# Erklärung der Urheberschaft

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, _____ _____

Unterschrift


***Technische Universität Berlin***
Fakultät IV – Elektrotechnik und Informatik
Modelle und Theorie Verteilter Systeme

Bachelorarbeit

# Automatisierte Reduktion von reaktiver zu starker Bisimilarität

Zead Alshukairi
Matrikel-Nr. 402861

Betreuer: Benjamin Bisping
Erster Prüfer: Prof. Dr. Uwe Nestmann
Zweiter Prüfer: Prof. Dr Stephan Kreutzer

Berlin
November 2022

# Abstrakt

Diese Arbeit stellt ein Werkzeug vor, das eine reaktive Bisimilarität [vG20] auf beschrifteten Transitionssysteme mit Timeouts ($LTS_t$) [vG21a] überprüft.

Das Werkzeug ist anhand einer von mir automatisierten Reduktion von reaktiver zu starker Bisimilarität [Poh21] und eine Tool-Set vom mCRL2-Projekt [mCR19] entwickelt.

Es können ein Paar oder alle Paare von Prozessen des eingegebenen $LTS_t$ auf eine reaktive Bisimilarität überprüft werden. Außerdem ist in dieser Arbeit eine Verbesserungsmöglichkeit bezüglich der Komplexität des Algorithmus der Reduktion implementiert. Die Komplexität ist vom exponentiellen Aufwand auf linearen vermindert. Die Verbesserung des Algorithmus ermöglicht eine schnellere und effiziente Überprüfung von großen Systemen, bei denen die ursprüngliche Version nur mit sehr langer Ausführungszeit auskommt.

# Inhaltsverzeichnis

# 1 Einführung

Verschiedene Äquivalenzbegriffe wie starke Bisimilarität [AILS07, S. 42–61] und reaktive Bisimilarität [vG20, S. 3–6] können auf beschriftete Transitionssysteme [AILS07, S. 20–25] angewandt werden.

Bisimilaritätsmodelle, die keine Timeouts definieren, wie starke und schwache Bisimilarität [AILS07, 3.4 Weak bisimilarity], werden in beschrifteten Transitionssysteme ohne Timeouts ($LTS_s$) [AILS07, S. 20–25] benutzt.

Reaktive Bisimilarität ist jedoch ein Bisimilaritätsmodell, welches Timeouts benötigt und wird deswegen auf beschrifteten Transitionssystemen mit Timeouts ($LTS_t$) [vG21a] verwendet.

Ein beschriftetes Transitionssystem kann ein System mit dessen Aktionen zwischen Prozessen modellieren [Poh21, S. 8] und es kann darauf verschiedene Äquivalenzbegriffe angewandt werden, somit ist die Existenz von Algorithmen für die verschiedenen Äquivalenzbegriffe auf solche Systemen von Bedeutung. Damit bei Test- oder experimentellen Zwecke auf den theoretischen Vergleich verzichtet und den damit verbundenen Zeitverlust gespart werden kann.

Es existieren mehrere Algorithmen und Tools, die verschiedene Äquivalenzbegriffe behandeln. Ein Beispiel dafür ist das mCRL2-Projekt [mCR19], wo verschiedene Algorithmen von z.B. starker und schwacher Bisimilarität implementiert bzw. automatisiert wurden. Diese Algorithmen können auf Transitionssystemen ausgeführt werden, die im mCRL2 in einem Spezifikationsformat [mCR19, S. 23–25] geschrieben sind.

Der neue Äquivalenzbegriff reaktive Bisimilarität ist ein wichtiges Bisimilaritätsmodell. Denn dort werden Systeme behandelt, deren Implementierung Zeitüberschreitungen (Timeouts) enthalten [vG20, S. 3–6]. Diese können aber von anderen Bisimilaritätsmodelle wie starker Bisimilarität nicht richtig verglichen werden, da solche ein System in Prozess-Algebra wie CCS ohne Timeouts [AILS07, S. 9–35] nicht korrekt modelliert werden kann [vG20, S. 35].

Gegenseitiger Ausschluss (mutual exclusion) [Lam19a] und [Lam19b] ist ein Problem, das zwischen parallelen Prozessen bestehen kann, wenn mindestens zwei Prozesse versuchen einen kritischen Bereich zu betreten. Ein kritischer Bereich ist ein Code-Abschnitt, wo eine geteilte Variable überschrieben werden soll. Solche Probleme werden gelöst, indem nur ein Prozess diesen kritischen Code-Abschnitt betreten gelassen wird. Alle andere Prozesse, die auch diesen Code-Abschnitt betreten möchten, müssen **warten**.

Genau hier kann reaktive Bisimilarität ihre Rolle bei dem Vergleich spielen. Dank [vG20, S. 3–6] können solche Systeme verglichen werden. Ein Algorithmus ist benötigt, der diese Aufgabe erledigt, um den Vergleich automatisiert auszuführen.

An dieser Stelle ist die Arbeit von POHLMANN [Poh21] hervorzuheben, wo eine Reduktion definiert wird, die Systeme von reaktiver zu starker Bisimilarität reduziert. Das heißt, zwei Prozesse können auf starke Bisimilarität anstatt auf reaktive Bisimilarität überprüft werden, sodass das Theorem 1 gilt.

Diese Reduktion ist algorithmisch und kann automatisiert werden.

In dieser Arbeit ist ein Werkzeug entwickelt, das eine reaktive Bisimilarität [vG20, S. 3–6] durch die oben erwähnte in Java automatisierte Reduktion und eine Tool-Set vom mCRL2-Projekt überprüft.

In meinem Werkzeug ist die Reduktion zuerst roh implementiert und dann verbessert. Der Benutzer kann auswählen, ob der ursprüngliche Algorithmus

oder die verbesserte Version zur Überprüfung ausgeführt werden soll. Es können auch ein Paar oder alle Paare von Prozessen auf reaktive Bisimilarität überprüft werden.

Diese Arbeit ist wie folgt strukturiert:
Im Abschnitt Hintergrund sind beschriftete Transitionssysteme, starke und reaktive Bisimilarität, mCRL2 und die Reduktion von reaktiver zu starker Bisimilarität vorgestellt. Bei vertrauten Kenntnisse in diesen Themen kann diesen Abschnitt übersprungen werden. Sonst ist es empfehlenswert, diesen Abschnitt zu lesen. Der Abschnitt Ansatz zeigt, wie mein Werkzeug aufgebaut ist, die Arbeitsweise des ursprünglichen Algorithmus und die verbesserte Version. Im Abschnitt Evaluation ist dann zu erkennen, wie effektiv die verbesserte Version des Algorithmus ist. Dort ist den Unterschied zwischen dem ursprünglichen und verbesserten Algorithmus gezeigt. Der Abschnitt Verwendungsweise erklärt, wie mein Werkzeug anzuwenden ist. Am Ende ist im Abschnitt Fazit das allgemeine Ergebnis sowie mögliche, zukünftige, mit meinem Thema verwandte Arbeiten diskutiert.

# 2 Hintergrund

## 2.1 Beschriftete Transitionssysteme

Ein beschriftetes Transitionssystem [AILS07, S. 20–25] besteht aus Zuständen und Aktionen. Diese Aktionen passieren, wenn eine Transition zwischen zwei Zuständen genommen wird. Alle Transitionen sind mit Aktionen aus einer gegebenen Menge $A$ beschriftet. Die Zustände können auch benannt werden.
Die folgende Definition ist aus [AILS07, Definition 2.1].

### 2.1.1 Definition

Ein beschriftetes Transitionssystem $LTS$ ist ein Tripel
$(Proc, Act, \{\xrightarrow{\alpha} \mid \alpha \in Act\})$, mit

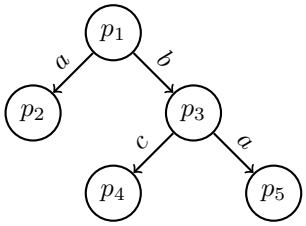
- $Proc$ : Menge der Zustände.
- $Act$ : Menge der Aktionen.
- $\xrightarrow{\alpha} \in Proc \times Proc$, für alle $\alpha \in Act$.

Dass eine Aktion $a \in Act$ zwischen Prozessen $p, p' \in Proc$ passiert, ist als Tripel $(p, a, p')$ oder durch die intuitive Notation $p \xrightarrow{a} p'$ darzustellen.

Ein $LTS$ kann reaktive Systeme modellieren [Poh21, S. 8], wo die starke und reaktive Bisimilarität angewandt werden kann ([AILS07, S. 42–61] und [vG20, S. 3–6]).

Gleich sind $LTS_s$ für starke Bisimilarität und $LTS_t$ für reaktive Bisimilarität vorgestellt.

### 2.1.2 Beispiel




In diesem *LTS* ist zu sehen, dass der Prozess $p_1$ zwei Aktionen nach $p_2$ und $p_3$ vornehmen kann.

