Technische Universität Berlin

Fakultät IV - Elektrotechnik und Informatik

Modelle und Theorie Verteilter Systeme

Bachelorarbeit

Integration eines generischen Äquivalenzprüfers in CAAL

Fabian Magnus Ozegowski

Matrikelnummer: 413502

11.10.2023

Prüfende

Prof. Dr.-Ing. Uwe Nestmann

Prof. Dr. Stephan Kreutzer

Betreuer

Benjamin Bisping

Zusammenfassung

Das Prinzip der Spectroscopy erlaubt es, viele Gleichheitsbegriffe für Transitionssysteme gleichzeitig mit einer Berechnung zu verifizieren, indem die Ausdrucksstärke der unterscheidenden Formeln für die betreffenden Prozesse analysiert wird. In dieser Arbeit wird das Spectroscopy Energy Game und der zugehörige Algorithmus zur Lösung dieses Spiels in TypeScript implementiert und in CAAL, eine Webapplikation zur Modellierung, Visualisierung und Verifikation von Prozessen, integriert.

Inhaltsverzeichnis

1 Einleitung

In der Concurrency Theory beschäftigt sich ein Teilgebiet mit der Frage, wie „gleich“ zwei Prozesse sind. Jedoch existiert keine einheitliche Version von Gleichheit im klassischen Sinne, vielmehr lassen sich Prozesse auf diverse Kriterien untersuchen. Ist ein Kriterium zwischen zwei Prozessen erfüllt, so können diese Prozesse als gleich in Bezug auf jenes Kriterium angesehen werden. Diese Kriterien können zwar willkürlich definiert und gewählt werden, im Laufe der Forschungsgeschichte hat sich jedoch eine Menge an Kriterien herausgebildet, die jeweils für spezifische Anwendungsfälle relevant sind. Weiterhin wurden individuell für diese Anwendungsfälle Verfahren zur Verfikation entwickelt. Aus der Effizienzperspektive ist es hierbei nachteilig, für jeden Gleichheitsbegriff jeweils ein spezifisch dafür konzipiertes Verifikationsverfahren verwenden zu müssen. In [1] wird für dieses Problem ein allgemein anwendbarer Algorithmus vorgestellt, der viele der relevanten Gleichheitsbegriffe in einer Berechnung verifiziert. Dieser Algorithmus löst ein eigens für dieses Problem konzipiertes Logikspiel.

Im Rahmen dieser Arbeit wurde sowohl der eben erwähnte Algorithmus zur Verifikation als auch das Logikspiel implementiert und in eine bereits bestehende Webapplikation namens CAAL integriert. CAAL steht für Concurrency Workbench, Aalborg Edition und ist ein von drei Studierenden als Teil ihrer Masterarbeit entwickeltes Tool zur Modellierung, Visualisierung und Verifikation von Prozessen, welches seither zu Lehrzwecken in Verwendung ist. Die theoretische Basis der Webapplikation bildet das Lehrbuch „Reactive Systems“ [2]. Eine Erweiterung des Tools durch die Ergebnisse von [1] ist sinnvoll, da dadurch einerseits das Verifizieren sowohl effizienter als auch komfortabler ist und andererseits die Erweiterung den logischen nächsten Schritt in der Entwicklung von Verifikationsalgorithmen darstellt und es somit wichtig ist, für die edukativen Zwecke des Tools den Lernenden Zugang hierzu zu ermöglichen. Die Erweiterung ist unter `https://fabian-o01.github.io/CAAL/` abrufbar.

In den Kapiteln 2 bis 7 wird sich mit der Theorie beschäftigt, die zum Verständnis der Funktionsweise des Algorithmus sowie CAAL notwendig ist. Zuerst wird dafür auf die grundlegenden Konzepte der Prozesstheorie eingegangen und im Anschluss ein Überblick über eine Auswahl an Gleichheitsbegriffen gegeben. Da das erwähnte Logikspiel ein Energiespiel ist, wird auch dieses Konzept erläutert, um daraufhin vorzustellen, wie für diesen spezifischen Fall das Energiespiel definiert ist. Abschließend wird sich mit dem implementierten Algorithmus und der Methode beschäftigt, wie das Ergebnis dessen interpretiert

werden kann, um auf Gleichheitsbegriffe zu schließen.

Kapitel 8 beschäftigt sich mit dem praktischen Teil dieser Arbeit und stellt vor, welche Änderungen im Detail an CAAL vorgenommen und welche Designüberlegungen dafür angestellt wurden. Es folgt ein Überblick zur Code-Architektur der Applikation.

Kapitel 9 schließt mit einem Fazit die Arbeit ab.

2 Systemspezifikation mit LTS und CCS

Um Eigenschaften von Prozessen zu analysieren, benötigen wir grundlegend eine Struktur, mit der die Modellierung dieser Prozesse möglich ist. CAAL verwendet zu diesem Zweck Labeled Transition Systems.

Definition 1 (Labeled Transition System)**.**
Ein Labeled Transition System (LTS) ist ein Tupel $(\mathcal{P}, \mathcal{A}, \rightarrow)$, wobei $\mathcal{P}$ eine Menge von Prozessen, $\mathcal{A}$ eine Menge von Aktionen und $\rightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ eine Transitionsrelation ist.

Wenn ein Prozess $p \in \mathcal{P}$ eine Aktion $a \in \mathcal{A}$ ausführen kann, welche in Prozess $p' \in \mathcal{P}$ resultiert, schreiben wir $p \xrightarrow{a} p'$.

Mit $I(p) \subseteq \mathcal{A}$ bezeichnen wir die Menge aller Aktionen, die von einem Prozess $p \in \mathcal{P}$ initial direkt ausführbar sind, also $I(p) \coloneqq \{a \in \mathcal{A} \mid \exists p'.p \xrightarrow{a} p'\}$.

Der in [3] vorgestellte Calculus of Communicating Systems bietet einen Rahmen, um Prozesse zu definieren. Im Folgenden definieren wir die Sprache, die hierfür verwendet wird. Dabei gilt, dass A eine Menge an Namen und $\overline{A} \coloneqq \{\overline{a} \mid a \in A\}$ die dazu korrespondierende Menge an Co-Namen ist. Weiterhin gilt, dass $A \cup \overline{A} \cup \{\tau\} = \mathcal{A}$.

Definition 2 (CCS Syntax)**.**
Die Menge $\mathcal{P}$ von Calculus-of-Communicating-Systems (CCS)-Ausdrücken ist durch die folgende Backus-Naur-Grammatik definiert:

$$p_1 ::= 0 \;\Big|\; B \;\Big|\; a.p_1 \;\Big|\; p_1 + p_2 \;\Big|\; p_1|p_2 \;\Big|\; p_1[f] \;\Big|\; p_1 \backslash A_{\subseteq}$$

Es gilt:

- 0 ist der inaktive Nullprozess
- B ist ein Bezeichner eines Prozesses

- $a \in \mathcal{A}$ ist eine Aktion
- $f : \mathcal{A} \to \mathcal{A}$ ist eine Funktion zur Umbenennung von Namen für die gilt:
  - $f(\tau) = \tau$
  - $f(\overline{a}) = \overline{f(a)}$
  - $\overline{\tau} = \tau$
- $A_{\subseteq} \subseteq A$.

Für jeden Bezeichner B existiert eine Definition $B \stackrel{\text{def}}{=} p$, wobei p auch B selbst enthalten kann und somit rekursive Definitionen erlaubt.

Die Bindungsstärke der Operatoren verhält sich in absteigender Reihenfolge wie folgt:

- Umbenennung $p[f]$ und Restriktion $p\backslash A$,
- Aktionspräfix $a.p$,
- Nebenläufigkeit $p_1|p_2$ und
- Summation $p_1 + p_2$

Definition 3 (CCS Semantik)**.**
Die Semantik von CCS ist durch folgende Ableitungsregeln definiert:

$$\text{CON}\ \frac{p \xrightarrow{a} p'}{B \xrightarrow{a} p'} \quad \text{für } B \stackrel{\text{def}}{=} p \qquad \text{ACT}\ \frac{}{a.p \xrightarrow{a} p}$$

$$\text{SUM1}\ \frac{p_1 \xrightarrow{a} p_1'}{p_1 + p_2 \xrightarrow{a} p_1'} \qquad \text{SUM2}\ \frac{p_2 \xrightarrow{a} p_2'}{p_1 + p_2 \xrightarrow{a} p_2'}$$

$$\text{COM1}\ \frac{p_1 \xrightarrow{a} p_1'}{p_1|p_2 \xrightarrow{a} p_1'|p_2} \qquad \text{COM2}\ \frac{p_2 \xrightarrow{a} p_2'}{p_1|p_2 \xrightarrow{a} p_1|p_2'}$$

$$\text{COM3}\ \frac{p_1 \xrightarrow{a} p_1' \quad p_2 \xrightarrow{\overline{a}} p_2'}{p_1|p_2 \xrightarrow{\tau} p_1'|p_2'} \quad \text{für } a, \overline{a} \neq \tau \qquad \text{REL}\ \frac{p \xrightarrow{a} p'}{p[f] \xrightarrow{f(a)} p'[f]}$$

$$\text{RES}\ \frac{p \xrightarrow{a} p'}{p\backslash A_{\subseteq} \xrightarrow{a} p'\backslash A_{\subseteq}} \quad \text{für } a, \overline{a} \notin A_{\subseteq}$$

Es gilt $a \in \mathcal{A}$.

Beispiel 1 (CCS)**.**
Schauen wir uns nun ein Beispiel an, in dem wir einen Alltagsprozess in CCS modellieren. Stellen wir uns einen Schallplattenspieler vor:

$$\text{Plattenspieler} \stackrel{\text{def}}{=} \text{abspielen}.0$$

Damit haben wir einen Prozess mit dem Namen „Plattenspieler“ definiert, der die Aktion „abspielen“ ausführt und dann in den Nullprozess übergeht, der per Definition keine Aktionen mehr ausführen kann.

Der Gebrauch von Prozessnamen ermöglicht auch rekursive Definitionen:

$$\text{Plattenspieler} \stackrel{\text{def}}{=} \text{abpielen}.\text{Plattenspieler}$$

Mit dieser Definition kann der Prozess die Aktion „abspielen“ ausführen und dann wieder zum Prozess „Plattenspieler“ werden.

Zu einem willkürlichen Zeitpunkt kann es passieren, dass der Plattenspieler hängen bleibt. Dies lässt sich mit dem Summationsoperator + darstellen:

$$\text{Plattenspieler} \stackrel{\text{def}}{=} \text{abspielen}.(\text{Plattenspieler} + \overline{\text{hängen}}.0)$$

Nun verhält sich der Prozess nach dem Abspielen entweder rekursiv wie der Plattenspieler oder er hängt sich auf.

Fügen wir nun zu diesem Modell eine zuhörende Person hinzu, die das Hängen der Schallplatte wieder beheben kann:

$$\text{Person} \stackrel{\text{def}}{=} \text{hängen}.\overline{\text{beheben}}.0$$

Da nun das Hängen nicht mehr zur Annahme des Nullprozesses führt, gilt nun für den Plattenspieler

$$\text{Plattenspieler} \stackrel{\text{def}}{=} \text{abspielen}.(\text{Plattenspieler} + \overline{\text{hängen}}.\text{beheben}.\text{Plattenspieler})\ .$$

Der Plattenspieler und die Person sind bisher noch unassoziiert bzw. ihre Aktionen werden unabhängig voneinander ausgeführt. Damit sie nebenläufig zueinander ausgeführt werden, brauchen wir den Nebenläufigkeits- bzw. Parallelkompositionsoperator.

$$\text{Musiksession} \stackrel{\text{def}}{=} \text{Plattenspieler} \mid \text{Person}$$

Der Plattenspieler und die Person teilen sich nun die beiden Kanäle „hängen“ und „beheben“ zur Kommunikation. Die Co-Aktionen „$\overline{\text{hängen}}$“ und „$\overline{\text{beheben}}$“ sind hierbei als Outputs und deren Pendants als Inputs zu interpretieren. Die Kommunikation der beiden Prozesse ist intern und somit von außen nicht zu beobachten. Die daraus resultierende Aktion bezeichnen wir als τ. Es ist jedoch nach wie vor nicht ausgeschlossen, dass beide Prozesse weiterhin unabhängig voneinander laufen. Um zu gewährleisten, dass Kommunikation stattfindet, können wir daher den Restriktionsoperator verwenden:

$$\text{Musiksession} \stackrel{\text{def}}{=} (\text{Plattenspieler} \mid \text{Person}) \setminus \{\text{hängen,beheben}\}$$

Nun ist die Kommunikation über „hängen“ und „beheben“ eingeschränkt, sodass Kommunikation zwischen den Prozessen erzwungen wird, was intern durch die τ-Aktion geschieht.

