAAL. The current up-to-date version of the code for CAAL can be accessed under `https://github.com/equivio/CAAL`.

Notable parts of the implementation are found under the "src" and "test" directories, the latter mainly containing quality assurance utilities. The "src" directory holds the main functionality of CAAL and will now be briefly presented. The file "main.ts", found directly under "src" is initially used to create the different pages found on the CAAL web-page. It calls corresponding functions from the "activityhandler.ts" file, contained in "src/activity", where many administrating parts of the code can be found. Said handler file im-

plements functions to add and navigate the pages, which each have their own classes in respective files, inheriting from a general activity class. Since this project's focus were the addition of the weak spectroscopy for verification purposes and an overhaul to the game-view for the spectroscopy energy game, changes under the "activity" directory are primarily found in the "verifier.ts" and "segame.ts" files. While the game has mainly been altered in regards to UI and some underlying structural adjustments to the code, the verifier page has been expanded in UI, structure and functionality. Code regarding the properties containing relations for verification can be found under the "src/gui/property.ts" file. Here, new property types for all relations of the weak spectroscopy have been added. Additionally a new property for simultaneous verification using spectroscopies was implemented, replacing the previously utilized practice of adding an individual property for each required relation. Old and new properties are verified using corresponding functions in the "verifier.ts" file of the "src/workers" directory. The verification of spectroscopy relations is based on the algorithm for computing minimal attacker winning budgets presented in 1. It is implemented along the general game graph creation and other functionalities in both the "weak-spectroscopy.ts"- and "strong-spectroscopy.ts" files under the "src/spectroscopy" directory for the strong and weak spectroscopy versions respectively. In these files, the computed minimal attacker winning budgets are also used to derive applying relations for the respective spectroscopy version. Additionally, code covering spectroscopy-specific functionality has been adjusted to work directly with CAAL's basic datastructures if possible. This allowed for some redundant data conversions to be removed from the code, resulting in an overall better optimized codebase.

The "integration" directory, found in the "test" directory contains files corresponding to different functionalities of CAAL, which implement tests used for quality assurance. Since the already existing files for bisimulation, model checking and traces remained unchanged and did not contribute much to this projects result, they will not be covered in greater detail. The new addition to the test files "weakSpectroscopyTest.js" contains several test cases that have been created partially based on the cases found under `https://equiv.io/main/` to ensure correct behavior of implemented code for the weak spectroscopy and consequently for utilities of the strong spectroscopy used in the weak version. Per case two independent tests have been implemented: One checking the calculated budgets that are needed to win given a set of processes and the other testing the applying relations given a set of budgets. These tests helped in dis-

covering unwanted behavior during the implementation process and thus played a big part in the creation of the current version of CAAL. A simple example for such unwanted behavior was altering elements of a data structure that was being iterated over during the iteration itself. The most notable bug found due to the constructed test cases was located in the verification algorithm for spectroscopies. While the algorithm needed to determine all possible combinations of a set of sets containing exactly one element from each set in order to determine attacker winning budgets from defender positions, the implementation only supported this operation for at most two sets, thus creating a hard to spot bug impacting both strong and weak spectroscopies, that remained overlooked in the codebase used as a basis of this project.

4 Conclusion

In conclusion, this project's focus was put on the extension of CAAL's verification functionality and the UI of the game view for the strong spectroscopy energy game. The implementation of the weak spectroscopy version and its integration into the verifier was successfully accomplished. Entailed changes to some underlying datastructures used in both spectroscopy versions enhanced code efficiency by making some previously needed conversions from CAAL's original datastructures redundant. Additionally, the UI of relation verification using spectroscopy has been overhauled by clearly separating it from individual verifications during creation and by grouping contained relations into a single property where they are displayed using a visual hierarchy to increase clarity. To ensure the correctness of the new functionality, CAAL's existing testcases have been expanded to include quality assurance of the weak spectroscopy and thus some functions of the strong version. Due to this expansion, some bugs in the codebase serving as basis for this project could be found and removed. The spectroscopy energy game's UI has been overhauled to improve the user experience in exchange for less uniformity among all of CAAL's games. This change entailed displaying the entire game on one graph and revising the manner in which moves are selected. Instead of using a disconnected table for move selection, players can now directly interact with the graph. Additionally, updates to the energy budet corresponding to selected moves are highlighted, further increaseing the intuitivity of the game.

Some extensions that would further increase CAAL's functionality and user

experience are the implementation of the weak spectroscopy energy game into the game view, to allow users to better understand how it is used for verification, or the creation of distingushing formulas for the weak spectroscopy in the verifier. Additionally, users could benefit from an introduction to the spectroscopy energy game on the corresponding game page or a legend covering information such as the correlation of game positions and the coloring of the graph.

While CAAL would certainly benefit from these extensions, every tool can be enhanced almost indefinetly. The version created in this project improves its predecessor by eliminating some contained bugs and expanding the verification capabilities siginificantly, while improving the UI of both the verification view and the spectroscopy energy game view.