## 2.2 Starke Bisimilarität

Ein normales beschriftetes Transitionssystem wird als $LTS_s$ [vG21a, S. 1] bezeichnet. Das ist ein beschriftetes Transitionssystem, welches nur die sichtbaren Aktionen in der gegebenen Menge $A$ und die Aktion $\tau$ enthalten kann. Die $\tau$ Aktion ist nicht beobachtbar und somit gehört sie nicht zu der Menge $A$.

Eine starke Bisimilarität [AILS07, S. 42–61] ist auf Zuständen eines gegebenen $LTS_s$ definiert und ist die feinste Verhaltensäquivalenz auf Transitionssystemen [Poh21, Strong Bisimilarity]. Dieser Begriff betrachtet alle Aktionen in der gegebenen Menge $A$ und auch die versteckte Aktion $\tau$.

Die folgende Definition ist aus [AILS07, Definition 3.2].

### 2.2.1 Definition

Sei:

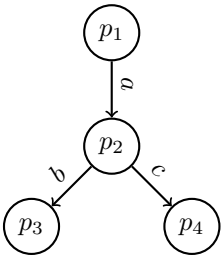
- $LTS_s$: Ein normales beschriftetes Transitionssystem.
- $Act$: Die gegebene Menge der Aktionen im $LTS_s$.
- $z_i, z_i'$: Zustände in einem $LTS_s$, mit $i \in \mathbb{N}$.

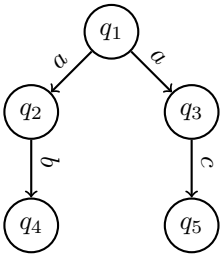
Eine starke Bisimulation ist eine binäre Relation $\mathfrak{R}$ auf einer Menge von Zuständen in einem $LTS_s$, wenn es für alle $z_1 \ \mathfrak{R} \ z_2$ eine Aktion $\alpha$ gibt, mit den Eigenschaften:

- Wenn $z_1 \xrightarrow{\alpha} z_1'$, dann gibt es eine Transition $z_2 \xrightarrow{\alpha} z_2'$ und $z_1' \ \mathfrak{R} \ z_2'$.
- Wenn $z_2 \xrightarrow{\alpha} z_2'$, dann gibt es eine Transition $z_1 \xrightarrow{\alpha} z_1'$ und $z_1' \ \mathfrak{R} \ z_2'$.

Zwei Zustände $z_1$ und $z_2$ sind stark bisimilar, geschrieben $z_1 \sim z_2$, wenn sie in einer starken Bisimulation enthalten sind.

### 2.2.2 Beispiele

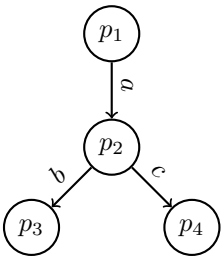


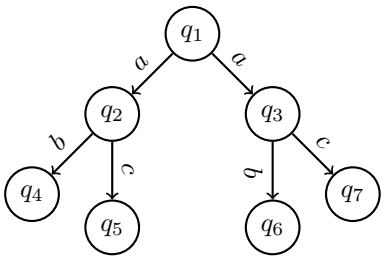




Die Prozesse $p_1$ und $q_1$ sind nicht stark bisimilar, weil nach der Ausführung der Aktion $a$ ausgehend von $p_1$ der Nachfolger-Prozess $p_2$ entscheiden kann, ob er die Aktion $c$ oder $b$ vornimmt.

Jedoch wird bei dem Prozess $q_1$ mit der nicht-deterministischen [Poh21, 8] Ausführung der Aktion $a$ entschieden, was der Nachfolger-Prozess ausführen wird.

Somit kann der einzige von $p_1$-Nachfolger-Prozess $p_2$ die Aktion $c$ ausführen, wenn der Prozess $q_1$ die Aktion $a$ nach $q_2$ wählen würden und $q_2$ kann die Aktion $c$ nicht nachmachen, da er über eine solche Aktion nicht verfügt. Auf der anderen Seite wird der einzige von $p_1$-Nachfolger-Prozess $p_2$ die Aktion $b$ ausführen, wenn der Prozess $q_1$ die Aktion $a$ nach $q_3$ wählen würde und $q_3$ kann die Aktion $b$ nicht simulieren.




In diesem Beispiel sind die Prozesse $p_1$ und $q_1$ jedoch stark bisimilar, denn nach der Ausführung der $a$ Aktionen ist es bei allen Nachfolger-Prozesse von $p_1$ und $q_1$ möglich, zwischen den Aktionen $b$ und $c$ zu wählen.

## 2.3 Reaktive Bisimilarität

Ein beschriftetes Transitionssystem mit Zeitüberschreitungen (Timeouts) wird als $LTS_t$ [Poh21, S. 20] bezeichnet. Dieses Transitionssystem kann nicht nur die Aktionen in der gegebenen Menge $A$ und die Aktion $\tau$ enthalten, sondern auch eine Timeout-Aktion $t$.

Eine reaktive Bisimilarität [vG20, S. 3–6] ist auf Zuständen eines gegebenen $LTS_t$ definiert. Reaktive Bisimilarität ist eine schwächere Variante von starker Bisimilarität, da reaktive Bisimilarität in bestimmten Fällen Umgebungen haben kann, wo ein Prozess bestimmte Aktionen nicht vornehmen kann, obwohl diese tatsächlich existieren.

Die folgende Definition ist aus [vG20, Definition 1].

### 2.3.1 Definition

Sei

- $Act$: Menge von Aktionen.
- $LTS_t$: Ein Transitionssystem mit Timeouts.
- $P$: Die Menge der Prozesse in einem $LTS_t$.
- $t$: Die spezielle Timeout-Aktion.
- $A$: Eine Menge von Aktionen, mit $A \subseteq Act \setminus \{\tau, t\}$.
- $X$: Eine Menge von Aktionen, mit $X \subseteq A$.
- $a_i, b_j, ..., z_h$: Zustände in $P$ und $i, j, h \in \mathbb{N}$.
- $\mathcal{I}(p_1)$: Die initiale Aktionen von Prozess $p_1 \in P$.
- $\mathcal{P}(A)$: Die Potenzmenge von $A$.

Eine reaktive Bisimulation ist eine symmetrische Relation:

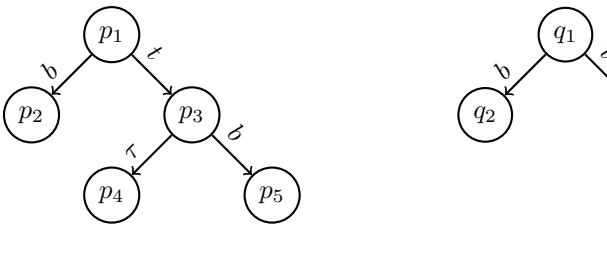
$$\mathfrak{R} \subseteq (P \times \mathcal{P}(A) \times P) \cup (P \times P)$$
Das bedeutet, dass $(p_1, X, q_1) \in \mathfrak{R} \Leftrightarrow (q_1, X, p_1) \in \mathfrak{R}$ und
$(p_1, q_1) \in \mathfrak{R} \Leftrightarrow (q_1, p_1) \in \mathfrak{R}$, mit:

- wenn $(p_1, q_1) \in \mathfrak{R}$ und $p_1 \xrightarrow{\tau} p_2$, dann existiert ein $q_2$, mit $q_1 \xrightarrow{\tau} q_2$ und $(p_2, q_2) \in \mathfrak{R}$
- wenn $(p_1, q_1) \in \mathfrak{R}$, dann $(p_1, X, q_1) \in \mathfrak{R}$ für alle $X \subseteq A$

und für alle $(p_1, X, q_1) \in \mathfrak{R}$ gilt:

- wenn $p_1 \xrightarrow{a} p_2$ mit $a \in X$, dann existiert ein $q_2$, mit $q_1 \xrightarrow{a} q_2$ und $(p_2, q_2) \in \mathfrak{R}$.
- wenn $p_1 \xrightarrow{\tau} p_2$, dann existiert ein $q_2$, mit $q_1 \xrightarrow{\tau} q_2$ und $(p_2, X, q_2) \in \mathfrak{R}$.
- wenn $\mathcal{I}(p_1) \cap (X \cup \{\tau\}) = \emptyset$, dann $(p_1, q_1) \in \mathfrak{R}$ und
- wenn $\mathcal{I}(p_1) \cap (X \cup \{\tau\}) = \emptyset$ und $p_1 \xrightarrow{t} p_2$, dann existiert $q_2$ mit $q_1 \xrightarrow{t} q_2$ und $(p_2, X, q_2) \in \mathfrak{R}$.