Das Beispiel lässt sich in CAAL unter „Plattenspieler Example“ betrachten.

3 Beschreibung von Systemverhalten mit HML

Um nun Aussagen über die durch CCS definierten Prozesse zu treffen, benutzen wir die Hennessy-Milner-Logik (HML). Diese ist eine Erweiterung der klassischen Aussagenlogik durch eine Modaloperation.

Definition 4 (Syntax Hennessy-Milner-Logik)**.**
Die Syntax der Hennessy-Milner-Logik über eine Menge $\mathcal{A}$ an Aktionen ist durch folgende Grammatik für HML[$\mathcal{A}$] induktiv definiert:

$$\begin{array}{lll} \varphi \coloneqq & \langle a \rangle \varphi & \mid \quad \bigwedge_{i \in I \subseteq \mathbb{N}} \psi_i \qquad \text{mit } a \in \mathcal{A} \\ \psi \coloneqq & \varphi & \mid \quad \neg\varphi \end{array}$$

Definition 5 (Semantik Hennessy-Milner-Logik)**.**
Sei $T = (\mathcal{P}, \mathcal{A}, \rightarrow)$ ein LTS und $p \in \mathcal{P}$. Die Erfüllbarkeitsrelation $\models : (\mathcal{P}, \text{HML}[\mathcal{A}])$ ist definiert durch:

$$\begin{array}{lll} p & \models & \langle a \rangle \varphi \qquad \text{falls } \exists p'.p' \models \varphi \wedge p \xrightarrow{a} p' \\ p & \models & \neg\varphi \qquad \text{falls } p \not\models \varphi \\ p & \models & \bigwedge_{i \in I \subseteq \mathbb{N}} \{\psi_i\} \quad \text{falls } \forall i \in I.p \models \psi_i \end{array}$$

Hierbei ist der spezielle Fall

$$\bigwedge_{i \in \varnothing} \{\psi_i\} = \bigwedge \varnothing$$

zu beachten, der aufgrund der Eigenschaft der leeren Wahrheit des Allquantors als die universell wahre Konstante $\top$ benutzt werden kann.

Beispiel 2 (Hennessy-Milner-Logik)**.**
Für dieses Beispiel benutzen wir ein LTS, das mit folgendem CCS definiert ist:

$$P_1 = a.b.0$$
$$P_2 = a.b.0 + a.c.0$$

Das zugehörige LTS sieht folgendermaßen aus:

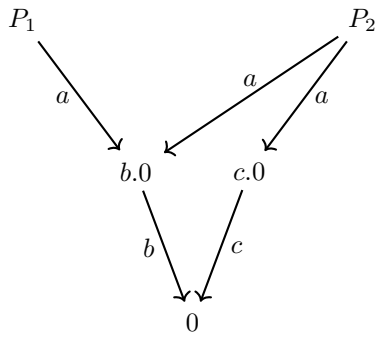

Abbildung 1: Beispiel eines LTS in Graphschreibweise

Nun können wir Aussagen über dieses LTS anhand von HML-Ausdrücken tätigen. Die Formel $\varphi_1 \coloneqq \langle a \rangle \langle b \rangle \bigwedge \varnothing$ formalisiert die Aussage „Es existiert eine Aktion a, sodass eine Aktion b möglich ist.“ Daher gilt $P_1 \models \varphi_1$ und $P_2 \models \varphi_1$. Für Formel $\varphi_2 \coloneqq \langle a \rangle \langle c \rangle \bigwedge \varnothing$ gilt $P_2 \models \varphi_2$, aber $P_1 \not\models \varphi_2$, da von P_1 keine Aktion a ausgeführt werden kann, sodass eine Aktion c möglich ist, von P_2 jedoch schon. In anderer Richtung dazu gilt folglich für die Negation von φ_2, $\varphi_{2\neg} = \bigwedge \{\neg \langle a \rangle \langle c \rangle \bigwedge \varnothing\}$, dass $P_1 \models \varphi_{2\neg}$, aber $P_2 \not\models \varphi_{2\neg}$. Daher sind φ_2 und $\varphi_{2\neg}$ sogenannte „Distinguishing Formulas“, also unterscheidende Formeln, da sie P_2 von P_1 bzw. P_1 von P_2 unterscheiden. In Kapitel 6 werden wir uns genauer mit solchen Formeln beschäftigen. Das Beispiel inklusive Queries zu den HML-Formeln ist in CAAL unter „HML Example“ zu finden.

4 Gleichheitsbegriffe zwischen Prozessen

4.1 Klassen von Gleichheitsbegriffen

Bevor die spezifischen Gleichheitsbegriffe vorgestellt werden können, muss zuerst auf die Klassen dieser Relationen eingegangen werden. Wir unterscheiden hierbei in der Ordnungstheorie zwischen Präordungen und Äquivalenzrelationen.

Definition 6 (Präordnung)**.**
Eine Präordnung ist eine binäre Relation R über einer Menge M, die reflexiv und transitiv ist. Für alle $a, b, c \in M$ muss also gelten:

- $(a, a) \in R$ (Reflexivität)
- $(a, b) \in R \wedge (b, c) \in R \longrightarrow (a, c) \in R$ (Transitivität)

Definition 7 (Äquivalenzrelation)**.**
Eine Äquivalenzrelation ist eine binäre Relation R über einer Menge M, die reflexiv, transitiv und symmetrisch ist. Symmetrie beschreibt hierbei die Eigenschaft $(a, b) \in R \longrightarrow (b, a) \in R$ für $a, b \in M$.

Daraus folgt, dass die Menge der Äquivalenzrelationen eine Teilmenge der Menge der Präordnungen ist und somit jede Äquivalenzrelation auch eine Präordnung sein muss.

Für die im Folgenden vorgestellten Relationen ist zu beachten, dass trotz der Verwendung des Terminus „Gleichheitsbegriff“ einige dieser Relationen nicht symmetrisch sind und daher nur Präordnungen darstellen, die einseitig gelten.

4.2 Trace-Semantiken

Sei $T = (\mathcal{P}, \mathcal{A}, \rightarrow)$ ein LTS.

Definition 8 (Traces)**.**
Seien $p_0, ..., p_n \in \mathcal{P}, a_1, ..., a_n \in \mathcal{A}, n \in \mathbb{N}$, sodass für $0 \leq j < n$ eine Transition $P_j \overset{a_{j+1}}{\rightarrow} p_{j+1}$ existiert. Dann ist $(a_1, ..., a_n) \in \mathcal{A}^*$ ein Trace von p_0 in T. Wir definieren

$$\text{Traces}(p) \coloneqq \{w \in \mathcal{A}^* \mid w \text{ ist Trace von } p \text{ in } T\}.$$

Definition 9 (Trace-Inklusion)**.**
Für zwei Prozesse $p_1, p_2 \in \mathcal{P}$ wird p_1 genau dann von p_2 Trace-inkludiert, wenn $\text{Traces}(p_1) \subseteq \text{Traces}(p_2)$. Da Trace-Inklusion keine symmetrische Relation ist, ist sie eine Präordnung.

Für Symmetrie muss zusätzlich gelten, dass auch p_2 von p_1 Trace-inkludiert wird. Dies ist gleichzusetzen mit der Aussage der Mengengleichheit Traces(p_1) = Traces(p_2).

Definition 10 (Trace-Äquivalenz und Enabledness)**.**
Zwei Prozesse $p_1, p_2 \in \mathcal{P}$ sind Trace-äquivalent, wenn Traces(p_1) = Traces(p_2) gilt. Wir schreiben $p_1 =_T p_2$. In anderen Worten sind zwei Prozesse Trace-äquivalent, wenn sie die gleichen Aktionssequenzen in T erlauben.

Enabledness bezeichnet den speziellen Fall, dass nur Aktionssequenzen der Länge 1 betrachtet werden. Dies bedeutet, dass p_1 und p_2 genau dann in der Relation stehen, wenn $I(p_1) = I(p_2)$, geschrieben $p_1 =_E p_2$.

Beispiel 3.
Schauen wir uns erneut die Prozesse P_1 und P_2 aus Abbildung 1 an. P_1 wird von P_2 Trace-inkludiert, da

$$\text{Traces}(P_1) = \{\lambda, (a), (a,b)\} \subseteq \{\lambda, (a), (a,b), (a,c)\} = \text{Traces}(P_2).$$

Trace-Äquivalenz liegt jedoch nicht vor, da $(a,c) \in \text{Traces}(P_2)$, aber $(a,c) \notin \text{Traces}(P_1)$ und somit Traces(P_1) $\neq$ Traces(P_2).

Weiterhin gilt $P_1 =_E P_2$, da $I(P_1) = \{a,b\} = I(P_2)$. Das Beispiel mit Queries zu Traces und Enabledness lässt sich in CAAL unter „Traces Example“ auffinden.

4.3 Simulation

Definition 11 (Simulation)**.**
Simulation ist eine binäre Relation $R \subseteq \mathcal{P} \times \mathcal{P}$, für die für jedes Paar von Prozessen $p_1, p_2 \in \mathcal{P}$ und für alle Aktionen $a \in \mathcal{A}$ gilt:

- wenn $(p_1, p_2) \in R$ und $p_1 \xrightarrow{a} p_1'$, dann $\exists p_2' \,.\, p_2 \xrightarrow{a} p_2'$ und $(p_1', p_2') \in R$

p_1 wird von p_2 simuliert, wenn es eine Simulation R gibt mit $(p_1, p_2) \in R$. Wir schreiben $p_1 \lesssim p_2$. Bei dieser Relation handelt es sich aufgrund der nicht-Symmetrie um eine Präordnung. Wenn $p_1 \lesssim p_2$ und $p_2 \lesssim p_1$, dann schreiben wir $p_1 \simeq p_2$ bzw. $p_2 \simeq p_1$. Die Symmetrieeigenschaft von $\simeq$ ist daher gegeben und es liegt eine Äquivalenzrelation vor, welche wir Simulationsäquivalenz nennen.

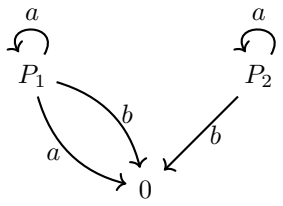

Abbildung 2: LTS mit simulationsäquivalenten Prozessen P_1 und P_2

Beispiel 4 (Simulation)**.** Gegeben sind die Prozesse $P_1 = a.P_1 + a.0 + b.0$ und $P_2 = a.P_2 + b.0$, welche in Abbildung 2 dargestellt sind. P_1 verhält sich so, dass eine Aktion b in den Nullprozess führt und durch Aktion a entweder wieder P_1 oder auch der Nullprozess erreicht wird, wohingegen P_2 deterministisch mit a in der eigenen Schleife bleibt oder mit b zum Nullprozess gelangt. Damit $P_1 \lesssim P_2$ gilt, muss eine Relation R_1 existieren, die das obige Kriterium erfüllt. Diese Relation ist gegeben mit

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

Damit zusätzlich $P_1 \simeq P_2$ gilt, muss auch $P_2 \lesssim P_1$ gelten. Die Relation R_2 beweist dies mit

$$R_2 = \{(P_2, P_1), (0, 0)\}.$$

Dieses Beispiel ist in CAAL unter „Simulation and Bisimulation Example“ zu finden.

