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

Bachelor Arbeit

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
Oktober 2022

1 Abstrakt

Diese Arbeit stellt ein Werkzeug vor, das die reaktive Bisimilarität [vG20] auf beschriftete Transitionssysteme mit Timeouts (LTS_t) [vG21a] überprüft.
Das Werkzeug ist anhand einer Reduktion von reaktiver zu starker Bisimilarität [Poh21] entwickelt. Das Werkzeug kann ein Paar oder alle Paare von Prozessen des eingegebenen LTS_t auf reaktive Bisimilarität überprüfen. Außerdem implementiere ich in dieser Arbeit eine Verbesserungsmöglichkeit bezüglich der Komplexität des Algorithmus der Reduktion. Die Komplexität ist von $m \cdot (1 + 2^{|n|})$ auf $m \cdot (n + t + 1)$ vermindert. Die Effizienz kann deutlich bemerkbar sein, wenn die beiden Versionen auf große Systeme umgesetzt werden. Diese Verbesserung ermöglicht eine schnellere und effiziente Überprüfung von großen Systemen, mit denen die ursprüngliche Version nur mit sehr langer Ausführungszeit auskommt.

Inhaltsverzeichnis

2 Einführung

Wie im Abschnitt Hintergrund zu sehen ist, können verschiedene Gleichheitsbegriffe wie starke Bisimilarität [AILS07, S. 42–61] und reaktive Bisimilarität [vG20, S. 3–6] auf beschrifteten Transitionssysteme [AILS07, S. 20–25] angewandt werden. Bisimilaritätsmodelle, die keine timeouts definieren wie starke und schwache Bisimilarität [AILS07, 3.4 Weak bisimilarity] werden mit beschrifteten Transitionssysteme ohne timeouts (LTS_s) [AILS07, S. 20–25] benutzt und die aber timeouts verwenden wie die reaktive Bisimilarität werden mit beschrifteten Transitionssysteme mit timeouts (LTS_t) [vG21a] verwendet.

Solange ein beschriftetes Transitionssystem ein System mit dessen Aktionen zwischen Prozesse modellieren kann [Poh21, S. 8] und wir darauf verschiedene Gleichheitsbegriffe anwenden können, ist die Existenz von Algorithmen für die verschiedenen Gleichheitsbegriffe auf solche Systeme von Bedeutung. Damit bei Test- oder experimentellen Zwecke auf die theoretische Vergleichung verzichtet und den damit verbundenen Zeitverlust gespart werden kann.

Es existieren mehrere Algorithmen und Tools, die verschiedene Gleichheitsbegriffe behandelt haben. Ein Beispiel dafür ist das mCRL2-Projekt [mCR19], wo verschiedene Algorithmen von z.B. der starken und schwachen Bisimilarität implementiert bzw. automatisiert wurden. Diese Algorithmen können auf Transitionssysteme ausgeführt werden, die im mCRL2 in einem Spezifikationsformat [mCR19, S. 23–25] geschrieben sind.

Der neue Gleichheitsbegriff reaktive Bisimilarität ist ein wichtiges Bisimilaritätsmodell. Denn dort werden Systeme behandelt, deren Implementierung Auszeiten (timeouts) enthalten [vG20, S. 3–6]. Diese können aber von anderen Bisimilaritätsmodelle wie starke Bisimilarität nicht 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].

Gegenseitigen 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 einen geteilten Variable überschrieben werden soll. Solche Probleme werden gelöst, indem man nur einen Prozess in diesem kritischen Code-Abschnitt reinkommen lässt. Alle andere Prozesse, die auch reinkommen möchten, müssen aber **warten**. Genau hier kann die reaktive Bisimilarität ihre Rolle für den Vergleich spielen. Dank [vG20, S. 3–6] können wir solche Systeme vergleichen. Wir wollen aber auch einen Algorithmus haben, die diese Aufgabe erledigt, um den Vergleich automatisiert auszuführen. An dieser Stelle erwähne ich die Arbeit von **Maximilian Pohlmann** [Poh21]. Wo eine Reduktion definiert wird, die Systeme von reaktiver zu starker Bisimilarität reduziert. Das heißt, wir können zwei Prozesse auf starke Bisimilarität anstatt von reaktiver Bisimilarität überprüfen, sodass das Theorem 1 gilt.

Diese Reduktion ist algorithmisch und kann dann automatisiert werden.

In dieser Arbeit habe ich ein Werkzeug in Java entwickelt, das die reaktive Bisimilarität [vG20, S. 3–6] überprüft.