### 2.3.2 Beispiel




Die Unterprozesse $p_3$ und $q_3$ sind nicht reaktiv bisimilar, da $p_3$ eine Aktion $b$ ausführen kann, welche $q_3$ nicht simulieren kann. Jedoch sind die Prozesse $p_1$ und $q_1$

reaktiv bisimilar, da der t-Nachfolger $p_3$ im Vergleich zum einzigen t-Nachfolger $q_3$ sich bereits wegen der Timeout-Aktion in einer Umgebung befinden [vG20, S. 3–6], die einen $b$ Schritt verbietet. In diesem Sinne können die t-Nachfolger keine $b$ Aktion ausführen.

## 2.4 Reduktion von reaktiver zu starker Bisimilarität

Dieses Konzept [Poh21] beschäftigt sich mit der Transformation vom $LTS_t$ [vG21a] auf $LTS_s$ [AILS07, S. 20–25], sodass folgendes gilt:

**Theorem 1.** *Zwei Prozesse $p$ und $q$ sind im $LTS_t$ reaktiv bisimilar genau dann, wenn die dazugehörigen $\mathcal{U}(p)$ und $\mathcal{U}(q)$ in $LTS_s$ stark bisimilar sind [Poh21, S. 39].*

Die folgende Definition ist aus [Poh21, A Mapping for Transition Systems].

### 2.4.1 Definition

Die Transformation ist wie folgt definiert:

- $Proc$: Menge von Prozessen und $a, a', b, b', ..., z, z'$: Zustände in $Proc$.
- $Act$: Menge von Aktionen.
- $t_\varepsilon$: Spezielle Aktion.
- $t$: Spezielle Timeout-Aktion.
- $A$: Menge von Aktionen, mit $A = Act \setminus \{\tau, t\}$.
- $E_X$: Spezielle Aktion, mit $X \subseteq A$.

Es wird angenommen, dass $t_\varepsilon \notin Act$ und $\forall X \subseteq A.\ E_X \notin Act$.
Dann ist Folgendes definiert: $\mathcal{T}_\upsilon = (Proc_\upsilon, Act_\upsilon, \rightarrow_\upsilon)$, mit

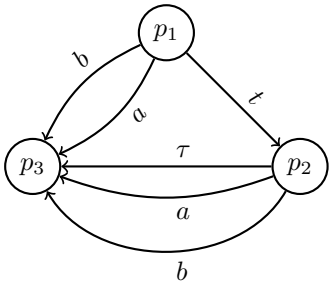
- $Proc_\upsilon = \{\mathcal{U}(p) \mid p \in Proc\} \cup \{\mathcal{U}_X(p) \mid p \in Proc \wedge X \subseteq A\}$.
- $Act_\upsilon = Act \cup \{t_\varepsilon\} \cup \{E_X \mid X \subseteq A\}$.
- und $\rightarrow_\upsilon$ ist bei den folgenden Regeln definiert:

$$1)\ \frac{p \xrightarrow{\tau} p'}{\mathcal{U}(p) \xrightarrow{\tau}_\upsilon \mathcal{U}(p')} \qquad 2)\ \frac{}{\mathcal{U}(p) \xrightarrow{E_X}_\upsilon \mathcal{U}_X(p)}\ X \subseteq A$$

$$3)\ \frac{p \xrightarrow{a} p'}{\mathcal{U}_X(p) \xrightarrow{a}_\upsilon \mathcal{U}(p')}\ a \in X \qquad 4)\ \frac{p \xrightarrow{\tau} p'}{\mathcal{U}_X(p) \xrightarrow{\tau}_\upsilon \mathcal{U}_X(p')}$$

$$5)\ \frac{p \not\xrightarrow{\alpha} \forall \alpha \in X \cup \{\tau\}}{\mathcal{U}_X(p) \xrightarrow{t_\varepsilon}_\upsilon \mathcal{U}(p)} \qquad 6)\ \frac{p \not\xrightarrow{\alpha} \forall \alpha \in X \cup \{\tau\}\ \ p \xrightarrow{t} p'}{\mathcal{U}_X(p) \xrightarrow{t}_\upsilon \mathcal{U}_X(p')}$$

### 2.4.2 Beispiel




Nach der Transformation sieht das System wie folgt aus:

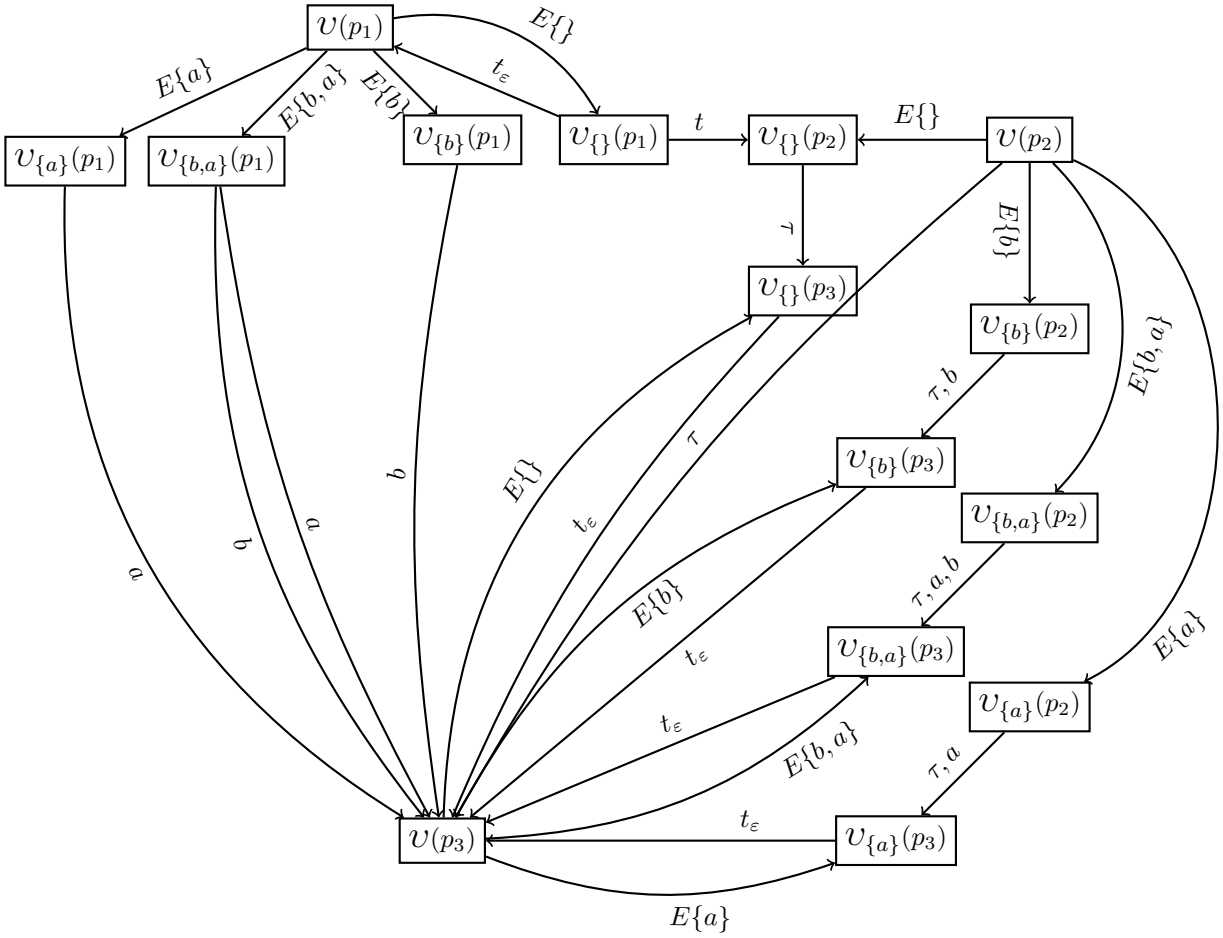



Abbildung 1: Transformiertes System

Es ist zu sehen, dass ausgehend von jedem der drei ursprünglichen Prozesse in eine Umgebung $U$ überführt wird, mit $U \subseteq \{a, b\}$. Erst danach werden die ursprünglichen Aktionen des entsprechenden Prozesses genommen. Falls diese

nicht existieren, dann verlässt das System die jeweilige Umgebung durch die Aktion $t_\varepsilon$.

Es ist auch zu sehen, dass ausgehend von $\mathcal{U}(p_1)$ der Prozess $\mathcal{U}(p_2)$ nicht erreicht werden kann, da die Timeout-Aktion in diesem Fall verursacht, eine Umgebung zu entstehen, die das verbietet, einen Zustand zu erreichen, wo eine $a$ oder $b$ Aktion möglich ist.