References

[1] Benjamin Bisping, David N. Jansen, and Uwe Nestmann. "Deciding All Behavioral Equivalences at Once: A Game for Linear-Time–Branching-Time Spectroscopy". In: *Logical Methods in Computer Science* Volume 18, Issue 3 (Aug. 2022). ISSN: 1860-5974. DOI: 10.46298/lmcs-18(3:19)2022. URL: http://dx.doi.org/10.46298/lmcs-18(3:19)2022.

[2] Benjamin Bisping. "Process Equivalence Problems as Energy Games". In: *Computer Aided Verification*. Springer Nature Switzerland, 2023, pp. 85–106. DOI: 10.1007/978-3-031-37706-8_5.

[3] Benjamin Bisping and David N. Jansen. *Linear-Time–Branching-Time Spectroscopy Accounting for Silent Steps*. 2023. arXiv: 2305.17671 [cs.LO].

[4] Fabian Ozegowski. *Integration eines generischen Äquivalenzprüfers in CAAL*. 2023. URL: https://github.com/Fabian-001/CAAL/blob/spectroscopy/docs/CAAL_Spectroscopy_Thesis.pdf.

[5] Matthew Hennessy and Robin Milner. "On observing nondeterminism and concurrency". In: *Automata, Languages and Programming*. Ed. by Jaco de Bakker and Jan van Leeuwen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1980, pp. 299–309. ISBN: 978-3-540-39346-7. DOI: 10.1007/3-540-10003-2_79.

[6] R.J. van Glabbeek. “The Linear Time - Branching Time Spectrum I.” In: *Handbook of Process Algebra*. Ed. by J.A. Bergstra, A. Ponse, and S.A. Smolka. Amsterdam: Elsevier Science, 2001, pp. 3–99. ISBN: 978-0-444-82830-9. DOI: 10.1016/B978-044482830-9/50019-9.

Appendices

A CAAL Tutorial

A.1 Verification of Equivalences/Preorders

CAAL supports the verification of relations presented in table 2.

supported relation	strong	weak	equivalence checking
bisimulation	✓	✓	
two-nested simulation	✓	✓	
ready simulation	✓	✓	
possible futures	✓	✓	
simulation	✓	✓	✓
readiness traces	✓		
failure traces	✓		
readiness	✓	✓	
impossible futures	✓	✓	
revivals	✓		
failures	✓	✓	
trace inclusion	✓	✓	✓
enabledness	✓		
stability-respecting branching bisimulation		✓	
branching bisimulation		✓	
stabiliy-respecting delay bisimulation		✓	
η-bisimulation		✓	
delay bisimulation		✓	
stable bisimulation		✓	
stable ready simulation		✓	
stable contrasimulation		✓	
stable readiness		✓	
stable impossible futures		✓	
stable failures		✓	
η-simulation		✓	
contrasimulation		✓	

Table 2: Supported relations of CAAL

In order to verify relations from the table, users first have to select the verifier page. Here, a new property can be added using the "Add Property"-button on the top left of the page. This action opens the property selection pop-up depicted in figure 10, where users can select the type of property they plan to verify. Choosing the "Individual Equivalence/Preorder Checking"-option allows the adjustment of four different verification components: A relation from table 2, its type and both the left and right process, for which the relation will be checked. For relations, which are supported by both preorder and equivalence checking, two relations are listed: For example, "Simulation" and "Simulation

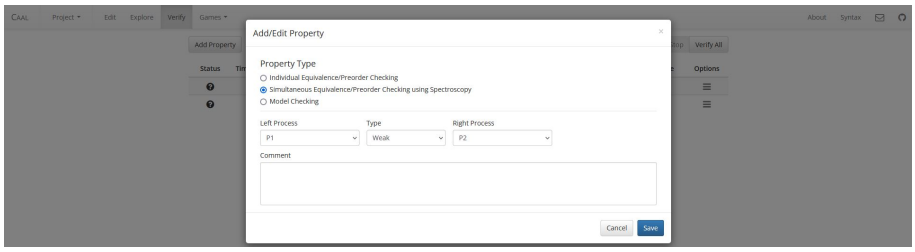

Figure 10: Property selection pop-up

Equivalence". After the desired configuration has been made, the property can be stored by selecting the "Save"-button. This adds the property to the property list, where the verification can be computed. To change settings of a property, users can select the pencil icon in the edit column of a property. Initially, individual equivalence/preorder properties are marked with a questionmark in the status colum, indicating that they have not been verified. The type of relation and the chosen processes are displayed in the property column. Verification for a single property can be carried out by pressing the icon in its verify column. Alternatively, all properties currently stored in the property list can be verified sequentially by pressing the "Verify All"-button at the top right of the page. After successful verification of a individual property, the icon in its status column will be updated with a red checkmark if the respective relation holds, or with a red cross if it does not. Additionally, the time column will display the elapsed time during verificaion for a given property. For verified properties, users can select supported options such as generating distinguishing formulas or playing games.