4.4 Failures

Die bisherigen Gleichheitsbegriffe haben sich mit möglichen Observationen befasst. Namensgebend für Failures ist, dass sich diese Relation mit den nicht möglichen, negativen Observationen beschäftigt. Hierfür wird für ein $p_0 \in \mathcal{P}$ für jede Trace $t = (a_1, ..., a_n) \in \text{Traces}(p_0)$ eine Menge $X \subseteq \mathcal{A}$ von Aktionen betrachtet, die nach dieser Sequenz t nicht mehr ausführbar sind. Diese Menge X nennen wir „Refusal Set“. Weiterhin definieren wir $\langle t, X\rangle$ als Failure-Paar.

Definition 12 (Failures-Inklusion)**.**
$\langle t, X\rangle \in \mathcal{A}^* \times 2^{\mathcal{A}}$ ist ein Failure-Paar von p_0, wenn die Prozesse $p_1, ..., p_n$ existieren, sodass $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} ... \xrightarrow{a_{n-1}} p_{n-1} \xrightarrow{a_n} p_n$ und $I(p_n) \cap X = \varnothing$. Sei $F(p)$ die Menge aller Failure-Paare von p. Für zwei Prozesse $p, q \in \mathcal{P}$ wird p genau

dann von q Failures-inkludiert, wenn $F(p) \subseteq F(q)$. Wir schreiben $p \preceq_F q$. Da Failures-Inklusion keine symmetrische Relation ist, ist sie eine Präordnung.

Definition 13 (Failures-Äquivalenz)**.**
Zwei Prozesse $p, q \in \mathcal{P}$ sind Failures-Äquivalent, wenn $\mathrm{F}(p) = \mathrm{F}(q)$ gilt. Wir schreiben $p =_F q$.

Failures sind eine Erweiterung der Trace-Semantik, da zusätzlich zu den Traces selber der Refusal-Set betrachtet wird. Daher lässt sich Traces(p) auch über $F(p)$ definieren, indem nur Failure-Paare betrachtet werden, deren Refusal Set die leere Menge ist:

$$\mathrm{Traces}(p) = \{t \in \mathcal{A}^* \mid \langle t, \varnothing \rangle \in F(p)\}$$

Aufgrund dessen gilt $p =_F q \rightarrow p =_T q$.

Beispiel 5.
Für dieses Beispiel betrachten wir ein weiteres Mal die Prozesse P_1 und P_2 aus Abbildung 1. Wir haben bereits festgestellt, dass $P_1 =_T P_2$. Es existiert jedoch das Failure-Paar $\langle (a), \{b\} \rangle \in F(P_2)$, da von P_2 nach a Aktion b nicht mehr möglich sein muss. Dieses Failure-Paar ist kein Element von $F(P_1)$, somit muss $F(P_1) \neq F(P_2)$ gelten und die beiden Prozesse sind nicht Failure-äquivalent. Dieses Beispiel ist in CAAL unter „Failures First Example“ aufrufbar und die zugehörigen Queries zu Failures ausführbar.

Ein Beispiel für zwei Failure-äquivalente Prozesse stellt Abbildung 3 dar, die die Prozesse $P_1 = a.(b.0 + c.d.0) + a.(c.e.0 + f.0)$ und
$P_2 = a.(b.0 + c.e.0) + a.(c.d.0 + f.0)$ mit $P_1 =_F P_2$ visualisiert. Zu sehen ist dies auch im CAAL unter „Failures Second Example“.

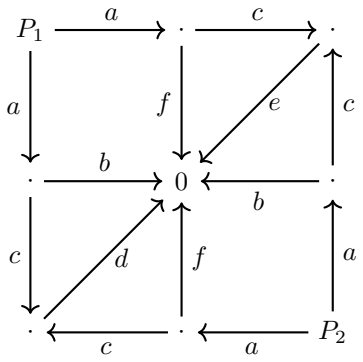

Abbildung 3: LTS mit Failure-äquivalenten Prozessen P_1 und P_2

4.5 Bisimulation

Definition 14 (Bisimulation)**.**
Der Begriff der Bisimulation beschreibt eine binäre Relation $R \subseteq \mathcal{P} \times \mathcal{P}$, für die für jedes Paar von Prozessen $p_1, p_2 \in \mathcal{P}$ und für alle Aktionen $a \in \mathcal{A}$ gilt:

- wenn $(p_1, p_2) \in R$ und $p_1 \xrightarrow{a} p_1'$, dann $\exists p_2' \,.\, p_2 \xrightarrow{a} p_2'$ und $(p_1', p_2') \in R$
- wenn $(p_1, p_2) \in R$ und $p_2 \xrightarrow{a} p_2'$, dann $\exists p_1' \,.\, p_1 \xrightarrow{a} p_1'$ und $(p_1', p_2') \in R$

Zwei Prozesse p_1 und p_2 sind bisimilar, wenn eine Bisimulation R zwischen ihnen existiert, sodass $(p_1, p_2) \in R$. Wir schreiben $p_1 \sim p_2$.

Hierbei liegt der Unterschied im Vergleich zur Simulationsäquivalenz darin, dass eine einzige Relation beide Prozesse simuliert und nicht zwei unterschiedliche Relationen erlaubt sind, um jeweils eine Richtung abzudecken.

Beispiel 6 (Unterschied zwischen Bisimulation und Simulationsäquivalenz)**.**
Um den Unterschied zwischen Simulationsäquivalenz und Bisimulation zu veranschaulichen, betrachten wir erneut die zwei Prozesse P_1 und P_2 aus Abbildung 2. Wir haben bereits festgestellt, dass diese beiden Prozesse simulationsäquivalent aufgrund folgender Relationen sind:

$$R_1 = \{(P_1, P_2), (0, P_2), (0, 0)\}$$
$$R_2 = \{(P_2, P_1), (0, 0)\}$$

P_1 und P_2 sind jedoch nicht bisimilar, da hierfür eine einzige Relation existieren muss, durch die sich die beiden Prozesse gegenseitig simulieren. Dies scheitert

dadurch, dass von P_1 eine Aktion a möglich ist, die in 0 resultiert. Folglich müssten auch wegen der Transition $P_2 \xrightarrow{a} P_2$ die Prozesse 0 und P_2 bismilar sein. Von P_2 sind sowohl eine Aktion a als auch b möglich, von dem Nullprozess lassen sich jedoch per Definition keine Aktionen mehr ausführen, sodass die beiden Prozesse nicht bisimilar sein können und folglich auch $P_1 \sim P_2$ nicht gelten kann. Zur Vergewisserung ist dieses Beispiel mit entsprechenden Queries unter „Simulation and Bisimulation Example“ aufzufinden.

4.6 Das Linear-Time-Branching-Time-Spektrum

In Abbildung 4 ist das Linear-Time-Branching-Time-Spektrum zu sehen, das van Glabbeek in [4] aufgestellt hat und in [1] angepasst wurde, um genau die Gleichheiten bzw. Präordnungen zu illustrieren, die im Spectroscopy Energy Game verifiziert werden können. In dieser Arbeit wurde nur auf eine Teilmenge dieser Gleichheiten im Detail spezifisch eingegangen, da eine genaue Auseinandersetzung mit allen Gleichheiten den Rahmen übersteigen würde.

Das Spektrum stellt einen azyklischen Graphen mit gerichteten Kanten dar, bei dem eine Kante von a nach b so zu interpretieren ist, dass aus dem Zutreffen von Gleichheit a auch auf Gleichheit b geschlossen werden kann. Bisimulation stellt somit die feinste bzw. engste Gleichheit dar, wohingegen Enabledness die gröbste bzw. allgemeinste Gleichheit des Spektrums repräsentiert.

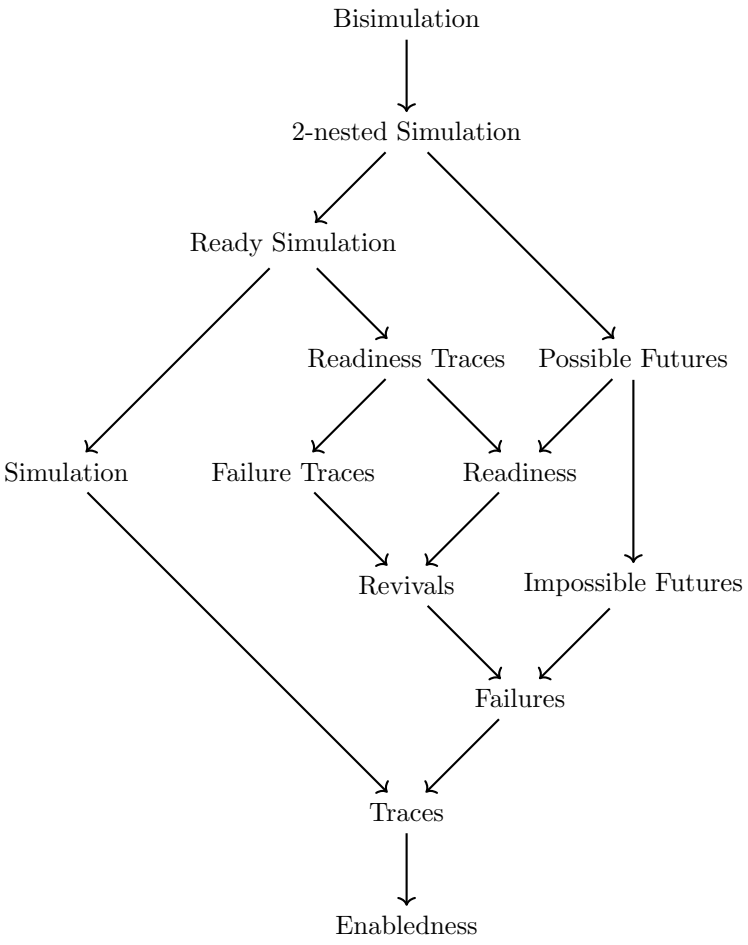

Abbildung 4: Hierarchie von Gleichheitsbegriffen

5 Energiespiele

In diesem und den darauf folgenden zwei Kapiteln wird das Prinzip der Spectroscopy erläutert, so wie es im Paper [1] von Benjamin Bisping definiert ist.

Als Energiespiele wird eine bestimmte Klasse von Spielen bezeichnet, die in der formalen Verifikation und Theorie formaler Sprachen als mathematische Abstraktion verwendet wird. Zweck dieser Energiespiele ist es häufig, Eigenschaften von Systemen zu überprüfen, die Zustandsübergänge durchführen. Hierfür treten üblicherweise zwei Spielende gegeneinander an. Aufgabe von Spieler 1 ist es meist, ein bestimmtes Ziel zu erreichen, wohingegen Spielerin 2 dies

zu verhindern versucht. Als Zusatz zu generischen Logikspielen existiert bei Energiespielen eine Ressourcenkomponente, die „Energie“. Zustandsübergänge sind daher nicht nur durch die logische Struktur selbst, sondern auch durch die verfügbare Energie bedingt. Energiespiele werden meist als gerichtete Graphen modelliert, wobei die Knoten Spielzustände und die Kanten Übergänge darstellen. Für jede Kante existiert zudem ein Vektor, der wie ein Kantengewicht als Update der Energie fungiert. In dem im Folgenden definierten Energiespiel interpretieren wir Spieler 1 als Angriff und Spielerin 2 als Verteidigung.

Definition 15 (N-dimensionale degressive Energiespiele)**.**
Ein n-dimensionales degressives Energiespiel $\mathfrak{S}[z_0, e_0] = (Z, Z_V, \rightarrowtail, g, z_0, e_0)$ besteht aus

- einer Menge Z von Spielzuständen,
- einer Menge $Z_V \subseteq Z$ von Verteidigungspositionen,
- einer Relation zwischen Spielzuständen $\rightarrowtail \subseteq Z \times Z$, die Spielzüge darstellt,
- einer Gewichtsfunktion $g: (\rightarrowtail) \rightarrow \mathbb{Z}_0^-$, die jedem Spielzug seine Energiekosten zuweist,
- einer Startposition $z_0 \in Z$ und
- einem Startbudget an Energie $e_0 \in (\mathbb{N} \cup \{\infty\})^n$.