### 2.4.3 Funktionalität der Reduktion

Eine formale Erklärung der Funktionalität der Reduktion ist in der Arbeit von POHLMANN [Poh21, S. 33–41] zu finden. Hier ist diese Funktionalität in informeller Art und Weise erklärt, um ein einfaches Bild davon erzeugen zu können. Zuerst ist Folgendes anzunehmen:

- $Proc_\upsilon$: Ist die Menge aller Prozesse nach der Transformation.
- $Proc$: Ist die Menge aller Prozesse vor der Transformation.
- $Act_\upsilon$: Ist die Menge aller Aktionen.
- $Act$: Eine Menge von Aktionen, mit $Act = Act_\upsilon \setminus \{t_\varepsilon, E_X\}$.
- $A$: Eine Menge aller sichtbaren Aktionen, mit $A = Act \setminus \{\tau, t\}$.
- $\mathcal{I}(p)$: Ist die Menge der Aktionen von $p$ vor der Transformation.
- $E(\mathcal{U}(p))$: Ist die Menge der Aktionen von $p$ nach der Transformation.

Die Reduktion weist jedem Prozess $p \in Proc$ die Aktionsmenge $\mathcal{P}(A)$ zu, sodass $E(\mathcal{U}(p)) = \mathcal{I}(p) \cup \mathcal{P}(A)$. Die Menge $\mathcal{P}(A)$ ist als die Menge der Umgebungen zu bezeichnen. Denn nach der Ausführung von $E_X \in \mathcal{P}(A)$ landet das System in einem Prozess, der nur solche Aktionen ausführen kann, die in $X$ enthalten sind. Also landet das System in einer Umgebung, die nur bestimmte Aktionen erlaubt. Z.B. mit der Ausführung von $E_X = E_{\{a,b\}} \in \mathcal{P}(A)$ landet das System im Prozess $\mathcal{U}_{\{X\}}(p) = \mathcal{U}_{\{a,b\}}(p)$, wo das System nur die Aktionen $a$ und $b$ ausführen kann.

Nun kann jeder Prozess $\mathcal{U}(p)$, mit $p \in Proc$ eine Aktion $E_X \in \mathcal{P}(A)$ ausführen, die in einen Prozess $\mathcal{U}_X(p) \in Proc_\upsilon$ überführt (Siehe Abb. 1). Solange das System sich in einem solchen Prozess $\mathcal{U}_X(p)$ befindet, kann das System nur eine Aktion $\alpha \in (X \cup \{\tau, t\}) \cap \mathcal{I}(p)$ ausführen. Das heißt, die ausführbaren Aktionen sind die Aktionen, die die Umgebung erlaubt sowie die $\tau$ und die Timeout-Aktion $t$.

Es ist in Abb. 1 zu beachten, dass z.B. der Prozess $\mathcal{U}_{\{b\}}(p_1)$ nur die Aktion $b$ ausführen kann, mit $b \in \mathcal{I}(p_1) \cap X$ und $X = \{b\}$.

Beachte, dass eine $\tau$ Aktion aus einem Prozess $\mathcal{U}_X(p)$ auszuführen ist, wenn $\tau \in \mathcal{I}(p)$, obwohl diese eigentlich in $X$ nicht enthalten ist. Das hat den Grund, dass die $\tau$ Aktion nicht beobachtbar ist.

Die Timeout-Aktion $t \in \mathcal{I}(p).p \in Proc$ kann aus solchen Umgebungen $X'$ ausgeführt werden, mit $X' = \{e \mid e \in \mathcal{P}(A) \wedge \forall \alpha \in \mathcal{I}(p).\ \alpha \notin e\}$.

Nun stellt sich die Frage: Was ist, wenn ein Prozess $\mathcal{U}(p)$ eine Aktion $E_X \in \mathcal{P}(A)$ ausgeführt hat, sodass gilt $\forall a \in \mathcal{I}(p).\ a \notin X$ und $\tau, t \notin \mathcal{I}(p)$. Dann kommt das Bedürfnis der speziellen Aktion $t_\varepsilon$, mit der das System von $\mathcal{U}_X(p)$ in $\mathcal{U}(p)$ zurückkehrt. Das heißt, wenn das System nach der Ausführung eine $E_X \in \mathcal{P}(A)$

Aktion keine weitere Aktion ausführen kann, kehrt es mit der speziellen Aktion $t_\varepsilon$ zum ursprünglichen Prozess zurück.

## 2.5 Das mCRL2 Projekt

Das mCRL2-Projekt [mCR19] ist eine Menge von Tools, die sich mit beschrifteten Transitionssystemen beschäftigt. Dort kann ein *LTS* ausgehend vom mCRl2-Spezifikationsformat [mCR19, S. 23–25] erstellt und das System dann auf bestimmte Eigenschaften überprüft, visualisiert oder mit anderem *LTS* auf einen Äquivalenzbegriff wie starke Bisimilarität [AILS07, S. 42–61] verglichen werden.

Um die Funktionsweise zu verstehen, siehe [mCR19, S. 24, Fig. 2].

Die folgenden Tools sind in meinem Werkzeug verwendet:

- mcrl22lps [mCR19, S. 24–34]: Transformiert ein Transitionssystem in mCRL2-Spezifikationsformat in das lineare Prozess-Spezifikationsformat (*LPS*) [mCR19, S. 24–34].
- lps2lts [mCR19, S. 24–34]: Transformiert ein Transitionssystem in linearem Prozess-Spezifikationsformat (*LPS*) in ein beschriftetes Transitionssystem (*LTS*).
- ltscompare [mCR19, S. 24–34]: Vergleicht zwei beschriftete Transitionssysteme auf bestimmte Bisimilaritätsmodell.
- ltsgraph [mCR19, S. 24–34]: Visualisiert ein Transitionssystem.

# 3 Ansatz

In diesem Abschnitt sind die Funktionsweise meines Werkzeugs und dessen Aufbau gezeigt.

Um einen allgemeinen Überblick zu geben, ist im Folgenden eine Visualisierung der Arbeitsweise meines Werkzeugs gezeigt.

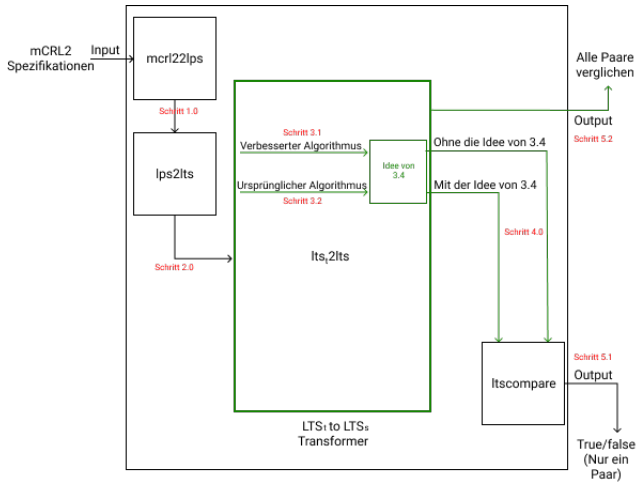



Abbildung 2: Architektur des Werkzeuges

Mein Ziel ist es eine reaktive Bisimilarität automatisiert zu überprüfen. Im Abschnitt Hintergrund wurde Folgendes eingeführt:

- Beschriftete Transitionssysteme ($LTS$) [AILS07, S. 20–25].
- Starke Bisimilarität [AILS07, S. 42–61].
- Reaktive Bisimilarität [vG20, S. 3–6].
- Reduktion von reaktiver zu starker Bisimilarität [Poh21].
- mCRL2-Projekt [mCR19].

All dieser Technologien und Konzepte sind in meinem Ansatz gebraucht.

## 3.1 Kernidee

Reaktive Systeme können in beschrifteten Transitionssystemen mit Timeouts $LTS_t$ [vG21a] modelliert werden [Poh21, S. 8]. Damit darauf reaktive Bisimilarität angewandt und überprüft werden kann [vG20, S. 3–6], sind zwei Sachen benötigt:

1. Ein Tool, um ein $LTS_t$ im Allgemeinen zu modellieren.
2. Ein Tool, um eine reaktive Bisimilarität in einem $LTS_t$ zu überprüfen.

Beginnen wir mit dem ersten Punkt: Es ist ein Tool gebraucht, welches ein beschriftetes Transitionssystem [AILS07, S. 20–25] modellieren kann, sodass die Ausgabe dieses Tools später für die Überprüfung einer reaktiven Bisimilarität verwendet werden kann.

MCRL2 kann ein beschriftetes Transitionssystem modellieren, indem das gewünschte System im mCRL2-Spezifikationsformat [mCR19, S. 23–25] eingegeben wird. Die Transformation vom mCRL2-Spezifikationsformat in einem $LTS_t$ passiert in mCRL2 in zwei Schritte.