Der implementierte Algorithmus basiert auf die mathematisch bewiesene Reduktion von **Maximilian Pohlmann** [Poh21, S. 33–41], sodass mein Werkzeug mit mCRL2 kompatibel ist.

Ich habe zuerst in meinem Werkzeug die Reduktion roh implementiert und dann verbessert. Der Benutzer kann auswählen, ob den ursprünglichen Algo-

rithmus oder den verbesserten zur Überprüfung ausgeführt werden soll. Womit auch alle Paare oder nur ein Paar von Prozessen auf die reaktive Bisimilarität überprüft werden können.

Um eine angenehme Erklärungsmethode zu ermöglichen, erkläre ich als erstes die beschrifteten Transitionssystemen, die starke und die reaktive Bisimilarität. Dann kann man in die Reduktion von reaktiver zu starker Bisimilarität einsteigen. Danach stelle ich einiges vom mCRL2 vor und am Ende erkläre ich mein Werkzeug.

Diese Arbeit ist wie folgt strukturiert:
Einen abstrakten Überblick habe ich im Abschnitt Abstrakt geschafft. Bei vertrauten Kenntnisse mit beschrifteten Transitionssysteme, starker und reaktiver Bisimilarität, mCRL2 und der Reduktion von reaktiver zu starker Bisimilarität kann den Abschnitt Hintergrund übersprungen werden. Sonst ist es empfehlenswert diesen Abschnitt zu lesen. Der Abschnitt Ansatz zeigt wie mein Werkzeug aufgebaut ist und die Arbeitsweise des ursprünglichen Algorithmus sowie die verbesserte Version. Man kann dann erkennen, wie effektiv die verbesserte Version des Algorithmus ist. Indem man im Abschnitt Evaluation den Unterschied zwischen dem ursprünglichen und verbesserten Algorithmus sieht. Der Abschnitt Verwendungsweise erklärt wie mein Werkzeug anzuwenden ist. Am Ende erläutere ich im Abschnitt Fazit das allgemeine Ergebnis sowie zukünftige, mögliche, mit meinem Thema Verwandte Arbeiten.

3 Hintergrund

3.1 Beschriftete Transitionssysteme

Ein beschriftetes Transitionssystem [AILS07, S. 20–25] besteht aus Zustände und Aktionen. Diese Aktionen passieren, wenn eine Transition zwischen zwei Zustände genommen wird. Dementsprechend sind alle Transitionen mit Aktionen aus einer gegebenen Menge A beschriftet. Die Zustände können auch mit einem Prädikat aus einer gegebenen Menge P beschriftet sein.
Die folgende Definition ist aus [AILS07, Definition 2.1].

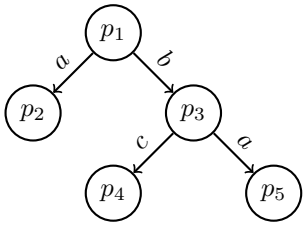
3.1.1 Definition

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

- P : Menge der Zustände.
- A : Menge der Aktionen.
- $\xrightarrow{\alpha} \in P \times P$, für jede $\alpha \in A$.

Somit passiert eine Aktion $a \in A$ zwischen Prozess $p, p' \in P$ mit (p, a, p') oder wir können die Intuitive Notation $p \xrightarrow{a} p'$ zur Nutze machen.
Ein LTS kann reaktive Systeme modellieren [Poh21, S. 8], wo die starke und reaktive Bisimilarität angewandt wird [AILS07, S. 42–61] und [vG20, S. 3–6]. Gleich werde ich LTS_s für starke Bisimilarität und LTS_t für reaktive Bisimilarität vorstellen.

3.1.2 Beispiel



Wir sehen in diesem LTS, dass der Prozess p_1 zwei Aktionen nach p_2 und p_3 vornehmen kann.

3.2 Starke Bisimilarität

Ein normales beschriftetes Transitionssystem wird als LTS_s [vG21a, S. 1] bezeichnet. Das ist ein beschriftetes Transitionssystem, das nur die Aktionen in der gegebenen Menge A und die Aktion τ enthalten kann.
Die starke Bisimilarität [AILS07, S. 42–61] ist auf Zustände eines gegebenen LTS_s definiert. Die starke Bisimilarität ist der feinste Äquivalenzbegriff auf Transitionssysteme [Poh21, Strong Bisimilarity]. Dieser Begriff betrachtet alle Aktionen in der gegebenen Menge A und auch die versteckte Aktion τ.