Für $z \rightarrowtail z'$ mit $g((z, z')) = u$ und $z, z' \in Z$ schreiben wir $z \overset{u}{\rightarrowtail} z'$.

Degressiv bedeutet in diesem Kontext, dass bei jedem Spielzug die verfügbare Energie monoton sinkt.

Definition 16 (Spieldurchläufe und Energieniveau)**.**
Wir nennen die endlichen Wege $w = z_0, ..., z_n \in Z^n$ bzw. unendlichen Wege $w = z_0, z_1, ... \in Z^\infty$ mit $z_i \overset{w(z_i, z_{i+1})}{\rightarrowtail} z_{i+1}$ des Spielgraphen Spieldurchläufe von $\mathfrak{S}[z_0, e_0]$.

Das Energieniveau $EN_w(i)$ in Runde i eines Spieldurchlaufs w ist rekursiv definiert als

- $EN_w(0) \coloneqq e_0$
- $EN_w(i+1) \coloneqq EN_w(i) + w(z_i, z_{i+1})$

Mit dem Weglassen des Arguments, EN_w, ist das Energieniveau zum Ende des Spieldurchlaufes w gemeint.

Spieldurchläufe werden von der Verteidigung gewonnen, wenn mindestens eine Komponente des Energieniveaus in einer Runde negativ ist. Spieldurchläufe unendlicher Länge werden ebenfalls von der Verteidigung gewonnen. Wenn für einen Spielzustand eines Durchlaufs kein weiterer Übergang existiert, so gewinnt die Person, die nicht in diesem Spielzustand am Zug ist. Der Angriff gewinnt also, wenn der letzte Spielzustand Element von Z_V ist, und sonst die Verteidigung.

Definition 17 (Gewinnstrategien)**.**
Eine Strategie ist eine Funktion $s : Z^* \to (\rightarrowtail)$, die einen Weg von gespielten Zuständen auf einen darauf möglichen Spielzug abbildet, sodass $s(z_0, ..., z_i) \in (z_i \rightarrowtail *)$. Wenn $z_i \in Z_V$, dann nennen wir s eine Verteidigungsstrategie, ansonsten eine Angriffsstrategie. s wird eine Gewinnstrategie genannt, wenn alle Spieldurchläufe, die sich aus der Anwendung von s ergeben, zu einem Sieg für ausschließlich Angriff oder Verteidigung führen. Wir definieren

$$GB_A(z) \coloneqq \{e \mid \text{Angriff besitzt eine Gewinnstrategie für } \mathfrak{S}[z, e]\}$$

als die Menge der Budgets, mit denen der Angriff das Spiel $\mathfrak{S}$ von Spielzustand z gewinnt. Weiterhin lässt sich feststellen, dass für jedes Budget e, das komponentenweise größer als ein $e' \in GB_A(z)$ ist, auch $e \in GB_A(z)$ gelten muss. Daher ist $GB_A(z)$ entweder die leere Menge, nämlich im Falle einer nicht existierenden Gewinnstrategie für den Angriff, oder $|GB_A(z)| = \infty$. Da für degressive Energiespiele nur minimale Gewinnbudgets von Relevanz sind, definieren wir zusätzlich

$$GB_A^{min} \coloneqq \{e \in GB_A(z) \mid \neg\exists e' \in GB_A(z).\ e' < e\},$$

wobei $e' < e$ für $e = (e_0, ..., e_n),\ e' = (e'_0, ..., e'_n)$ bedeutet, dass
$\forall\ 0 \leq i \leq n.\ e'_i < e_i$.

Um die Struktur der in dieser Arbeit verwendeten Energiespiele zu verdeutlichen, modellieren wir nun ein Beispiel eines solchen Spieles, welches in der Praxis keine Relevanz besitzt und nur zu Anschauungszwecken dient.

Beispiel 7.
Wir definieren das 2-dimensionale degressive Energiespiel
$\mathfrak{S}[P_1, (3, 3)] = (\mathcal{P}, \{P_2, P_3, P_5, P_6, P_9\}, \rightarrowtail, g, P_1, (3, 3))$ über dem
LTS $T = (\mathcal{P}, \mathcal{A}, \to)$ aus Abbildung 5. Hierbei korrespondieren die Spielzustände

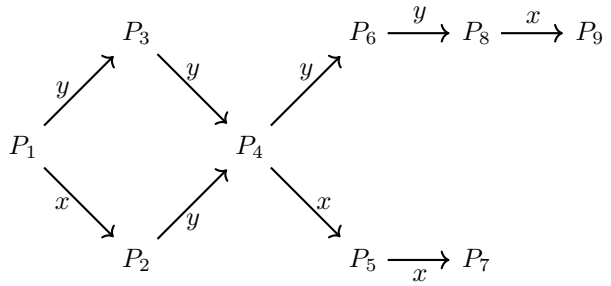

Abbildung 5: LTS T

mit der Prozessmenge $\mathcal{P}$ und die Spielzüge mit der Transitionsrelation $\rightarrow$ insofern, dass $\rightarrowtail := \{(P, Q) \mid P, Q \in \mathcal{P} \wedge \exists a \in \mathcal{A}.\ P \xrightarrow{a} Q\}$. g ist so definiert, dass $g((P, Q)) = (-1, 0)$, wenn $P \xrightarrow{x} Q$, und ansonsten $(0, -1)$. Der Spielgraph hierzu ist in Abbildung 6 zu sehen.

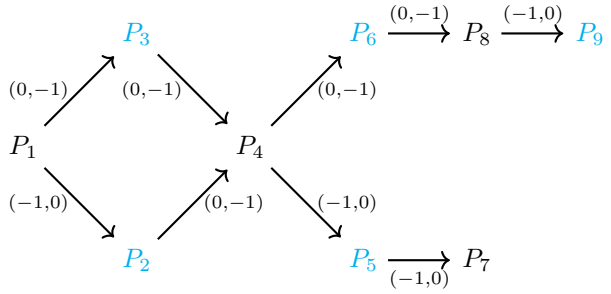

Abbildung 6: Spielgraph von $\mathfrak{S}[P_1, (3, 3)]$ mit blau markierten Verteidigungspositionen

Wir betrachten die vier vollständigen Spieldurchläufe mit ihren zugehörigen finalen Energieniveaus

- $w_1 = (P_1, P_2, P_4, P_5, P_7)$ mit $EN_{w_1} = (0, 2)$,
- $w_2 = (P_1, P_2, P_4, P_6, P_8, P_9)$ mit $EN_{w_2} = (1, 0)$,
- $w_3 = (P_1, P_3, P_4, P_5, P_7)$ mit $EN_{w_3} = (1, 1)$ und
- $w_4 = (P_1, P_3, P_4, P_6, P_8, P_9)$ mit $EN_{w_4} = (2, -1)$.

Bei sowohl w_1 als auch w_3 existiert kein weiterer möglicher Spielzug. Da P_7 eine Angriffsposition ist, gewinnt folglich die Verteidigung das Spiel. Ebenso gewinnt

die Verteidigung im Spieldurchlauf w_4, da das Energiebudget überschritten wurde. In w_2 gewinnt hingegen der Angriff, da bei nicht überschrittenem Budget die Verteidigung keine Spielzüge mehr zur Verfügung hat. Der Angriff besitzt in diesem Beispiel eine Gewinnstrategie, da unabhängig von den Spielzügen der Verteidigung, die zur Simplizität des Beispiels immer nur eine Option an Spielzügen besitzt, ein Spieldurchlauf forciert werden kann, der zu einem Sieg führt.

Weiterhin ist $GB_A^{min}(P_1) = \{(3,3) - EN_{w_2}\} = \{(2,3)\}$, da der Angriff ein initiales Energiebudget von mindestens $(2,3)$ benötigt, um eine Gewinnstrategie zu besitzen.

6 Unterscheidung von Prozessen durch HML

Wie in Beispiel 2 bereits angedeutet, lässt sich das Verhalten von Prozessen durch HML formalisieren, um Unterschiede herauszustellen. Solche unterscheidenden Formeln lassen sich dann weiter untersuchen, um Informationen über die Gleichheitsbegriffe zu erlangen, die auf die unterschiedenen Prozesse zutreffen.

Definition 18 (Unterscheidende Formeln)**.**
Sei $T = (\mathcal{P}, \mathcal{A}, \rightarrow)$ ein beliebiges LTS. Eine Formel $\varphi \in \text{HML}[\mathcal{A}]$ gilt als unterscheidende Formel zweier Prozesse $p_1, p_2 \in \mathcal{P}$, wenn $p_1 \models \varphi$ und $p_2 \not\models \varphi$. Wenn $\varphi \in \mathcal{O}_X \subseteq \text{HML}[\mathcal{A}]$ p_1 und p_2 unterscheidet, lässt sich daraus $p_1 \not\preceq_X p_2$ schließen. Sollte solch eine Formel φ nicht existieren, gilt $p_1 \preceq_X p_2$. $\mathcal{O}_X$ repräsentiert dabei eine bestimmte Formelmenge, die zur Unterscheidung zweier beliebiger Prozesse im Kontext eines Gleichheitsbegriffs X benötigt wird. Auf diese Formelmengen wird im folgenden Abschnitt näher eingegangen.

6.1 Quantifikation von unterscheidenden Formeln

Damit Verifikationsalgorithmen, die auf dem Linear-Time-Branching-Time-Spektrum arbeiten, effektive Problemlösung betreiben können, ist es sinnvoll, unterscheidende Formeln bzw. deren Sprache zu quantifizieren. Die quantitative Repräsentation ist somit als „Kosten“ der Formeln zu verstehen. Um die Kosten von unterscheidenden Formeln für die Gleichheitsbegriffe im Linear-Time-Branching-Time-Spektrum zu repräsentieren, benutzen wir sechs Dimensionen:

1. Observationstiefe
2. Verschachtelungstiefe von Konjunktionen
3. maximale Tiefe positiver Observationen in Konjunktionen
4. zweitgrößte Tiefe positiver Observationen in Konjunktionen
5. maximale Tiefe negativer Observationen in Konjunktionen
6. Verschachtelungstiefe von Negationen

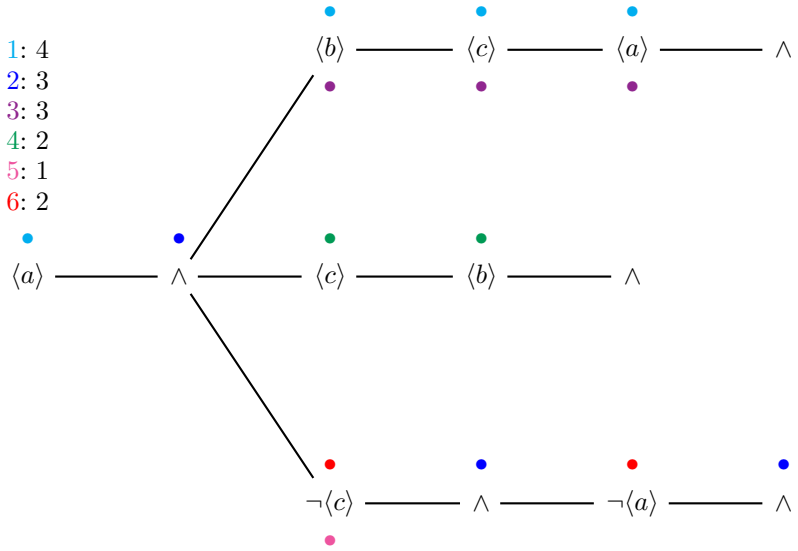

Abbildung 7: Kosten der Formel
$\varphi \coloneqq \langle a \rangle \bigwedge\{\langle b \rangle \langle c \rangle \langle a \rangle \bigwedge \varnothing, \langle c \rangle \langle b \rangle \bigwedge \varnothing, \neg\langle c \rangle \bigwedge\{\neg\langle a \rangle \bigwedge \varnothing\}\}$

In Abbildung 7 ist am Beispiel der in Baumstruktur dargestellten Formel $\varphi \coloneqq \langle a \rangle \bigwedge\{\langle b \rangle \langle c \rangle \langle a \rangle \bigwedge \varnothing, \langle c \rangle \langle b \rangle \bigwedge \varnothing, \neg\langle c \rangle \bigwedge\{\neg\langle a \rangle \bigwedge \varnothing\}\}$ illustriert, wie sich die Kosten für jede Dimension entfalten. Um die einzelnen Dimensionen zu unterscheiden, sind diese basierend auf ihrer in 6.1 zugewiesenen Farbe eingefärbt. Ein Punkt über bzw. unter einem Knoten im Baum signalisiert, dass dieser Knoten für die zugehörige Dimension zu den Kosten beiträgt.