1. Transformation vom mCRL2-Spezifikationsformat zu einem $LPS$ [mCR19, S. 24–34] durch das Tool mcrl22lps [mCR19, S. 24–34] (Siehe Schritt 1.0 in Abb. 2).
2. Transformation von diesem $LPS$ zu einem $LTS_t$ durch das Tool lps2lts [mCR19, S. 24–34] (Siehe Schritt 2.0 in Abb. 2).

lps2lts-Tool liefert eine Datei, die später mit einem anderem $LTS_t$ (bzw. Datei) auf verschiedene in mCRL2 implementierte Bisimilaritäten überprüft werden kann. Eine dieser Bisimilaritäten ist starke Bisimilarität. Das heißt, dass es immer zwei LTS's an lps2lts-Tool einzugeben sind, damit diese später verglichen werden können.

Das Problem ist jetzt aber, dass reaktive Bisimilarität überprüft werden soll und dafür gibt es kein Tool. Hier kommt also der zweite Punkt.

Wie in der Reduktion zu sehen ist, hat POHLMANN eine Reduktion [Poh21] definiert, die das reaktive Bisimilaritätsproblem zu starkem Bisimilaritätsproblem reduziert. Das heißt, Es ist nur noch die vom lps2lts-Tool ausgegebene $LTS_t$-Datei so zu modifizieren, dass die Reduktion auf dieses $LTS_t$ umgesetzt wird. Nach dem Umsetzen der Reduktion wird das in der Datei enthaltenes

$LTS_t$ zu einem $LTS_s$ transformiert. So kann das resultierende modifizierte $LTS_s$ in mCRL2 auf eine starke Bisimilarität überprüft werden. Die reaktiv bisimilaren Prozessen in $LTS_t$ sind dann in $LTS_s$ wegen Theorem 1 auch stark bisimilar.

Da an lps2lts-Tool zwei LTS's bzw. Dateien eingegeben werden müssen, kann es sein, dass beide LTS's verschiedene Aktionen haben. Es ist zu wissen, dass allen Prozesse in beiden Systeme alle mögliche Umgebung als Aktionen zugewiesen bekommen (Siehe Reduktion von reaktiver zu starker Bisimilarität und Funktionalität der Reduktion). Somit kann System 1 aufgrund der verschieden Aktionen im Vergleich zum System 2 solche Umgebungen haben, die aber bei System 2 nicht existieren (oder umgekehrt). Dieses Problem ist in der Implementierung der Transformation umgegangen, indem alle Umgebungen beim Ablesen beider Systemen eingesammelt und diese dann als erster Schritt der Transformation allen Prozesse zugewiesen werden.

Diese Aufgabe, also das Tool zur Transformation von $LTS_t$ zu $LTS_s$ mit dem Einhaltung des mCRL2-Dateiformats ist Ein Hauptgegenstand meiner Arbeit. Mein Tool habe ich $lts_t2lts$ genannt (Siehe $lts_t2lts$ in Abb. 2).

Die Implementierung des Reduktionsalgorithmus erfolgt in zwei Versionen. Die eine ist die ursprüngliche Reduktion (Siehe Schritt 3.1 in Abb. 2) und die andere ist eine Verbesserung davon (Siehe Schritt 3.2 in Abb. 2). Beide Varianten sind in den nächsten zwei Abschnitten erklärt.

## 3.2 Ursprünglicher Algorithmus

Um den Algorithmus leichter zu verstehen, empfiehlt es sich, den Abschnitt Funktionalität der Reduktion sorgfältig zu lesen.

Für den Algorithmus 1 sind drei Variablen zu konstruieren:

- *processes*: Ursprüngliche Prozesse im beschrifteten Transitionssystem mit Timeouts ($LTS_t$) [vG21a].
- *allSharedEnvironments*: Ist $\mathcal{P}(A)$, mit $\mathcal{P}(A)$ die Potenzmenge der eingegebenen Aktionen ohne $\{\tau, t\}$.
- *processSharedEnvironment*: Enthält Paare von (Prozessname, *allSharedEnvironments*).

**Algorithm 1** $LTS_t$ zu $LTS_s$ Transformationsalgorithmus ($getLTS_s$)

**Require:**

```
1:  processes, allSharedEnvironments, processSharedEnvironment
2:  newProcesses ← emptyList
3:  for process ∈ processes do
4:      processSharedEnvironment.add(process.getName(), allSharedEnvironments)
5:      ApplyRule2(allSharedEnvironments, process)
6:      for transition ∈ process.getOriginalTransitions() do
7:          processEnvironments ← processSharedEnvironment.getValueOf(process.getName())
8:          for environment ∈ processEnvironments do
9:              if transition.getAction().isTauAction() then
10:                 ApplyRule1(process, transition)
11:                 ApplyRule4(process, transition, environment)
12:             else if transition.getAction().isNormalAction() then
13:                 ApplayRule3(process, transition, environment)
14:             else if transition.getAction().isTimeAction() then
15:                 ApplyRule5(process, environment)
16:                 ApplyRule6(process, transition, environment)
17:             end if
18:         end for
19:     end for
20:     applyRule5OfRemainingEnvironments(process, processSharedEnvironment)
21: end for
22: result ← getNewProcesses()
```

In Zeile 2 ist über alle Prozesse im eingegebenen $LTS_t$ iteriert. Das Hinzufügen des Paares in der Liste *processSharedEnvironment* in Zeile 3 ist wegen geändertem Status der Umgebungen notwendig. Somit hat jeder Prozess seine eigene Umgebungen mit deren Status. In Zeile 4 ist Regel 2 gegeben der Definition der Reduktion anzuwenden, sodass jeder ursprüngliche Prozess $p$ eine neue Menge von Transitionen T, mit $T = \{t \mid t = (\mathcal{U}(p), E_X, \mathcal{U}_{\{X\}}(p)) \wedge X \in allSharedEnvironments\}$ bekommt (Hinweis: In den Funktionen *ApplyRule∗* wird beim Bedarf ein neuer Prozess erstellt, der alle ursprüngliche Aktionen sowie die neu generierten Aktionen, sprich die Umgebungsaktionen und alle neu entstehende Transitionen beinhaltet. Der neu erstellte Prozess wird dann auch durch die Funktionen *ApplyRule∗* in der Liste *newProcesses* hinzugefügt). Dann ist in Zeile 5 über alle ursprüngliche Transitionen des aktuellen Prozesses und danach in Zeile 7 über alle Umgebungsaktionen dieses Prozesses iteriert, die vorhin dem Prozess *process* zugewiesen ist. Das ist zu machen, da von jeder Umgebung eine Aktion ausgehen muss. Diese Aktion kann eine Aktion aus $A$ (Zeile 11), $\{\tau\}$ (Zeile 8) oder $\{t\}$ (Zeile 13) sein. Kann aus der jeweiligen Umgebung keine Aktion ausgehen, dann ist die Aktion $t_\varepsilon$ (Zeile 19) zu verwenden.

Nun sind die if und else-if Anweisungen in Zeilen 8, 11 und 13 in Betracht zu nehmen. Dort wird getestet, welche ursprüngliche Aktionen der jeweilige ursprüngliche Prozess hat. Je nachdem welche Aktion der Prozess hat, sind die benötigten Regeln aus der Definition der Reduktion anzuwenden. Folgende drei Möglichkeiten gibt es:

- Wenn die Aktion eine $\tau$ Aktion ist (Zeile 8), dann müssen laut der Definition der Reduktion die Regeln 1 und 4 angewandt werden.

- Wenn die Aktion eine $\alpha \in A$ ist (Zeile 11) ($A$ ist die Menge aller sichtbaren Aktionen $\tau \notin A$), dann muss laut der Definition der Reduktion den Regel 3 angewandt werden.
- Wenn die Aktion eine $t$ Aktion ist (Zeile 13), dann müssen laut der Definition der Reduktion die Regeln 5 und 6 angewandt werden.

Nach der Iteration über alle ursprüngliche Prozesse kann die Liste $newProcesses$ durch die Funktion $getNewProcesses()$ aufgerufen werden, die alle neu entstehende Prozesse und deren neu entstehenden Transitionen enthält, sodass das neue Transitionssystem $LTS_s$ gebildet werden kann.

Wird das $LTS_t$ in ein $LTS_s$ [AILS07, S. 20–25] transformiert, so kann nun das resultierende $LTS_s$ mit Hilfe von mCRL2 [mCR19] auf starke Bisimilarität [AILS07, S. 42–61] überprüft werden und somit laut Reduktion auch auf reaktive Bisimilarität [vG20, S. 3–6]. Dabei wird das Ziel dieser Arbeit erreicht.

## 3.3 Verbesserter Algorithmus

Unter den in dieser Arbeit getesteten Stichproben war [vG21b, 46], dessen Transformation mit der ursprünglichen automatisierten Version der Reduktion sehr viel Zeit dauert. Dieses System kann schneller transformiert werden, wenn die Reduktion verbessert werden könnte. Eine Verbesserungsmöglichkeit ist in dieser Arbeit gefunden aber nicht bewiesen.