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

3.2.1 Definition

Sei:

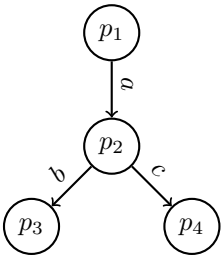
- LTS_s: Ein normales beschriftetes Transitionssystem.
- A: Die eingegebene Menge der Aktionen in LTS_s.
- $\mathbb{N}$: Die Menge der natürlichen Zahlen.
- z_i, z_i': Zustände in einem LTS_s, mit $i \in \mathbb{N}$.
- α: Eine Aktion in A.
- $z_1 \xrightarrow{\alpha} z_2$: Eine Transition von z_1 zu z_2, mit $\alpha \in A$.

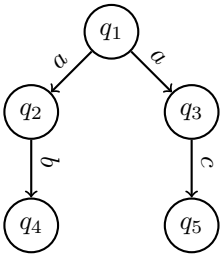
Die starke Bisimulation ist eine binäre Relation $\mathfrak{R}$ auf eine Menge von Zustände von einem LTS_s, wenn alle $z_1 \ \mathfrak{R} \ z_2$ und eine Aktion α gibt, mit

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

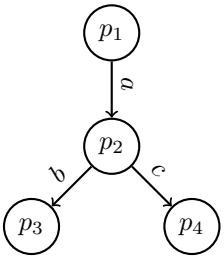
Somit sind zwei Zustände z_1 und z_2 stark bisimilar geschrieben $z_1 \sim z_2$, wenn sie in einer starken Bisimulation enthalten sind.

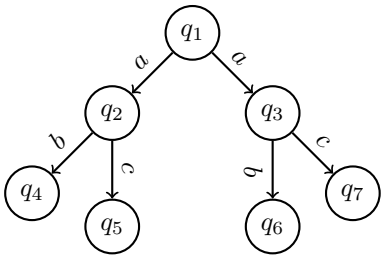
3.2.2 Beispiele





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 würde und q_2 kann die Aktion c nicht nachmachen. Da er auf 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 bei allen von p_1 und q_1 Nachfolger Prozesse möglich zwischen den Aktionen b und c zu wählen.

3.3 Reaktive Bisimilarität

Ein beschriftetes Transitionssystem mit Auszeiten (timeouts) wird als LTS_t [Poh21, S. 20] bezeichnet. Dieses Transitionssystem kann nicht nur die Aktionen in der gegebenen Menge A und die Aktion τ enthalten, sondern auch eine Zeit-Aktion t. Die reaktive Bisimilarität [vG20, S. 3–6] ist auf Zustände eines gegebenen LTS_t definiert. Die reaktive Bisimilarität ist eine schwächere Variante von der starken Bisimilarität. Da die 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].

3.3.1 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 $X \subseteq A$.
- $a, a', b, b', ..., z, z'$: Zustände in P.
- $p \xrightarrow{\alpha} p'$: Eine Transition von p zu p', mit $\alpha \in Act$.
- $\mathcal{I}(p)$: Die initiale Aktionen von Prozess $p \in P$.
- $\mathcal{P}(A)$: Die Potenzmenge von A.

Die starke reaktive Bisimulation ist eine symmetrische Relation:

$$\mathfrak{R} \subseteq (P \times \mathcal{P}(A) \times P) \cup (P \times P)$$

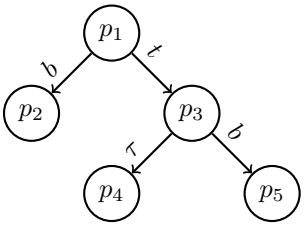
Bedeutet dass, $(p, X, q) \in \mathfrak{R} \Leftrightarrow (q, X, p) \in \mathfrak{R}$ und $(p, q) \in \mathfrak{R} \Leftrightarrow (q, p) \in \mathfrak{R}$, mit:

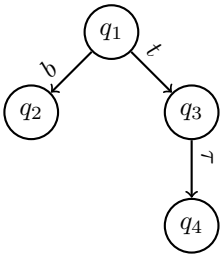
- wenn $(p, q) \in \mathfrak{R}$ und $p \xrightarrow{\tau} p'$, dann existiert ein q', mit $q \xrightarrow{\tau} q'$ und $(p', q') \in \mathfrak{R}$.
- wenn $(p, q) \in \mathfrak{R}$, dann $(p, X, q) \in \mathfrak{R}$ für alle $X \subseteq A$.

und für alle $(p, X, q) \in \mathfrak{R}$

- wenn $p \xrightarrow{a} p'$, mit $a \in X$ dann existiert ein q', mit $q \xrightarrow{a} q'$ und $(p', q') \in \mathfrak{R}$.
- wenn $p \xrightarrow{\tau} p'$, dann existiert ein q', mit $q \xrightarrow{\tau} q'$ und $(p', X, q') \in \mathfrak{R}$.
- wenn $\mathcal{I}(p) \cap (X \cup \{\tau\}) = \emptyset$, dann $(p, q) \in \mathfrak{R}$ und
- wenn $\mathcal{I}(p) \cap (X \cup \{\tau\}) = \emptyset$, und $p \xrightarrow{t} p'$, dann $\exists q'$, mit $q \xrightarrow{t} q'$ und $(p', X, q') \in \mathfrak{R}$.