Durch die Quantifikation der in den Teilsprachen $\mathcal{O}_X$ enthaltenen Formeln lässt sich nun für verschiedene Gleichheitsbegriffe X bestimmen, welche Kosten diese Formeln maximal aufweisen dürfen. Diese Kosten sind in Tabelle 1 zu sehen.

Es mag auffallen, dass $\mathcal{O}_{Bisimulation}$ keine Kosteneinschränkungen besitzt, also $\mathcal{O}_{Bisimulation} = \mathrm{HML}[\mathcal{A}]$. Dies ist konsistent mit dem Hennessy-Milner-

$\mathcal{O}_X$	Observationstiefe	VT Konj.	max. Tiefe pos. Obs. in Konj.	zweitgrößte Tiefe pos. Obs. in Konj.	max. Tiefe neg. Obs. in Konj.	VT Negationen
$\mathcal{O}_{\text{Bisimulation}}$	∞	∞	∞	∞	∞	∞
$\mathcal{O}_{\text{2-nested Simulation}}$	∞	∞	∞	∞	∞	1
$\mathcal{O}_{\text{Ready Simulation}}$	∞	∞	∞	∞	1	1
$\mathcal{O}_{\text{Readiness Traces}}$	∞	∞	∞	1	1	1
$\mathcal{O}_{\text{Possible Futures}}$	∞	2	∞	∞	∞	1
$\mathcal{O}_{\text{Simulation}}$	∞	∞	∞	∞	0	0
$\mathcal{O}_{\text{Failure Traces}}$	∞	∞	∞	0	1	1
$\mathcal{O}_{\text{Readiness}}$	∞	2	1	1	1	1
$\mathcal{O}_{\text{Revivals}}$	∞	2	1	0	1	1
$\mathcal{O}_{\text{Impossible Futures}}$	∞	2	0	0	∞	1
$\mathcal{O}_{\text{Failures}}$	∞	2	0	0	1	1
$\mathcal{O}_{\text{Traces}}$	∞	1	0	0	0	0
$\mathcal{O}_{\text{Enabledness}}$	1	1	0	0	0	0

Tabelle 1: Obere Schranke an Kosten der Sprachen $\mathcal{O}_X$

Theorem [5], welches den Umkehrschluss formuliert, dass zwei Prozesse genau dann bisimilar sind, wenn keine unterscheidende HML-Formel existiert.

7 Das „Spectroscopy Energy Game“

In diesem Kapitel gehen wir auf das Energiespiel ein, das schließlich auch im Rahmen dieser Arbeit implementiert und in CAAL integriert wurde. In diesem Spiel ist es Ziel des Angriffs, unterscheidende Formeln zu konstruieren, die durch ein Energiebudget, ihren Kosten, in ihrer Ausdrucksstärke limitiert sind.

Definition 19 (Spectroscopy Energy Game)**.**
Das 6-dimensionale Spectroscopy Energy Game $\mathfrak{S}_\Delta[z_0, e_0] = (Z, Z_V, \rightarrowtail, g, z_0, e_0)$ über einem LTS $T = (\mathcal{P}, \mathcal{A}, \rightarrow)$ setzt sich zusammen aus einer Menge Z, partitioniert in generische Angriffspositionen $(p, Q)_A \in Z \setminus Z_V$, Angriffspositionen für Klauseln $(p, q)^\wedge_A \in Z \setminus Z_V$ und Konjunktionspositionen für die Verteidigung $(p, Q, Q_*)_V \in Z_V$, wobei $p, q \in \mathcal{P}$ und $Q, Q_* \subseteq \mathcal{P}$. Die Spielzugrelation $\rightarrowtail$ ergibt sich aus sechs Vorschriften:

1. Observationen $(p, Q)_A \overset{(-1,0,0,0,0,0)}{\rightarrowtail} (p', Q')_A$,
   für $p \xrightarrow{a} p'$, $Q' = \{q' \in \mathcal{P} \mid \exists q \in Q.\ q \xrightarrow{a} q'\}$

2. konjunktive Herausforderungen $(p, Q)_A \overset{(0,-1,0,0,0,0)}{\rightarrowtail} (p, Q \setminus Q_*, Q_*)_V$, für $Q_* \subseteq Q$

3. konjunktive Revivals $(p, Q, Q_*)_V \overset{(min_{\{1,3\}},0,0,0,0,0)}{\rightarrowtail} (p, Q_*)_A$, für $Q_* \neq \varnothing$

4. konjunktive Antworten $(P, Q, Q_*)_V \overset{(0,0,0,min_{\{3,4\}},0,0)}{\rightarrowtail} (p, q)^{\wedge}_A$, für $q \in Q$

5. positive Entscheidungen $(p, q)^{\wedge}_A \overset{(min_{\{1,4\}},0,0,0,0,0)}{\rightarrowtail} (p, \{q\})_A$

6. negative Entscheidungen $(p, q)^{\wedge}_A \overset{(min_{\{1,5\}},0,0,0,0,-1)}{\rightarrowtail} (q, \{p\})_A$, für $p \neq q$

In diesen Vorschriften ist auch die Gewichtsfunktion g beschrieben. In einigen Updatevektoren dieser ist eine Komponente der Form $min_{\{x,y\}}$ mit $x, y \in \{1, ..., 6\}$ angegeben. Für ein Energieniveau $e = (e_1, ..., e_6)$, das mit einem Update der Form $u = (min_{\{x,y\}}, 0, 0, 0, 0, -1)$ aktualisiert wird, ergibt sich das Energieniveau $e' = (min(e_x, e_y), e_2, e_3, e_4, e_5, e_6 - 1)$.

7.1 Generation der unterscheidenden Formeln

Schauen wir uns nun an, wie die sechs verschiedenen Arten von Spielzügen jeweils unterscheidende Formeln aufbauen. Hierzu muss beachtet werden, dass im Allgemeinen nur dann unterscheidende Formeln generiert werden, wenn die Ausführung eines Spielzuges Teil der Gewinnstrategie des Angriffs ist.

Definition 20 (Formeln der Gewinnstrategie)**.**
Die Formelmenge der Gewinnstrategie des Angriffs GSF_A mit einer Menge an Gewinnbudgets des Angriffs GB_A für eine Position mit gegebenem Energieniveau ist für die sechs Arten von Spielzügen folgendermaßen definiert:

1. für eine Observation ist $\langle a \rangle \varphi \in GSF_A((p, Q)_A, e)$, wenn $e' = (e_1 - 1, e_2, ..., e_6) \in GB_A((p', Q')_A)$ und $\varphi \in GSF_A((p', Q')_A, e')$.

2. für eine konjunktive Herausforderung ist $\varphi \in GSF_A((p, Q)_A, e)$, wenn $e' = (e_1, e_2 - 1, e_3, ..., e_6) \in GB_A((p, Q, Q_*)_V)$ und $\varphi \in GSF_A((p, Q, Q_*)_V, e')$.

3. für ein konjunktives Revival ist $\bigwedge_{q_* \in Q_*} \psi_{q_*} \in GSF_A((p, Q, Q_*)_V, e)$, wenn $e' = (min(e_1, e_3), e_2, ..., e_6) \in GB_A((p, Q_*)_A)$ und $\psi_{q_*} \in GSF_A((p, Q_*)_A, e')$ eine modale Observation ist.

4. für eine konjunktive Antwort ist $\bigwedge_{q \in Q} \psi_q \in GSF_A((p, Q, Q_*)_V, e)$, wenn $e' = (e_1, e_2, e_3, min(e_3, e_4), e_5, e_6) \in GB_A((p, q)^{\wedge}_A)$ und $\psi_q \in GSF_A((p, q)^{\wedge}_A), e')$ für jedes $q \in Q$.

5. für eine positive Entscheidung ist $\varphi \in GSF_A((p,q)^\wedge_A, e)$, wenn
   $e' = (min(e_1, e_4), e_2, ..., e_6) \in GB_A((p,\{q\})_A)$ und
   $\varphi \in GSF_A((p,\{q\})_A, e')$ eine modale Observation ist.

6. für eine negative Entscheidung ist $\neg\varphi \in GSF_A((p,q)^\wedge_A, e)$, wenn
   $e' = (min(e_1, e_5), e_2, ..., e_5, e_6 - 1) \in GB_A((q,\{p\})_A)$ und
   $\varphi \in GSF_A((q,\{p\})_A, e')$ eine modale Observation ist.

7.2 Extraktion der Gleichheitsbegriffe

Wir wissen nun, dass für zwei Prozesse $P_1 \preceq_X P_2$ genau dann gilt, wenn der Angriff eine Gewinnstrategie für $\mathfrak{S}_\Delta[(P_1,\{P_2\})_A, e_X]$ besitzt, wobei X ein Gleichheitsbegriff und e_X die obere Schranke an Kosten der Sprache $\mathcal{O}_X$ wie in Tabelle 1 ist. Im nächsten Schritt gilt es folglich zu untersuchen, welche minimalen Gewinnbudgets für ein solches Spiel existieren, um nicht für jedes e_X berechnen zu müssen, ob der Angriff mit diesem Startbudget gewinnt. Effizienter ist es, zu vergleichen, ob ein Gewinnbudget existiert, das in allen Dimensionen kleiner ist als die jeweiligen oberen Schranken der verschiedenen $\mathcal{O}_X$. Dies gilt genau dann, wenn eine Präordnung X nicht zutrifft. Im Umkehrschluss gilt daher, dass eine Präordnung X genau dann zutrifft, wenn jedes Gewinnbudget in mindestens einer Dimension größer ist, als es die obere Schranke an Kosten der Sprache $\mathcal{O}_X$ erlaubt.

Definition 21 (Berechnung der Gewinnbudgets des Angriffs)**.**
Die Gewinnbudgets des Angriffs $GB_A(z)$ für eine Position $z \in Z$ sind folgendermaßen induktiv definiert:

- Der Angriff gewinnt mit beliebigem Budget in Spielzuständen, in denen die Verteidigung keinen Spielzug ausführen kann: $\{0\}^6 \in GB^{min}_A(z)$ mit $z \in Z_V$ und $\neg\exists z' \in Z.\ (z,z') \in \rightarrowtail$

- Wenn die Verteidigung ausführbare Spielzüge z' hat, sodass $z \overset{u_i}{\rightarrowtail} z'_i$, $z \in Z_V$ für eine Indexierung $i \in I$ über alle verfügbaren Spielzüge, dann gewinnt der Angriff, wenn ein Budget übrig ist, das größer gleich aller nach dem Spielzug der Verteidigung notwendigen Budgets ist: Sei e'_i das Energieniveau nach Update u_i auf e. Wenn für alle i gilt, dass $e'_i \in GB_A(z'_i)$, dann gilt $e \in GB_A(z)$.

- Wenn der Angriff einen Spielzug ausführen kann, nach dem das aktualisierte Budget Teil der Gewinnbudgets für die Folgeposition ist, gewinnt

er: Wenn $z \in Z \setminus Z_V, z \overset{u}{\rightarrowtail} z'$ und e' das Energieniveau nach Update u auf e ist, sodass $e' \in GB_A(z')$, dann ist $e \in GB_A(z)$.

Die Gewinnbudgets des Angriffs müssen folglich von hinten nach vorne bzw. von den Spielzuständen, in denen die Verteidigung keine Spielzüge mehr ausführen kann, zu dem Startzustand aufgerollt werden. Da dies also entgegen der eigentlichen Spielrichtung geschieht, müssen auch die Updates rückwarts berechnet werden.