Die Verbesserung des Algorithmus stellt sich in in der Variable $allSharedEnvironments$ im Algorithmus 1 dar. Diese Variable enthält im ursprünglichen Algorithmus alle $e \in \mathcal{P}(A)$, wobei $\mathcal{P}(A)$ die Potenzmenge der angegebenen sichtbaren Aktionen außer $\{\tau, t\}$. In der verbesserten Version hingegen ist $allSharedEnvironments = \{\{a\} \mid a \in A\} \cup \{T \mid T = A \setminus \mathcal{I}(p_t).\ p_t \in Proc\}$, wobei $p_t$ ein Prozess ist, der mindestens eine Timeout-Aktion hat und $\mathcal{I}(p_t)$ die initialen Aktionen vom Prozess $p_t$. Sei $P_t$ die Menge aller solchen Prozesse $p_t$. Somit ist $|allSharedEnvironments| = |A| + |P_t|$.

Damit ist die Anzahl aller generierten Prozesse im Algorithmus von $|Proc_\upsilon| = |Proc| \cdot (1 + 2^{|A|})$ auf höchstens $|Proc_\upsilon| = |Proc| \cdot (|A| + |P_t| + 1)$ reduziert. Es ist auf $|Proc_\upsilon| = |Proc| \cdot (|A| + |P_t| + 1)$ zu kommen, denn jeder Prozess $p \in Proc_\upsilon$ hat die Liste der Umgebungsaktionen $allSharedEnvironments$. Somit geht das System über alle $X \in allSharedEnvironments$ in einen Prozess $\mathcal{U}_X(p)$. Also $|\mathcal{U}_X(p)| = |allSharedEnvironments| \cdot |Proc|$. Es gibt aber noch die Prozesse $\mathcal{U}(p)$ und dafür ist die $+1$ in $|Proc_\upsilon| = |Proc| \cdot (|A| + |P_t| + 1)$, denn $|\mathcal{U}(p)| = |Proc|$

Somit ändert sich nur einen Regel in der Definition der Reduktion von reaktiver auf starke Bisimilarität und zwar den Regel 2. Der Regel 2 wird zu:

$$\frac{}{\mathcal{U}(p) \xrightarrow{E_X}_\upsilon \mathcal{U}_X(p)}\ X \in U, \text{ mit}$$

$U = \{\{a\} \mid a \in A\} \cup \{A \setminus \mathcal{I}(p) \mid t \in \mathcal{I}(p).\ p \in Proc \wedge t \text{ ist die Timeout-Aktion}\}$

Mit der verbesserten Version reduziert sich dementsprechend die Laufzeit des Algorithmus erheblich. Damit ist der Umfang der möglichen berechenbaren Algorithmen, schnell zu vergleichen viel größer als beim ursprünglichen Algorithmus.

Die Korrektheit der verbesserten Version ist nicht bewiesen, aber bei der Überprüfung aller Stichproben, ergibt deren Vergleich bei beiden Versionen das gleiche Ergebnis.

Das obige Beispiel (Abb. 1) wird mit der Verwendung der verbesserten Version wie folgt aussehen:

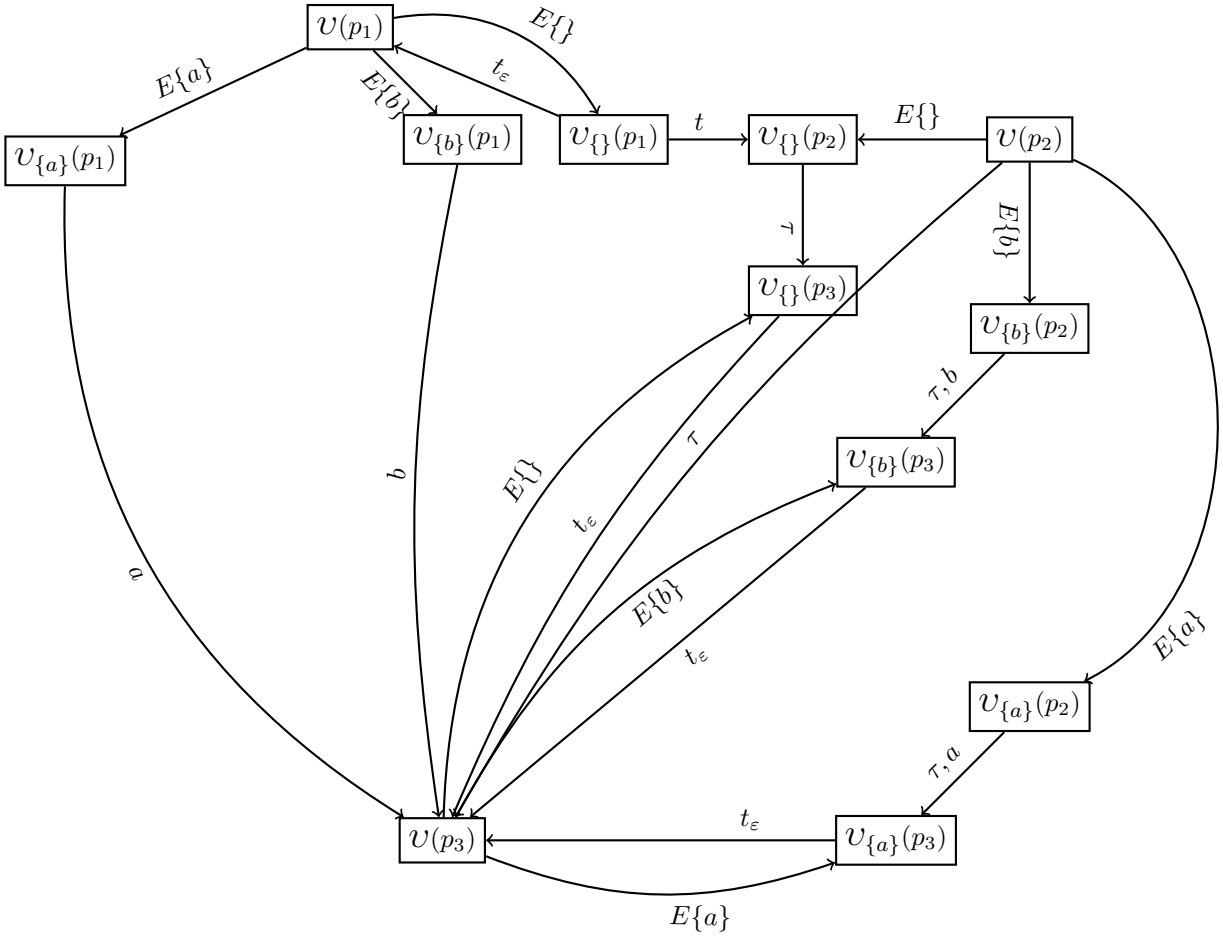



Abbildung 3: Verbessertes transformiertes System

### 3.4 Eine einzige Umgebungsaktion $E_{\{...\}}$ anstatt von verschiedenen

Wie im Abschnitt Funktionalität der Reduktion erwähnt, bekommt jeder ursprüngliche Prozess die Aktionsmenge $\mathcal{P}(A)$ nach der Reduktion zugewiesen. Diese Menge ist im ursprünglichen und verbesserten Algorithmus verschieden groß. Die eine Menge wächst exponentiell und die andere linear. Was ist aber, wenn diese Menge nur ein Element hat?

Wird nun nach der Reduktion die Menge $\mathcal{P}(A)$ aus der Aktionsmenge bei jedem ursprünglichen Prozess Entfernt und stattdessen nur eine Aktion $E_{\{...\}}$ als Vertretung hinzufügt. Dann führt man zum Vergleichen eine Überprüfung reaktiver Bisimilarität durch, so ist das gleiche Ergebnis zu erhalten. Das ist der Fall bei allen meinen Stichproben. Diese Idee erwähnt POHLMANN in seiner

Arbeit [Poh21, S. 46].

Diese Idee wirkt sich nicht auf die Effizienz der Reduktion beider Versionen aus, sondern auf den Vergleich mit starker Bisimilarität [AILS07, S. 42–61] und somit auf das gesamte Werkzeug.

Diese Idee, also nur eine Aktion $E_{\{...\}}$ anstatt von $\mathcal{P}(A)$ zu verwenden, kann in meinem Werkzeug nur beim Vergleich eines einzigen Paars umgesetzt werden (Siehe Schritt 4.0 in Abb. 2).