3.3.2 Beispiel





Die Prozesse p_3 und q_3 sind nicht reaktiv bisimilar. Weil p_3 eine Aktion b ausführen kann, welche q_3 aber 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 Zeit-Aktion in einer Umgebung befinden [vG20, S. 3–6], die einen b Schritt verbietet. In diesem Sinne können die t-Nachfolgern keine b Aktion ausführen.

3.4 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 sind im LTS_s stark bisimilar genau dann, wenn diese in LTS_t reaktiv bisimilar sind [Poh21, S. 39].*

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

3.4.1 Definition

Die Transformation ist wie folgt definiert:
Sei:

- $Proc$: Menge von Prozesse.
- Act: Menge von Aktionen.
- t_ε: Eine speziele Aktion.
- t: Die spezielle timeout Aktion.
- A: Eine Menge von Aktionen, mit $A = Act \setminus \{\tau, t\}$.
- E_X: Spezielle Aktion, mit $X \subseteq A$.
- $a, a', b, b', ..., z, z'$: Zustände in $Proc$.
- $p \xrightarrow{\alpha} p'$: Eine Transition von p zu p', mit $\alpha \in Act_\upsilon$.

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

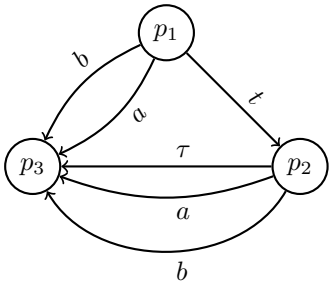
- $Proc_\upsilon = \{\mho(p) \mid p \in Proc\} \cup \{\mho_X(p) \mid p \in Proc \wedge X \subseteq A\}$.
- $Act_\upsilon = Act \cup \{t_\varepsilon\} \cup \{E_X \mid X \subseteq A\}$.
- und $\rightarrow_\upsilon$ ist bei den folgenden Regeln definiert:

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

$$3)\ \frac{p \xrightarrow{a} p'}{\mho_X(p) \xrightarrow{a}_\upsilon \mho(p')}\ a \in X \qquad 4)\ \frac{p \xrightarrow{\tau} p'}{\mho_X(p) \xrightarrow{\tau}_\upsilon \mho_X(p')}$$

$$5)\ \frac{p \overset{\alpha}{\not\rightarrow} \forall \alpha \in X \cup \{\tau\}}{\mho_X(p) \xrightarrow{t_\varepsilon}_\upsilon \mho(p)} \qquad 6)\ \frac{p \overset{\alpha}{\not\rightarrow} \forall \alpha \in X \cup \{\tau\} \quad p \xrightarrow{t} p'}{\mho_X(p) \xrightarrow{t}_\upsilon \mho_X(p')}$$

3.4.2 Beispiel



Mit t die Zeit Aktion.
Nach der Transformation sieht das System wie folgt aus:

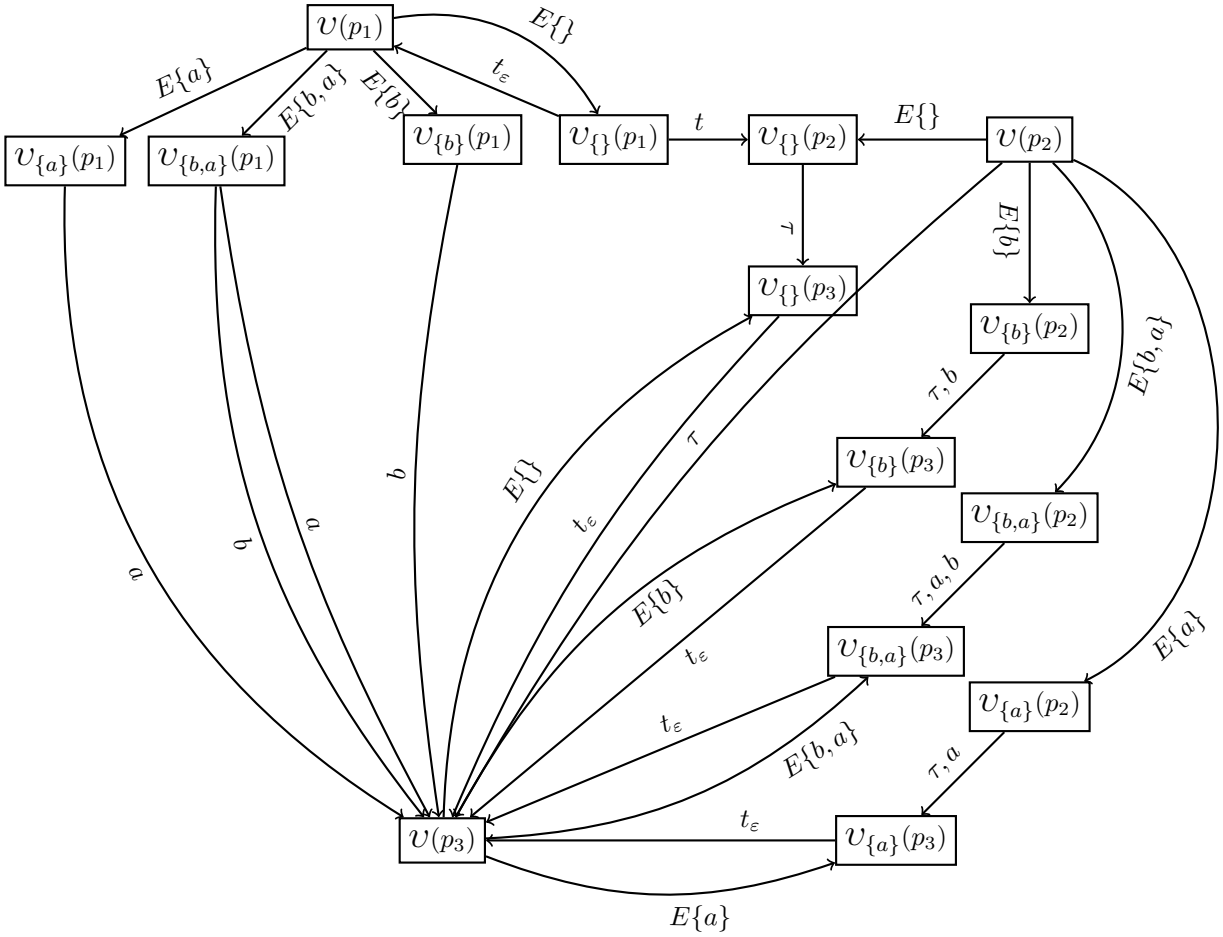


Fig. 1: Transformiertes System

Betrachte, dass in diesem Beispiel folgendes gilt:
$\mathcal{U}(p_1) \equiv p_1$, $\mathcal{U}(p_2) \equiv p_2$ und $\mathcal{U}(p_3) \equiv p_3$.
Wir sehen, dass ausgehend von jedem der drei ursprünglichen Prozessen 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_ε. Wir sehen auch, dass ausgehend von $\mathcal{U}(p_1)$ den Prozess $\mathcal{U}(p_2)$ nicht erreicht werden kann. Da die Zeit-Aktion in diesem Fall das verbietet, einen Zustand zu erreichen, wo eine a oder b Aktion möglich ist.

3.4.3 Funktionalität der Reduktion

Eine formale Erklärung der Funktionalität der Reduktion kann man in der Arbeit von **Maximilian Pohlmann** [Poh21, S. 33–41] finden. Hier erkläre ich diese Funktionalität in informeller Art und Weise, um ein einfaches Bild davon vorstellen zu können. Wir nehmen zuerst folgendes an:

- $Proc_\upsilon$: Ist die Menge aller Prozesse nach der Transformation.
- $Proc$: Ist die Menge aller Prozesse vor der Transformation.
- Act_υ: Ist 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.
- E_x: Eine Menge von Aktionen, mit $x \in \mathcal{P}(A)$.