Definition 22 (Inverse Updates)**.**
Sei $\text{upd}(e, u) = e'$ die Updatefunktion, die durch das Kantengewicht u das Energieniveau e zu e' aktualisiert. Dann ist die inverse Updatefunktion $\text{upd}^{-1}(e', u) \coloneqq sup(\{e\} \cup \{e_{min}(i) \mid \exists i \in \{1, ..., 6\}.\ u_i = min_{\{x_i, y_i \mid x_i, y_i \in \{1,...,6\} \wedge x \neq y\}}\})$ mit $e_i = e'_i - u_i$, wenn $u_i \in Z_0^-$ und $e_{min}(i)_{x_i} = e_{min}(i)_{y_i} = e'_i$, wenn $u_i = min_{\{x_i, y_i\}}$ und $e_{min}(i)_k = 0$, wenn $y_i \neq k \neq x_i$, für alle i.

Beispiel 8.
Sei $e \coloneqq (3, 1, 0, 0, 1, 1), u \coloneqq (min_{\{1,5\}}, 0, 0, 0, 0, -1)$ und $e' = \text{upd}(e, u) = (1, 1, 0, 0, 1, 0)$. Demnach ist $\text{upd}^{-1}(e', u) = sup(\{(1, 1, 0, 0, 1, 1), (1, 0, 0, 0, 1, 0)\}) = (1, 1, 0, 0, 1, 1)$. Aktualisieren wir dieses Ergebnis wieder „zurück“, so erhalten wir wiederum $\text{upd}((1, 1, 0, 0, 1, 1), u) = (1, 1, 0, 0, 1, 0) \geq e'$.

Mithilfe der inversen Updatefunktion lassen sich nun die minimalen Gewinnbudgets des Angriffs GB_A^{min} berechnen. Dies geschieht durch Algorithmus 1 aus [1].

Der Algorithmus initialisiert gemäß 21 eine To-do-Liste mit allen Spielzuständen, in denen die Verteidigung keine Spielzüge ausführen kann, setzt die zugehörigen Gewinnbudgets des Angriffs auf $\{0\}^6$ und arbeitet die Zustände iterativ ab. Dabei werden immer wieder sowohl neue als auch bereits erkundete Spielzustände zu der Liste hinzugefügt, für welche eine neue Möglichkeit gefunden wird, in diesem Zustand zu gewinnen. In Zeile 7 werden die invers aktualisierten Budgets als provisionale Gewinnbudgets von Angriffspositionen gespeichert. In Zeile 11 werden die Gewinnbudgets des Angriffs nur dann aktualisiert, wenn dieser in allen Folgepositionen Gewinnbudgets besitzt. Ist dies der Fall, werden die Minima der Suprema aller möglichen Kombinationen an Gewinnbudgets, die jede Folgeposition repräsentieren, als neues Gewinnbudget gespeichert (Zeile 12). Dies wird so lange wiederholt, bis die To-do-Liste keine Elemente mehr enthält. Im Anschluss lässt sich dann aus der Menge an Gewinnbudgets der Startposition erörtern, welche Gleichheitsbegriffe auf die zwei

Input: $\mathfrak{G} = (Z, Z_V, \rightarrowtail, g)$

$GB_A \coloneqq [z \mapsto \{\} \mid z \in Z]$
todo $\coloneqq \{z \in Z_V \mid z \not\rightarrowtail\}$
while todo $\neq \varnothing$ **do**
  $z \coloneqq$ beliebiges $x \in$ todo
  todo $\coloneqq$ todo $\setminus\{z\}$
  if $z \in Z \setminus Z_V$ **then**
    neuesGB_A $\coloneqq$min$(GB_A[z] \cup \{\text{upd}^{-1}(e', u) \mid z \overset{u}{\rightarrowtail} z' \wedge e' \in GB_A[z']\})$
  else
    verteidigungsFolgepositionen $\coloneqq \{z' \mid z \overset{u}{\rightarrowtail} z'\}$
    optionen $\coloneqq \{(z', \text{upd}^{-1}(e', u)) \mid z \overset{u}{\rightarrowtail} z' \wedge e' \in GB_A[z']\}$
    if verteidigungsFolgepositionen $\subseteq$ dom(optionen) **then**
      neues$GB_A \coloneqq \min(\{\sup_{z' \in \text{verteidigungsFolgepositionen}} GSF_A(z') \mid GSF_A \in (Z \to (\mathbb{N} \cup \infty)^6) \wedge \forall z'.\ GSF_A(z') \in \text{optionen}(z')\})$
    else
      neues$GB_A \coloneqq \varnothing$
  if neues$GB_A \neq GB_A[z]$ **then**
    $GB_A[z] \coloneqq$ neuesGB_A
    todo $\coloneqq$ todo $\cup\{z_{\text{vor}} \mid \exists u.\ z_{\text{vor}} \overset{u}{\rightarrowtail} z\}$
$GB_A^{min} \coloneqq GB_A$
return GB_A^{min}

Algorithmus 1: Algorithmus zur Berechnung der minimalen Gewinnbudgets des Angriffs, übernommen und angepasst aus [1]

im Input deklarierten Prozesse zutreffen.

8 Implementierung der Spectroscopy in CAAL

Dieses Kapitel beschäftigt sich mit dem praktischen Teil der Arbeit, der Implementierung des Spectroscopy Energy Games und des zugehörigen Algorithmus zur Lösung des Energiespiels. Es beginnt mit einer Ermittlung der Ausgangssituation bezüglich CAAL, um im Anschluss auf die Neuerungen der erweiterten Version, die im Zuge dieser Arbeit entstanden ist, einzugehen. Hierfür werden Designüberlegungen im Kontext der User Experience sowie die Code-Architektur erörtert. Die vollendete Erweiterung von CAAL ist unter `https://fabian-o01.github.io/CAAL/` verfügbar.

8.1 Ausgangspunkt: Funktionalitäten der ursprünglichen CAAL-Anwendung

Die aktuelle Version von CAAL ist unter `https://caal.cs.aau.dk/` aufzufinden und bietet die Möglichkeit, Prozesse zu modellieren, zu visualisieren und zu verifizieren sowie zwei Logikspiele zu spielen. Für jede dieser vier Aktivitäten existiert ein View. Ausgangspunkt ist hierbei der „Edit“-View (Abbildung 8), in welchem sich Prozesse unter Benutzung von CCS definieren lassen. Alternativ besteht auch die Option, über den „Project“-Button in ein Dropdown-Menü zu gelangen, das das Laden von integrierten Beispielprozessen sowie bereits vom User zu einem früheren Zeitpunkt gespeicherten Prozessen ermöglicht.

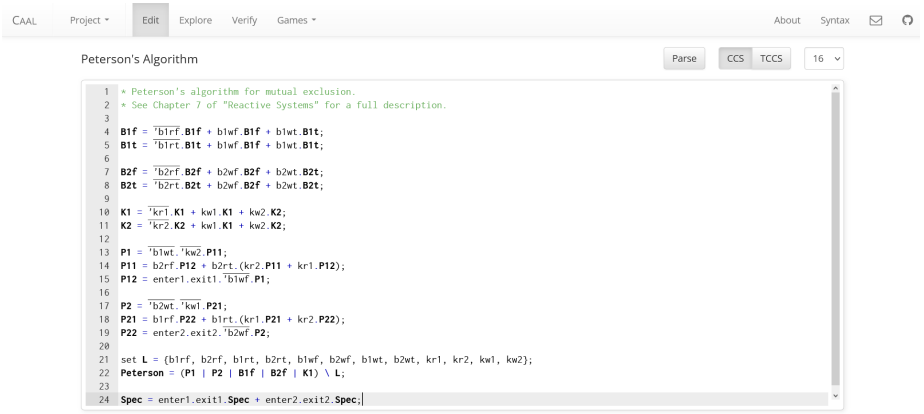

Abbildung 8: „Edit“-View mit einem der integrierten Beispiele

Ist eine korrekt definierte Prozessmenge im Editor modelliert, so lassen sich auch die drei anderen Views auswählen. Im „Explore“-View (Abbildung 9)lassen sich die verschiedenen benannten Prozesse auswählen, um diese in Graphform zu visualisieren. Nicht-benannte Prozesse bzw. Unterprozesse werden im Graphen nicht über ihre CCS-Definition, sondern als Nummern angezeigt, sind aber mit einem Hover-Effekt versehen, der ebenjene CCS-Definiton auf Anfrage angibt. Im unteren Teil sind zudem ausführbare Transitionen angegeben, welche per Klick ausgeführt werden können, um durch farbliche Markierungen ein Gefühl für das Transitionssystem zu bekommen.

Im „Verify“-View (Abbildung 10)lassen sich Queries formulieren. Die Queries sind in zwei Arten aufzuteilen: Zum einen Anfragen auf Gleichheitsbegriffe

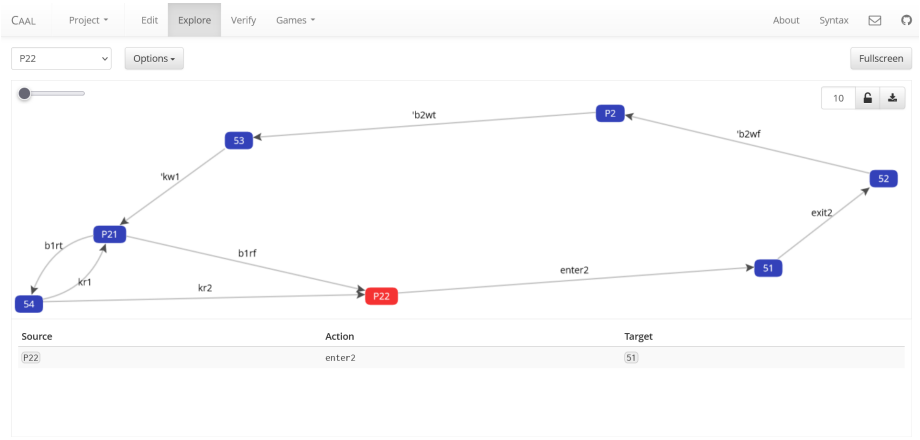

Abbildung 9: „Explore“-View

zwischen zwei Prozessen und zum anderen auf Erfüllung einer HML-Formel von einem Prozess. Unterstützt werden in sowohl schwacher als auch starker Version die Gleichheitsbegriffe Bisimulation, Trace-Inklusion, Simulation und für die letzen beiden jeweils auch die zugehörigen Äquivalenzrelationen. Ist mindestens eine Query formuliert, lassen sich diese entweder individuell oder auch durch einen „Verify All“-Button automatisch nacheinander verifizieren, indem ein Symbol darstellt, ob die Anfrage positiv oder negativ ist. Zusätzlich wird die Dauer der Berechnung angezeigt und im Falle, dass zwei Prozesse nicht in der Relation des abgefragten Gleichheitsbegriffs stehen, lässt sich auch die unterscheidende Formel hierzu anzeigen. Schließlich besteht bei Queries zu HML-Formeln sowie Bisimulation und Simulation die Option, in den „Games“-View zu wechseln, um im zugehörigen Spiel bereits die respektive Konfiguration gesetzt zu haben.

Der „Games“-View setzt sich aus den Spielen zu Bisimulation und Simulation (Abbildung 11) sowie einem Spiel zur schrittweisen Verifikation einer HML-Formel ausgehend von einem Prozess zusammen. Diese Spiele lassen sich nicht von mehreren Usern, sondern nur gegen einen computergesteuerten Gegner spielen. Die Spielkonfiguration lässt sich in einer oberen Leiste auswählen. Mittig ist das dem ausgewählten Prozess zugehörige LTS ähnlich wie im „Explore“-View dargestellt, links unten führt ein Game Log Protokoll über die rechts davon ausgewählten Spielzüge und deren Auswirkungen.