## 3.5 Überprüfung aller Paare oder von nur einem Paar

Mein Werkzeug kann entweder durch mCRL2 [mCR19] ein Prozess-paar (Siehe Schritt 5.1 in Abb. 2) oder durch mein Tool mit der Verwendung die von mir implementierte Definition starker Bisimilarität [AILS07, S. 42–61] alle Prozess-paare der eingegebenen Systemen überprüfen (Siehe Schritt 5.2 in Abb. 2).

Da die Definition starker Bisimilarität relativ bekannt ist, genügt es hier nur oberflächlich den Pseudocode vorzustellen:

**Algorithm 2** Algorithmus von starker Bisimilarität

**Require:**

1: $processList1$
2: $processList2$
3: $pairs \leftarrow processList1 \times processList2$
4: **while** $true$ **do**
5: $\quad pairs.removeIf(p \rightarrow$
6: $\quad\quad !checkSimulation(p2q, pairs, p.getFirst(), p.getLast()) ||$
7: $\quad\quad !checkSimulation(q2p, pairs, p.getLast(), p.getFirst())$
8: $\quad )$
9: $\quad$ If $pairs$ does not change $beark$
10: **end while**
11: **return** $pairs$

**Algorithm 3** checkSimulation

**Require:**

1: $leftToRight$ (check (p,q) or (q,p))
2: $pairs$
3: Prozess $p$ from pair
4: Prozess $q$ from pair
5: **return** $p.getTransitions().allMatch(t1 \rightarrow$
6: $\quad\quad q.getTransitions().anyMatch(t2 \rightarrow$
7: $\quad\quad\quad t1.getAction().getName().equals(t2.getAction().getName())$
8: $\quad\quad \&\&$
9: $\quad areSuccessorsInR(leftToRight, pairs, t1.getSuccessor(), t2.getSuccessor())))$

# 4 Evaluation

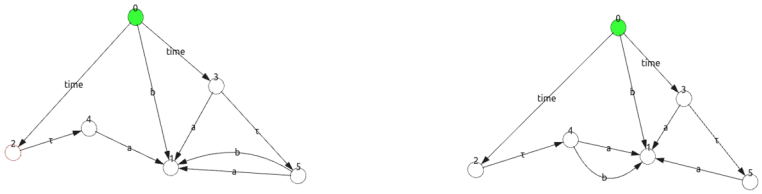
In diesem Abschnitt sind 7 Beispiele von reaktiver Bisimilarität und die Überprüfungsergebnisse sowie Statistiken über den ursprünglichen und verbesserten Algorithmus gezeigt.

Hinweis: (x, y) in allen Beispiele, gehört x zum System1 und y zum System2.

Hinweis: $a \to b \to c$ ist eine Sequenz von Aktionen beginnend von Prozess 0.

Hinweis: In allen Beispiele steht System1 auf der linken Seite und System2 auf der rechten.

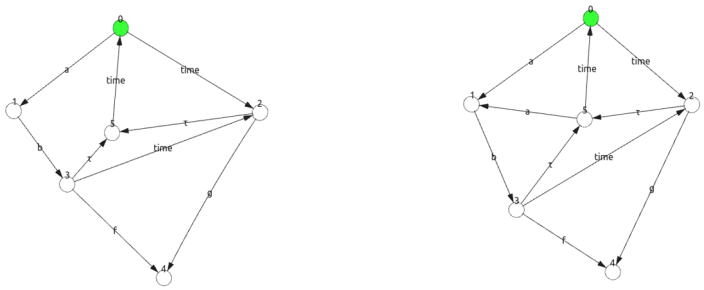
## 4.1 Beispiel 1




| | Ursprüngliche Version | Verbesserte Version |
|---|---|---|
| Überprüfe (0, 0) | true | true |
| reaktiv bisimilare Paare | (0, 0), (1, 1), (4, 5), (5, 4) | (0, 0), (1, 1), (4, 5), (5, 4) |
| Prozess-Anzahl | 60 | 36 |
| Reduktionsausführungszeit | 120,3 ms | 117 ms |

Das Paar (0,0) ist reaktiv bisimilar, da die Timeout-Aktionen von Prozess 0 in beiden Systemen die Ausführung von der Aktion $b$ verbieten. Das Paar (1, 1) ist ein Blatt und dementsprechend reaktiv bisimilar. (4, 5) haben nur die Aktion $a$ und (5, 4) haben nur die Aktionen $a, b$.

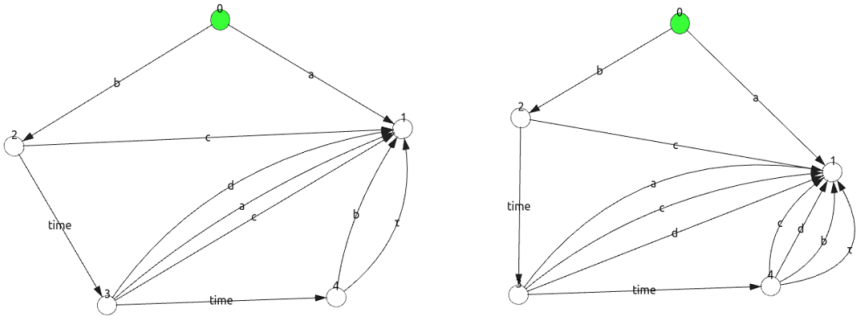
## 4.2 Beispiel 2



| | Ursprüngliche Version | Verbesserte Version |
|---|---|---|
| Überprüfe (0, 0) | false | false |
| reaktiv bisimilare Paare | (4, 4) | (4, 4) |
| Prozess-Anzahl | 204 | 84 |
| Reduktionsausführungszeit | 130,6 ms | 122,9 ms |

Das Paar (0,0) ist nicht reaktiv bisimilar, da es im System2 einen anderen Weg als $time \rightarrow \tau \rightarrow a$ gibt, wo die Aktion $a$ erreicht wird, und zwar: $a \rightarrow b \rightarrow \tau \rightarrow a$. Das Paar (4, 4) ist ein Blatt und dementsprechend reaktiv bisimilar.

## 4.3 Beispiel 3

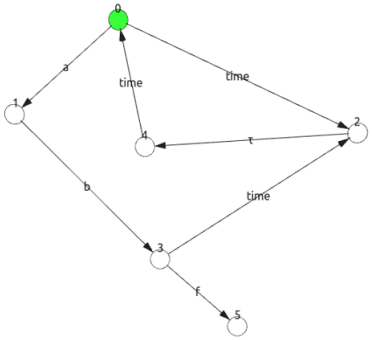


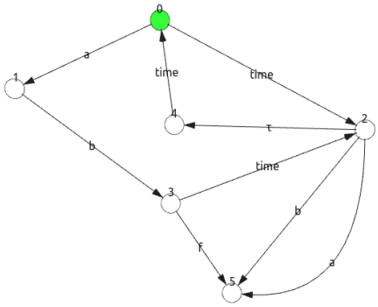

| | Ursprüngliche Version | Verbesserte Version |
|---|---|---|
| Überprüfe (0, 0) | true | true |
| reaktiv bisimilare Paare | (0, 0), (1, 1), (2, 2), (3, 3) | (0, 0), (1, 1), (2, 2), (3, 3) |
| Prozess-Anzahl | 170 | 60 |
| Reduktionsausführungszeit | 123 ms | 117,9 ms |

Das Paar (0,0) ist reaktiv bisimilar, da die Timeout-Aktion von Prozess 3 im System2 die Ausführung der Aktionen $c$, $d$ verbietet. Das Paar (1, 1) ist ein Blatt und dementsprechend reaktiv bisimilar. (2, 2) und (3, 3) haben die gleichen Aktionen.

## 4.4 Beispiel 4

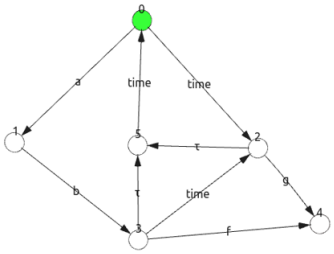
Das Paar (0,0) ist nicht reaktiv bisimilar, da die Timeout-Aktion von Prozess 0 im System2 die Ausführung der Aktion $b$ im Prozess 2 nicht verbietet. Das Paar (5, 5) ist ein Blatt und dementsprechend reaktiv bisimilar.