Die Reduktion weist jedem Prozess $p \in Proc$ die Aktionsmenge E_x zu, sodass $E(\mathcal{U}(p)) = \mathcal{I}(p) \cup E_x$. Die Menge E_x werden wir als die Menge der Umgebungen nennen. Denn nach der Ausführung von $E_X \in E_x$ landet das System in einem Prozess, der nur solche Aktionen ausführen kann, die in X enthalten sind. Also das System landet in einer Umgebung, die nur bestimmte Aktionen erlaubt. Z.B. mit der Ausführung von $E_X = E_{\{a,b\}} \in E_x$ 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 $E_X \in E_x$ ausführen, sodass solche Aktionen in einen Prozess $\mathcal{U}_X(p) \in Proc_\upsilon$ überführen (Siehe fig. 1). Solange das System sich in einem solchen Prozess $\mathcal{U}_X(p)$ befindet, kann das System nur eine Aktion $\alpha \in (X \cap \mathcal{I}(p)) \cup (\{\tau, t\} \cap \mathcal{I}(p))$ ausführen. Das heißt, die ausführbaren Aktionen sind die Aktionen, die die Umgebung erlaubt und die τ sowie die Zeit-Aktion t dazu. Man kann in fig. 1 bemerken, 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 $X = \{b\}$. Vielleicht ist auch offensichtlich, dass die τ Aktion aus jedem Prozess $\mathcal{U}_X(p)$ ausführbar ist, obwohl diese eigentlich in X nicht enthalten ist. Das hat den Grund, dass die τ Aktion nicht beobachtbar ist, dementsprechend passiert sie in jedem Fall, solange $\tau \in \mathcal{I}(p)$. Die Zeit Aktion t kann aus solchen Umgebungen X' ausgeführt werden, mit $t \in \mathcal{I}(p)$ und $X' = \{e \mid e \in \mathcal{P}(A) \wedge \forall \alpha \in \mathcal{I}(p).\ \alpha \notin e\}$. Nun stellt sich die Frage: Was ist, wenn ein Prozess $\mathcal{U}(p)$ eine Aktion $E_X \in E_x$ ausgeführt hat, sodass gilt $\forall a \in \mathcal{I}(p).\ a \notin X$ und $\tau, t \notin \mathcal{I}(p)$. Dann kommt die Aufgabe von der speziellen Aktion t_ε, die von $\mathcal{U}_X(p)$ in $\mathcal{U}(p)$ zurückkehrt. Das heißt, wenn das System nach der Ausführung von eine $E_X \in E_x$ Aktion keine weitere Aktion ausführen kann, kehrt es mit der speziellen Aktion t_ε zum ursprünglichen Prozess zurück.

3.5 mCRL2 Projekt

mCRL2-Projekt [mCR19] ist eine Menge von Tools, die sich mit beschrifteten Transitionssysteme [AILS07, S. 20–25] beschäftigt. Dort kann man ein *LTS* ausgehend vom mCRl2-Spezifikationsformat [mCR19, S. 23–25] erstellen und das System dann auf bestimmte Eigenschaften überprüfen, visualisieren oder mit anderem *LTS* auf einen Äquivalenzbegriff wie starke Bisimilarität [AILS07, S. 42–61] vergleichen.
Um die Funktionsweise zu verstehen siehe [mCR19, S. 24, Fig. 2].

Die folgenden Tools verwende ich in meinem Werkzeug:

- mcrl22lps [mCR19, S. 24–34]: Transformiert ein Transitionssystem im mCRL2-Spezifikationsformat zu linearem Prozess-Spezifikationsformat (*LPS*) [mCR19, S. 24–34].
- lps2lts [mCR19, S. 24–34]: Transformiert ein Transitionssystem im linearen Prozess-Spezifikationsformat (*LPS*) zu beschriftetem Transitionssystem (*LTS*).
- ltscompare [mCR19, S. 24–34]: Vergleicht zwei beschriftete Transitionssysteme auf bestimmte Bisimilarität.
- ltsgraph [mCR19, S. 24–34]: Visualisiert ein Transitionssystem.

4 Ansatz

In diesem Abschnitt zeige ich die Funktionsweise meines Werkzeugs und dessen Aufbau.

Um einen allgemeinen Überblick zu haben, zeige ich im folgenden die Visualisierung der Arbeitsweise meines Werkzeugs.

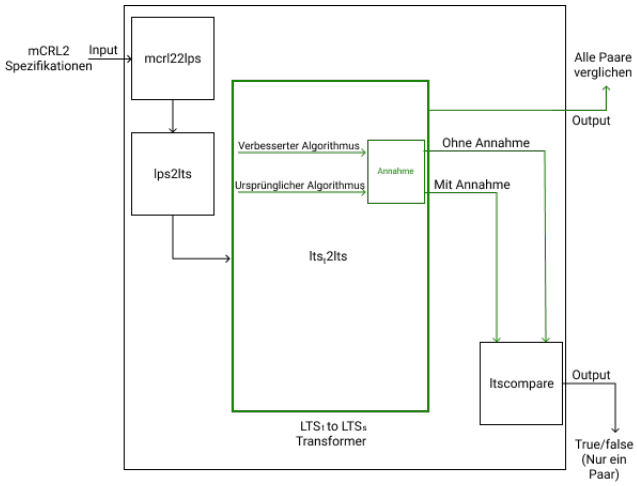


Fig. 2: Lösungsweg

Mein Ziel ist die reaktive Bisimilarität automatisiert zu überprüfen. Im Abschnitt Hintergrund ist folgendes eingeführt:

- Beschriftete Transitionssysteme (LTS_s) [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 werde ich in meinem Ansatz brauchen.