Abbildung 10: „Verify“-View mit Queries zu Trace-Äquivalenz, Bisimulation und einer HML-Formel auf den vordefinierten Prozess „Peterson“

8.2 Integration der Spectroscopy in CAAL

Die unter `https://fabian-o01.github.io/CAAL/` aufzufindende Version von CAAL stellt den praktischen Kern dieser Arbeit dar. Dort sind die in Kapitel 7 vorgestellten Konzepte umgesetzt und zur interaktiven Benutzung integriert. Die Kontribution lässt sich in zwei Teile aufteilen. Im „Games“-View existiert ein neues Spiel, das „Spectroscopy Energy Game“ (Abbildung 13) und im „Verify“-View (Abbildung 12) steht die neuartige Option der Verifikation zur Verfügung. Möchte man dort eine Query zu Gleichheitsbegriffen erstellen, so beinhaltet die Menge an zur Auswahl stehenden Relationen nicht mehr nur Bisimulation, Trace-Inklusion, Simulation und die korrespondierenden Äquivalenzrelationen, sondern zusätzlich auch alle weiteren Präordnungen aus dem Linear-Time-Branching-Time-Spektrum 4. Mit zehn weiteren Gleichheitsbegriffen ist CAAL nun folglich deutlich potenter in Bezug auf die Vielfalt an Verifikationsmöglichkeiten. Auch für alle Queries zu den neu unterstützen Relationen ist es möglich, sich unterscheidende Formeln zweier Prozesse generieren zu lassen, sofern diese existieren, und in den „Games“-View zu wechseln, um direkt die entsprechende Konfiguration für das Spectroscopy Energy Game eingestellt zu haben. Um von dem Vorteil Gebrauch zu machen, alle Präordnungen des Linear-Time-Branching-Time-Spektrums mit einer Berechnung verifizieren zu können, existiert im Menü zur Erstellung einer Query zusätzlich die Option

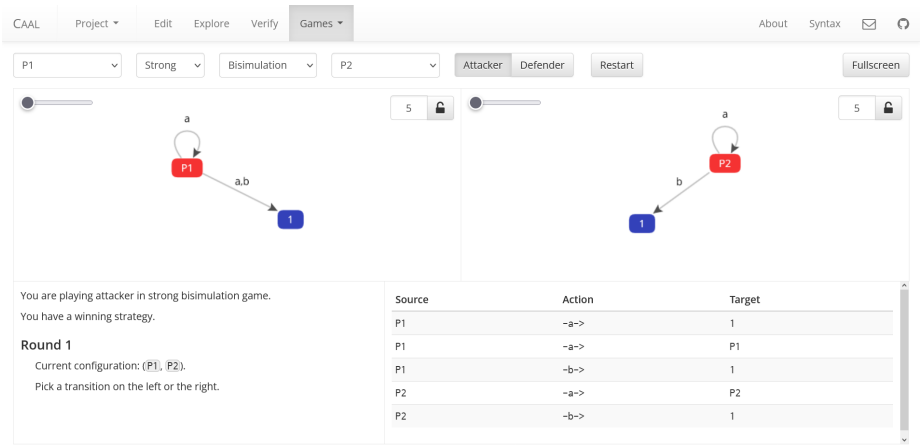

Abbildung 11: „Games“-View mit Spiel zu Bisimulation zwischen den Prozessen $P1$ und $P2$

„BJN-Algorithm“. BJN steht hierbei für Bisping, Jansen und Nestmann, die Autoren jenes Papers [6], welches die Grundidee für einen Algorithmus vorstellt, der mehrere Gleichheitsbegriffe auf einmal verifizieren kann. Wählt man nun „BJN-Algorithm“ aus, so wird für jeden Gleichheitsbegriff automatisch eine Query erstellt. Diese lassen sich zwar auch individuell abarbeiten, mit Betätigung des Buttons „Verify with BJN-Algorithm“ können aber stattdessen alle automatisch erstellten Queries durch eine Ausführung des Algorithmus zur Lösung des Spectroscopy Energy Games bearbeitet werden. Ein weiteres hinzugefügtes Feature zur Benutzungsfreundlichkeit ist der „Delete all“-Button, um alle existierenden Queries mit nur einem Klick statt für jede Query einzeln zu löschen.

Das über das Dropdown-Menü des „Games“-Views auswählbare Spectroscopy Energy Game ist in Bezug auf die Benutzeroberfläche ähnlich wie die beiden anderen Logikspiele aufgebaut. In einer oberen Leiste ist die Spielkonfiguration einstellbar, zusätzlich wird das Energiebudget angezeigt, das zum jeweiligen Spielstatus für den Angriff zur Verfügung steht. Unten rechts werden alle verfügbaren Spielzüge in Tabellenform aufgelistet. Die erste Spalte „Source“ steht für die Ausgangsposition und die dritte Spalte „Target“ für die Spielposition, die das Resultat des Spielzuges ist. Die mittige Spalte „Update“ beschreibt die Kosten des Spielzuges. Mittig werden links und rechts die Prozessgraphen angezeigt, die jeweils zu den Prozessen der aktuellen Spielposition gehören. Da das Spectroscopy Energy Game für die „rechte Seite“ einer Spielposition statt

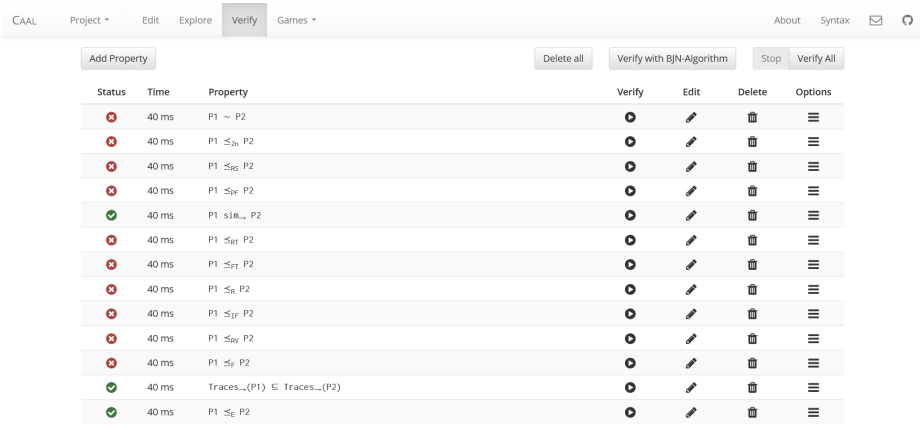

Abbildung 12: „Verify“-View des erweiterten CAAL mit automatisch erstellten Queries. Die Dauer der Verifikation ist bei allen Queries gleich, da für alle Verifikationsinstanzen nur ein Durchlauf des Algorithmus benötigt wird.

einem Prozess nicht nur eine Prozessmenge, sondern gleich zwei Mengen erlaubt, sind alle Knoten der Prozessgraphen mit folgender Farbcodierung versehen:

Seien gemäß 19 $(p, Q)_A$, $(p, Q, Q_*)_V$ und $(p, q)^{\wedge}_A$ die drei Arten von zulässigen Spielpositionen. Die farbliche Markierung ist

- blau für Prozesse, die nicht Teil der aktuellen Spielposition sind und
- rot für jeden Prozess p aller drei Arten von Spielpositionen,
- grün für alle Prozesse $q \in Q$ aus $(p, Q)_A$ und $(p, Q, Q_*)_V$,
- gelb für alle Prozesse $q_* \in Q_*$ aus $(p, Q, Q_*)_V$ und
- ebenfalls grün für alle Prozesse q aus $(p, q)^{\wedge}_A$, da sich diese gegenseitig mit den $q \in Q$ aus $(p, Q, Q_*)_V$ ausschließen.

Auch das Spectroscopy Energy Game wird gegen einen computergesteuerten Gegner gespielt. Nach Auswahl der Startposition, der Relation und der einzunehmenden Rolle beginnt das Spiel damit, dass im Game Log, welches sich unten links befindet, angezeigt wird, welche Partei eine Gewinnstrategie hat. Sollte der User eine Gewinnstrategie haben und einen Spielzug durchführen, der nicht Teil dieser Gewinnstrategie war, so wird dies auch im Game Log dokumentiert.

Die KI ist dahingegen so eingestellt, dass sie nur Spielzüge ausführt, die Teil der Gewinnstrategie sind, sofern diese gegeben ist. Da Spieldurchläufe unendlicher Länge in der Praxis nicht umsetzbar sind, gewinnt im Falle eines Zyklus bzw. des Besuchens einer bereits besuchten Spielposition immer die Verteidigung. Die Überlegung hierbei ist, dass es die Aufgabe des Angriffs ist, möglichst zielstrebig unter Einhaltung des Energiebudgets in eine Spielposition zu navigieren, in der die Verteidigung keine Spielzüge mehr zur Verfügung hat und somit verliert. Werden dabei Umwege genommen, die möglicherweise zu Zyklen führen, so hat der Angriff seine Aufgabe verfehlt. Dies ist jedoch nicht derart zu interpretieren, dass im Falle einer Niederlage des Angriffs aufgrund eines Zyklus die Relation zutreffen muss. Selbst eine Gewinnstrategie kann einen Zyklus beinhalten, jedoch existiert dann auch immer eine Gewinnstrategie, die diesen Zyklus auslässt und daher aufgrund der Monotonie der Updatefunktion maximal gleich viel des Energiebudgets verbraucht. Besonders deutlich wird dies bei einem Spieldurchlauf, in dem um Bisimulation gespielt wird, da dort ein beliebig großes Energiebudget zur Verfügung steht. Die Kosten der einzelnen Spielzüge sind daher irrelevant und es geht lediglich darum, ob der Angriff es schafft, eine Spielposition zu erreichen, in der die Verteidigung keinen ausführbaren Spielzug besitzt. Eine endliche Detour, die Zyklen beinhaltet, kann also trotzdem Teil einer Gewinnstrategie sein, wenn am Ende die beschriebene Spielposition erreicht wird.

8.3 Designüberlegungen

Leitgedanke zum Design der Erweiterungen zu CAAL war es, sich möglichst nah an den bestehenden, von den ursprünglichen Entwickler*innen getroffenen Designentscheidungen zu orientieren. Dies hat den Vorteil, dass die gesamte Anwendung konsistent und kohärent wirkt und die Erweiterungen nicht wie unpassende Nebenfunktionalitäten wirken. Zudem ermöglicht es dieser Ansatz, viele der vorhandenen Strukturen aus der originalen Codebasis wiederzuverwenden. Um die unterschiedlichen Verifikationsmethoden in gleicher Form darstellen zu können, war jedoch ein Maß an Kreativität zur Problemlösung erforderlich. Das ursprüngliche CAAL erlaubt das Erstellen einzelner Queries, die unabhängig voneinander bearbeitet werden. Kernfunktionalität der Erweiterung ist es hingegen, 13 Gleichheitsbegriffe zusammen zu verifizieren. Die Lösung, durch Auswahl des „BJN-Algorithm“ zwar mehrere Queries automatisch zu generieren, diese dann aber durch das Betätigen eines zusätzlichen Buttons zusammen zu verifizieren,

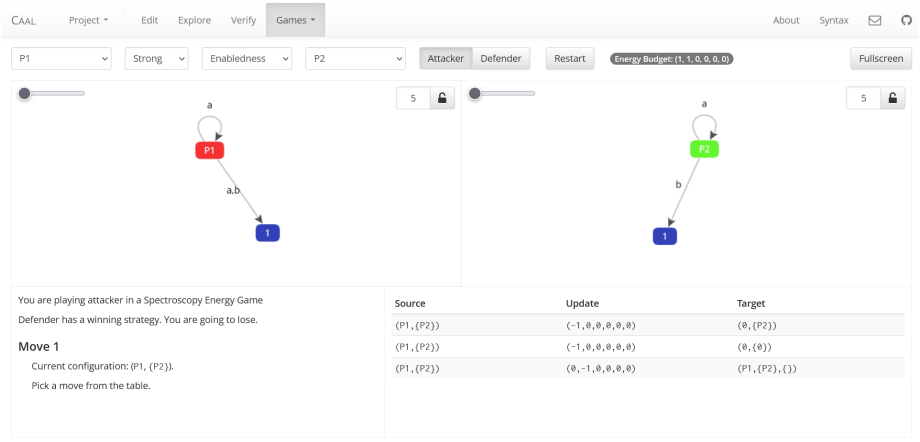

Abbildung 13: Das Interface des Spectroscopy Energy Games. Aufgabe des Users ist es in dieser Konfiguration, als Angriff spielerisch eine unterscheidende Formel aufzubauen, die an quantifizierter Ausdrucksstärke nicht das Budget $(1, 1, 0, 0, 0, 0)$ überschreitet.

vereinigt das Originaldesign mit der Natur der neuen Verifikationsmethode. Da Gleichheitsbegriffe weiterhin als einzelne Queries dargestellt werden, können diese auch wie in der ursprünglichen Version einzeln verifiziert werden und im Anschluss lassen sich auch einzeln unterscheidende Formeln generieren sowie mit entsprechender Konfiguration in das Spectroscopy Energy Game wechseln. Da eine Unterstützung von auch zusätzlich allen Äquivalenzrelationen zu insgesamt nicht 10, sondern 20 neuen Relationen und somit zu insgesamt 25 Optionen bei der Erstellung einer Query führen würde, wurde die Entscheidung getroffen, nur die Präordnungen zu integrieren. Dies führt jedoch keinesfalls zu einem Verlust an Mächtigkeit des Verifikationstools, da zum Prüfen einer Äquivalenzrelation lediglich das Prüfen auf die entsprechende Präordung in beide Richtungen erforderlich ist.