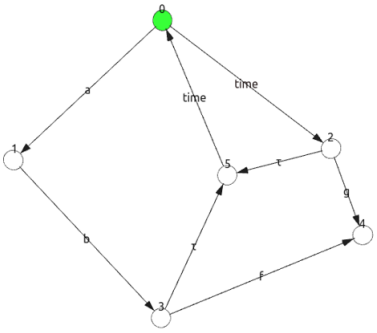



| | Ursprüngliche Version | Verbesserte Version |
|---|---|---|
| Überprüfe (0, 0) | false | false |
| reaktiv bisimilare Paare | (5, 5) | (5, 5) |
| Prozess-Anzahl | 108 | 84 |
| Reduktionsausführungszeit | 114,7 ms | 116,5 ms |

## 4.5 Beispiel 5

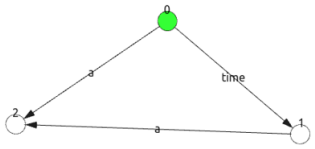


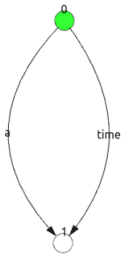




Hier sind die beiden Systeme gleich, da die Timeout-Aktion im System1 in Prozess 3 wegen der $\tau$ Aktion nicht betrachtet wird.

| | Ursprüngliche Version | Verbesserte Version |
|---|---|---|
| Überprüfe (0, 0) | true | true |
| reaktiv bisimilare Paare | (0,0),(1,1),(2,2),(3,3),(4,4),(5,5) | (0,0),(1,1),(2,2),(3,3),(4,4),(5,5) |
| Prozess-Anzahl | 204 | 84 |
| Reduktionsausführungszeit | 133,7 ms | 119,2 ms |

## 4.6 Beispiel 6




Dieses Beispiel zeigt einen interessanten Fall in einer reaktiven Bisimilarität.

Die Ausführung der Timeout-Aktion $t$ beschränkt bestimmte Aktionen durch eine Umgebung. Wenn das obige Beispiel angeguckt wird, ist es dann festzustellen, dass beide Systeme eine Timeout-Aktion $t$ haben, deren Ausführung jeweils die Aktion $a$ verbietet. Das heißt, würde die Timeout-Aktion $t$ in beiden Systemen genommen, dann ist es in Zustand 1 in beiden Systemen zu landen. Nun hat System2 sowieso keine weitere Aktionen. Bei System1 wäre die Aktion $a$ möglich, wenn die Timeout-Aktion $t$ nicht genommen würde. Es sieht nun so aus, als ob die beiden Systemen reaktiv bisimilar sind. Das ist aber erstaunlicherweise nicht der Fall.

Der Grund ist die folgende Bedingung in der Definition reaktiver Bisimilarität:

- wenn $\mathcal{I}(p) \cap (X \cup \{\tau\}) = \emptyset$, dann $(p, q) \in \mathfrak{R}$.

Somit müssten die Prozesse 0 in beiden Systeme nicht reaktiv bisimilar sein. Mein Werkzeug gibt hier natürlich $false$ aus.

## 4.7 Beispiel 7

Das Beispiel ist ist aus [vG21b, 46].

| | Ursprüngliche Version | Verbesserte Version |
|---|---|---|
| Prozess-Anzahl | 21074 | 1148 |
| Reduktionsausführungszeit | 151,958 s | 466,5 ms |

Das Ziel dieses Beispiels, ist zu zeigen, wie effizient die verbesserte Version des Algorithmus im Fall von großen Systemen ist.

Es ist hier zu sehen, dass die verbesserte Version in diesem Beispiel 325,7 mal effizienter als die ursprüngliche. Das ist natürlich so erwartet, denn die Zeitkomplexität der Generierung der Prozesse von dem ursprünglichen Algorithmus ist exponentiell. Die verbesserte Version ist hingegen linear.

# 5 Verwendungsweise

Mein Werkzeug ist einfach zu verwenden. Alles was gebraucht ist, ist in dem Ordner all_in_one in meinem GitHub Repo. Dort gibt es das Bash-skript bin.sh, mit welchem mein Werkzeug ausführt werden kann.

Anforderungen:

- Linux (Betriebssystem/Subsystem)
- JAVA Version $\geq$ v11.

Das Skript bekommt 3 Parameter:

- lts-Datei in mCRL2-Spezifikationsformat [mCR19, S. 23–25], die das System1 spiegelt (obligatorisch).
- lts-Datei in mCRL2-Spezifikationsformat [mCR19, S. 23–25], die das System2 spiegelt (obligatorisch).
- Features Parameter (optional).

Parameter 1 und 2 (System1 und System2) sind die Systeme, die auf eine reaktive Bisimilarität [vG20, S. 3–6] überprüft werden sollen.

Parameter 3 kann eine beliebige Kombination aus der Buchstaben **i, p, a und s** sein, die folgende Funktionen haben:

- i: Verwende den verbesserten Algorithmus (i $\rightarrow$ improved).
- p: Vergleiche alle nicht reflexiven, symmetrischen Paare der beiden Systeme (p $\rightarrow$ pairs).
- a: Wende nur eine Umgebungsaktion an (die Idee in 3.4).
- s: Visualisiere die beiden Systeme durch mCRL2 (s $\rightarrow$ show).

Beispiel ist der folgende Aufruf:
`./bin.sh system1.mcrl2 system2.mcrl2 ips`. Damit wird der verbesserte Algorithmus verwendet, alle nicht reflexiven symmetrischen Paare verglichen und die Systeme angezeigt.

`./bin.sh system1.mcrl2 system2.mcrl2 ias`. Damit wird der verbesserte Algorithmus verwendet, eine Umgebungsaktion angewandt und die Systeme angezeigt.

Würde das Werkzeug mit der folgenden Kombination ausführt werden: `./bin.sh system1.mcrl2 system2.mcrl2 pa`, dann wird a nicht betrachtet. Denn die Option a funktioniert mit der Überprüfung aller Paare nicht.

# 6 Fazit

In dieser Arbeit ist die Reduktion von mir automatisiert und in meinem entwickelten Werkzeug zur Überprüfung reaktiver Bisimilarität integriert. Welches auch ein Paar Tools vom mCRL2-Projekt [mCR19] als Plugin enthält.

Das Werkzeug kann nur ein Paar oder alle Paare auf reaktive Bisimilarität [vG20, S. 3–6] überprüfen. Das Werkzeug verwendet einen Algorithmus, dessen Aufwand exponentiell wächst. Eine verbesserte Version davon ist auch

entwickelt, die nur einen linearen Aufwand hat. Mein Werkzeug mit allen im Abschnitt Verwendungsweise erwähnten Features sind durch nur ein Skript verwendbar. Mein Werkzeug kann in wissenschaftlichen Arbeiten benutzt werden, um Studierenden beim Erlernen reaktiver Bisimilarität zu helfen und große Systeme durch die verbesserte Version des Algorithmus effizienter testen.

Die Korrektheit der verbesserten Version des Algorithmus ist nicht bewiesen. Somit könnte der Beweis von dessen Korrektheit Bestandteil einer zukünftigen Arbeit sein.

Eine andere mögliche zukünftige Arbeit könnte beweisen, was im Unterabschnitt Eine einzige Umgebungsaktion $E_{\{\dots\}}$ anstatt von verschiedenen erklärt ist.

# Literatur


[AILS07] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, 2007. doi:10.1017/CBO9780511814105.

[Lam19a] Leslie Lamport. *The Mutual Exclusion Problem: Part I—A Theory of Interprocess Communication*, page 227–245. Association for Computing Machinery, 2019. doi:10.1145/3335772.3335937.

[Lam19b] Leslie Lamport. *The Mutual Exclusion Problem: Part II—Statement and Solutions*, page 247–276. Association for Computing Machinery, 2019. doi:10.1145/3335772.3335938.

[mCR19] *The mCRL2 Toolset for Analysing Concurrent Systems Improvements in Expressivityand Usability*. Springer International Publishing, 2019. doi:10.1007/978-3-030-17465-1.

[Poh21] Maximilian Pohlmann. *Reducing Strong Reactive Bisimilarity to Strong Bisimilarity*. 2021. URL: https://maxpohlmann.github.io/Reducing-Reactive-to-Strong-Bisimilarity/thesis.pdf.

[vG20] Rob van Glabbeek. Reactive bisimulation semantics for a process algebra with time-outs. 2020. URL: https://arxiv.org/abs/2008.11499.

[vG21a] Rob van Glabbeek. Failure trace semantics for a process algebra with time-outs. 2021. doi:10.23638/LMCS-17(2:11)2021.

[vG21b] Rob van Glabbeek. *Modelling Mutual Exclusion in a Process Algebra with Time-outs*. CoRR, 2021. URL: https://arxiv.org/abs/2106.12785.

## Zusammenfassung in englischer Sprache / English-language summary

This work presents a tool that checks reactive bisimilarity [vG20] on labeled transition systems with timeouts ($LTS_t$) [vG21a].

The tool is developed from a reduction from reactive to strong bisimilarity [Poh21] I automated and a toolset from the mCRL2 project.

A pair or all pairs of processes of the entered $LTS_t$ can be checked for reactive bisimilarity. Furthermore, in this work an improvement possibility regarding the complexity of the algorithm of reduction is implemented. The complexity is reduced from exponential to linear effort. The improvement of the algorithm enables faster and more efficient verification of large systems, where the original version can only manage with very long execution time.