4.1 Kern Idee

Weil wir die reaktive Bisimilarität überprüfen möchten und reaktive Systeme in beschrifteten Transitionssystemen mit timeouts LTS_t [vG21a] modelliert [Poh21, S. 8] und darauf die reaktive Bisimilarität angewandt werden kann [vG20, S. 3–6], brauchen wir für mein Werkzeug nur Zwei Sachen:

1. Ein Tool, um ein LTS_t zu modellieren.
2. Ein Tool, um die reaktive Bisimilarität in einem LTS_t zu überprüfen.

Der Beginn ist mit dem ersten Punkt: Wir brauchen ein Tool, die ein beschriftetes Transitionssystem [AILS07, S. 20–25] modellieren kann, sodass die Ausgabe dieses Tools später für die Überprüfung der reaktiven Bisimilarität verwandt bzw. bearbeitet werden kann. MCRL2 kann ein beschriftetes Transitionssystem modellieren. Indem das gewünschte System im mCRL2-Spezifikationsformat [mCR19, S. 23–25] eingegeben wird. Die Transformation von 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].
2. Transformation von diesem LPS zu einem LTS_t durch das Tool lps2lts [mCR19, S. 24–34].

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 die starke Bisimilarität. Problem ist jetzt aber, dass wir die reaktive Bisimilarität überprüfen wollen und dafür gibt kein Tool. Hier kommt also der zweite Punkt.

Ein Tool zur Überprüfung der reaktiven Bisimilarität existiert nicht. Wie in der Reduktion aber zu sehen ist, hat **Maximilian Pohlmann** eine Reduktion [Poh21] definiert, die das reaktive Bisimilaritätsproblem zu starkem Bisimilaritätsproblem reduziert. Das heißt, wir brauchen nur noch die vom lps2lts-Tool ausgegebenes LTS_t bzw. Datei so zu modifizieren, sodass 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 man das resultierende modifizierende LTS_s in mCRL2 auf starke Bisimilarität überprüfen. Die stark bisimilaren Prozessen sind dann wegen Theorem 1 auch reaktiv bisimilar.

Diese Aufgabe, also das Tool zur Transformation von LTS_t zu LTS_s mit dem Einhalt des mCRL2-Dateiformats ist meine Arbeit. Mein Tool habe ich lts_t2lts genannt.

Die Implementierung des Reduktionsalgorithmus erfolgt in zwei Versionen. Die eine ist die ursprüngliche Reduktion und die andere ist eine Verbesserung davon. Beide Varianten erkläre ich in den nächsten zwei Abschnitten.

4.2 Ursprünglicher Algorithmus

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

Der Algorithmus1 bekommt drei Variable:

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

In Zeile 2 iteriere ich über alle Prozesse im eingegebenen LTS_t. Dann weise ich der Liste *ProccesEnvironmentsTupelList* ein neues Paar von dem aktuellen Prozess *process* und die Liste aller Umgebungen *EnvironmentsList* zu. In Zeile 4 wende ich den Regel 2 in der Definition der Reduktion. Sodass jeder ursprüngliche Prozess p eine neue Menge von Transitionen T, mit $T = \{t \mid t = (\mathcal{U}(p), E_X, \mathcal{U}_{\{X\}}(p)) \wedge X \in EnvironmentsList\}$ bekommt. Dann iteriere ich über alle ursprüngliche Transitionen des aktuellen Prozesses *process* und danach über alle Umgebungen dieses Prozesses *process*, die ich vorhin dem Prozess *process* zugewiesen habe. Das mache ich, da gleich 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 verwenden wir die Aktion t_ε (Zeile 19).

Nun kommen wir zu den if und else if Anweisungen in Zeilen 8, 11 und 13. Dort teste ich welche ursprüngliche Aktionen der jeweilige ursprüngliche Prozess hat. Je nachdem welche Aktion der Prozess hat, wende ich die benötigten Regeln in der Definition der Reduktion an. Hier haben wir drei Varianten:

- Die Aktion ist eine τ Aktion (Zeile 8), dann laut der Definition der Reduktion muss ich die Regeln 1 und 4 anwenden.
- Die Aktion ist eine $\alpha \in A$ (Zeile 11), dann laut der Definition der Reduktion muss ich den Regel 3 anwenden.
- Die Aktion ist eine t Aktion (Zeile 13), dann laut der Definition der Reduktion muss ich die Regeln 5 und 6 anwenden.