Da das Spectroscopy Energy Game deutlich komplexer als das bereits vorhandene „Equivalence Game“ ist, spielt das Design eine wichtige Rolle dabei, ebenjenes Spiel dem User möglichst anschaulich zu vermitteln. Durch die Verwendung gleicher UI-Elemente sowie gleicher Anordung dieser, kann sofort eine Intuition zur Funktionsweise des neuen Spiels gewonnen werden. Visuell unterscheidet sich aufgrund dessen das Spectroscopy Energy Game vom Equivalence Game nur darin, dass die farblichen Markierungen komplexer sind und die

Spielzüge statt Transitionen eines LTS nun von einer Spielposition zur nächsten führen und einen Updatevektor besitzen.

8.4 Code-Architektur

Die Codebasis von CAAL, aufzufinden unter `https://github.com/CAAL/CAAL`, besteht neben der Strukturierung durch HTML und einigen CSS-Stylesheets überwiegend aus TypeScript, das zu JavaScript kompiliert wird. Dies geschieht auf Basis von ECMAScript 5 (ES5), welches im Jahr 2009 veröffentlicht wurde. Die im Zuge dieser Arbeit erweiterte Version benutzt den ES2015-Standard. Dieser ist eine Obermenge von ES5 und unterstützt infolgedessen alle Code-Praktiken des bestehenden Codes, erlaubt aber zusätzlich die Anpassung an modernere Code-Konventionen, die sich seitdem entwickelt haben. Beispiele hierfür sind die Verwendung von Arrow-Funktionen, Variablendefinitionen durch `let` und `const`, `for...of`-Schleifen, Spread-Operatoren und eine Erweiterung des Funktionsumfanges zur Handhabung und Manipulation von Arrays [7]. Die Codebasis der Erweiterung von CAAL ist unter `https://github.com/Fabian-001/CAAL` verfügbar.

Abbildung 14 stellt eine stark vereinfachte Version des Dependency Graphs zur Struktur von CAAL dar, um die für diese Arbeit relevanten Zusammenhänge innerhalb der Code-Struktur zu veranschaulichen und den groben Verlauf der Applikation zu vermitteln. In `main.ts` wird die Klasse `ActivityHandler` aus dem gleichnamingen TypeScript-Dokument instanziiert, um durch deren Methode `addActivity` die verschiedenen Views aufzubauen. Diese erben von der Elternklasse `Activity` aus `activity.ts`, welche nicht selbst instanziiert wird. Die neue Datei `segame.ts` beinhaltet die Spiellogik des Spectroscopy Energy Games und verwendet in großen Teilen ähnliche Klassen und Funktionen wie `game.ts`. In `verifier.ts` des `activity`-Ordners wurde zur Integration der neuen Verifikationsmethode auf Code-Ebene die Funktionalität des Sammelns von Queries eingefügt, um diese dann gleichzeitig durch den BJN-Algorithmus verifizieren zu lassen. Zur Abarbeitung von Queries wird eine Anfrage an einen Worker gesendet, der in `verifier.ts` des „workers“-Ordners zu finden ist. In diesen Worker wurden für jede neu unterstützte Präordnung jeweils eine zugehörige Funktion eingefügt, die sich um die weitere Bearbeitung kümmert, sowie eine Funktion zur Verifikation des gesamten Spektrums an Gleichheitsbegriffen. Diese Funktionen greifen auf das BJN-Modul der neuen Datei `BJN.ts` zu. Dort wird der Spielgraph des Spectroscopy Energy Games erstellt, um im

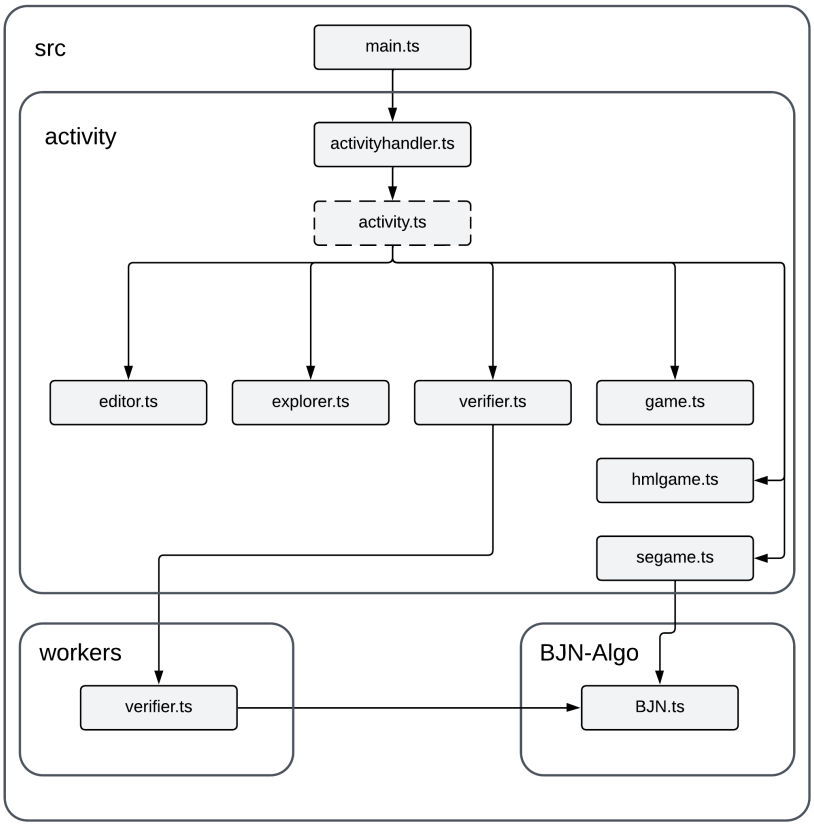

Abbildung 14: Simplifizierte Version des Dependency Graphs zur Struktur von CAAL. Große Umrandungen stellen Ordner da.

Anschluss die Gewinnbudgets des Angriffs durch den Algorithmus zu berechnen. Schließlich wird, wie in 7.2 beschrieben, über die Gewinnbudgets gefolgert, welche Gleichheitsbegriffe zutreffen. Sollte eine Relation nicht zutreffen, so wird die zur Laufzeit des Algorithmus mitgenerierte unterscheidende Formel extrahiert und in eine logisch äquivalente Formel der von CAAL verwendeten HML-Grammatik umgewandelt. Zuletzt wird das gesamte Ergebnis der bearbeiteten Query zurückgesendet.

Im Code zum interaktiven Spielen des Spectroscopy Energy Games wird das BJN-Modul ebenfalls verwendet. Zu Beginn des Spiels wird der gesamte Spielgraph generiert, um im Laufe des Spiels alle möglichen Spielzüge von einer Ausgangsposition ausgeben zu lassen. Diese Information ist zum einen für die Auswahl an Spielzügen für den User notwendig, zum anderen auch für die KI,

die den bestmöglichen Spielzug berechnet.

9 Fazit

Abschließend lässt sich feststellen, dass diese Arbeit zwei übergeordnete Herausforderungen mit sich gebracht hat: Auf der Designebene bestand die Aufgabe darin, die theoretischen Ansätze von CAAL mit dem Prinzip der Spectroscopy auf solch eine Weise zu vereinen, dass die entstandene Applikation dennoch nicht dissonant wirkt. Aus technischer Perspektive musste es gelingen, Datenstrukturen und Funktionen so anzupassen, dass sie wiederverwendet werden können, und Schnittstellen zu erschaffen, die die Spectroscopy mit CAAL kompatibel machen. Bei der Aufgabe der Vereinigung beider Prinzipe stößt man jedoch auf einen Designkonflikt: Das User Interface des Spectroscopy Energy Games sollte zwar aufgrund besserer Intuition und Kohärenz den beiden anderen Logikspielen von CAAL nah sein, jedoch besitzt das Spectroscopy Energy Game mehr Parameter. Es ist also komplexer und benötigt für eine gute Übersicht ein Interface, das mehr Informationen anzeigen kann. Konkret wäre es hilfreich, nicht nur den Spielzug in einer Liste an Auswahlmöglichkeiten anzuzeigen, sondern auch zu vermitteln, was dieser Spielzug auf theoretischer Ebene bedeutet und inwiefern dadurch die unterscheidende Formel gebildet wird.

Weitere Verbesserungsmöglichkeiten betreffen die Verifikation unter Benutzung des BJN-Algorithmus. Zurzeit unterstützt diese mit Ausnahme der Bisimulation sowohl nur Präordnungen als auch nur LTS mit starken Transitionen. Wie in 8.3 erläutert, ist ersteres durch die Übersichtlichkeit begründet. Eine Lösung hierfür könnte das Hinzufügen eines zusätzlichen Schaltfelds bei der Queryerstellung sein, das für jeden Gleichheitsbegriff die Einstellung ermöglicht, ob die Relation symmetrisch sein soll, um direkt auf Äquivalenzrelationen zu prüfen statt zwei Queries zu erstellen. Dies würde jedoch nicht nur eine UI-Veränderung erfordern, sondern auch gegen die Designentscheidung von CAAL verstoßen, Präordnungen und Äquivalenzrelationen einzeln und unabhängig voneinander bei der Queryerstellung anzuzeigen. Für die Unterstützung von schwachen Transitionen existiert ein Verfahren namens „Weak-Step-Saturation“, infolgedessen das zugrundeliegende LTS auf bestimmte Weise verändert wird. Die Verifikation auf dem abgewandelten LTS ist jedoch nicht für alle Gleichheitsbegriffe des Linear-Time-Branching-Time-Spektrums gleichzusetzen mit der Verifikation der jeweiligen schwachen Version dieser Gleichheitsbegriffe auf dem ursprünglichen

LTS.

Trotz dieser Verbesserungsmöglichkeiten bietet die erweiterte Version von CAAL eine funktionsfähige Integration der Spectroscopy mit einer Vielzahl neuer Gleichheitsbegriffe, effizienterer Verifikation und Veranschaulichung durch ein neues Logikspiel.

Literatur

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

[2] Luca Aceto u. a. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, 2007. DOI: 10.1017/CBO9780511814105.

[3] Robin Milner. *A Calculus of Communicating Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1980. ISBN: 978-3-540-38311-6. DOI: 10.1007/3-540-10235-3.

[4] R.J. van Glabbeek. "The Linear Time - Branching Time Spectrum I." In: *Handbook of Process Algebra*. Hrsg. von J.A. Bergstra, A. Ponse und S.A. Smolka. Amsterdam: Elsevier Science, 2001, S. 3–99. ISBN: 978-0-444-82830-9. DOI: 10.1016/B978-044482830-9/50019-9.

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

[6] Benjamin Bisping, David N. Jansen und 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 (2022). DOI: 10.46298/lmcs-18(3:19)2022.

[7] Allen Wirfs-Brock und Brendan Eich. "JavaScript: The First 20 Years". In: *Proc. ACM Program. Lang.* 4.HOPL (2020). DOI: 10.1145/3386327.