Instead of choosing the individual option for equivalence/preorder checking, users can select the "Simultaneous Equivalence/Preorder Checking using Spectroscopy"-option in the property selection pop-up. This results in the selection of three verification components: Both the left and right process, on which relations will be checked, and the type of spectroscopy which will be used. Again, the "Save"-button adds the property to the list. Here, the processes and type of spectroscopy are indicated in the property column, aswell as a hierarchical visualization of all relations covered by the respective spectroscopy. Instead of using the status column to display verification status, properties for simultaneous verification display this information by highlighting the relations in their visualization: White for unverified, red for not applying relations and green for applying relations. While the strong version of the spectroscopy suports the

generation of distinguishing formulas, the weak version currently generates true as a placeholder. Figure 11 depicts an example for each spectroscopy property.

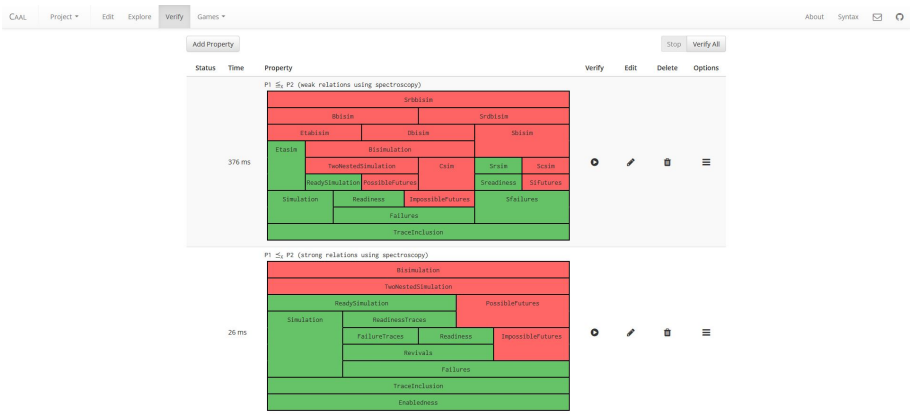

Figure 11: Exemplary spectroscopy properties

A.2 Strong Spectroscopy Energy Game

CAAL supports the spectroscopy energy game in its strong variant. The game consists of an attacker, a defender, two processes p and q and an initial energy budget determined by the relation specified along with the other parameters at the start of the game. The attacker's goal is to force the defender into a position where they are stuck, i.e. unable to perfom another move, while the defender's goal is to deplete the energy budget or force a cycle of game positions. As opposed to the bisimulation game, player turns are not necessarily alternating as there are attacker moves that lead to attacker positions once again. The different types of positions are

- general attacker positions $(p, Q)_A$,
- attacker clause positions $(p, q)^{\wedge}_A$ and
- defender conjunction positions $(p, Q, Q_*)_D$.

The graph nodes representing processes are colored depending on their affiliation in the curent game position according to the following rules:

- p is colored red,
- $q \in Q$ are colored in green

- $q_* \in Q_*$ are colored in purple and
- a q from an attacker clause position is colored in teal.

The type of position determines, which types of moves are possible. The types of moves are

- observations $(p, Q)_A \overset{(-1,0,0,0,0,0)}{\rightarrowtail} (p', Q')_A$,
  with $p \xrightarrow{a} p', Q' = \{q' \in P | \exists q \in Q . q \xrightarrow{a} q'\}$,
- conjunction challenges $(p, Q)_A \overset{(0,-1,0,0,0,0)}{\rightarrowtail} (p, Q \backslash Q_*, Q_*)_D$, with $Q_* \subseteq Q$,
- conjunction revivals $(p, Q, Q_*)_D \overset{(min_{\{1,3\}},0,0,0,0,0)}{\rightarrowtail} (p, Q_*)_A$, with $Q_* \neq \varnothing$,
- conjunction answers $(p, Q, Q_*)_D \overset{(0,0,0,min_{\{3,4\}},0,0)}{\rightarrowtail} (p, q)^\wedge_A$, with $q \in Q$,
- positive decisions $(p, q)^\wedge_A \overset{(min_{\{1,4\}},0,0,0,0,0)}{\rightarrowtail} (p, \{q\})_A$ and
- negative decisions $(p, q)^\wedge_A \overset{(min_{\{1,5\}},0,0,0,0,-1)}{\rightarrowtail} (q, \{p\})_A$, with $p \neq q$.

Moves are selected by clicking on elements of the graph. In detail, this works as follows:

- For observations, by clicking on an action label a of an outgoing edge of p,
- for conjunction challenges, by clicking on all $q \in Q$ that are supposed to become elements of Q_*, coloring them purple in the process to signal the provisional state of selection. Then, using the "confirm challenge"-button to confirm the selection,
- for conjunction revivals, by clicking on any $q_* \in Q_*$,
- for conjunction answers, by clicking on the desired $q \in Q$,
- for positive decisions, by clicking on p and
- for negative decisions, by clicking on q.

Hovering over a valid move decision causes the energy gauge to show the resulting budget, indicating changes of dimensions in red.

Winning as an attacker means that we have found a formula that distinguishes p from q with not more "distinguishing power" than allowed for the selected relation, meaning that p and q cannot be in said relation. Consequently, winning as the defender proves that p and q are equal in the selected context of a relation.