Damit wird das LTS_t in ein LTS_s [AILS07, S. 20–25] transformiert, so kann man nun das resultierende LTS_s mit Hilfe von mCRL2 [mCR19] auf die starke Bisimilarität [AILS07, S. 42–61] überprüfen und somit laut Reduktion auch auf reaktive Bisimilarität [vG20, S. 3–6]. Dabei erreiche ich das Ziel dieser Arbeit.

Algorithm 1 LTS_t zu LTS_s Transformationsalgorithmus ($getLTS_s$)

Require:

$processList, EnvironmentsList, ProccesEnvironmentsTupelList$
for $process \in processList$ **do**
  $ProccesEnvironmentsTupelList \leftarrow (process.getName(), EnvironmentsList)$
  $process.addNewTransitionsOfRule2(EnvironmentsList)$
  for $transition \in process.getTransitions()$ **do**
    $processEnvironments \leftarrow ProccesEnvironmentsTupelList.getValueOf(process.getName())$
    for $environment \in processEnvironments$ **do**
      if $transition.getAction().isTauAction()$ **then**
        $process.addNewTransitionsOfRule1(process, transition)$
        $process.addNewTransitionsOfRule4(process, transition, environment)$
      else if $transition.getAction().isNormalAction()$ **then**
        $process.addNewTransitionsOfRule3(process, transition, environment)$
      else if $transition.getAction().isTimeAction()$ **then**
        $process.addNewTransitionsOfRule5(process, environment)$
        $process.addNewTransitionsOfRule6(process, transition, environment)$
      end if
    end for
  end for
  $process.applyRule5OfRemainingEnvironments(ProccesEnvironmentsTupelList)$
end for

4.3 Verbesserter Algorithmus

Die Verbesserung des Algorithmus stellt sich in nur eine Sache dar. Die Variable *EnvironmentsList* im Algorithmus enthält im ursprünglichen Algorithmus alle $e \in \mathcal{P}(A)$, wo $\mathcal{P}(A)$ die Potenzmenge der angegebenen Aktionen außer $\{\tau, t\}$. Hingegen in der verbesserte Version ist $EnvironmentsList = \{\{a\} \mid a \in A\} \cup \{\{T\} \mid T \in A \setminus \mathcal{I}(p_{t_i}).i \in \mathbb{N}\}$, wo p_{t_i} ein Prozess, der eine time-out Aktion hat und $\mathcal{I}(p_{t_i})$ die initiale Aktionen von Prozess p_{t_i}.

Damit ist die Anzahl aller generierten Prozessen im Algorithmus von $|Proc_v| = |proc| \cdot (1 + 2^{|A|})$ auf $|Proc_v| = |proc| \cdot (|A| + |P_t| + 1)$ reduziert, wo $|P_t|$ die Anzahl der Prozesse ist, die mindestens eine Zeit-Aktion t haben.

Mit der verbesserten Version reduziert sich dementsprechend die Laufzeit des Algorithmus erheblich stark. 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.

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

Nach der Reduktion bekommt jedem ursprünglichen Prozess wie ich im Abschnitt Funktionalität der Reduktion erwähne die Aktionsmenge E_x 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?

Wenn man nun nach der Reduktion die Menge E_x aus dem Aktionsmenge

bei jedem ursprünglichen Prozess entfernt und stattdessen nur eine Aktion $E_{\{...\}}$ als Vertretung hinzufügt. Danach führt man den Verglich zur Überprüfung der reaktiven Bisimilarität, erhält man erstaunlicherweise das gleiche Ergebnis. Das ist der Fall bei allen meiner Stichproben. Eine ähnliche Idee erwähnt **Maximilian Pohlmann** in seiner Arbeit [Poh21, S. 46]. Diese Idee wirkt nicht auf die Effizienz der Reduktion beider Versionen aus, sondern auf den Vergleich bei der starken Bisimilarität [AILS07, S. 42–61] und somit auf das gesamte Werkzeug. Diese Annahme kann in meinem Werkzeug nur beim Vergleich eines einzigen Paars umgesetzt werden.

4.5 Überprüfung aller Paare oder nur ein Paar

Mein Werkzeug kann entweder durch mCRL2 [mCR19] ein Prozess-paar oder durch mein Tool mit der Verwendung die von mir implementierte Definition der starken Bisimilarität [AILS07, S. 42–61] alle Prozess-paare der eingegebenen Systemen überprüfen. Da die Definition der starken Bisimilarität berühmt ist, ist es genügend hier nur oberflächlich den Pseudocode vorzustellen:

Algorithm 2 Algorithmus der starken Bisimulation

Require:
1: $processList1$
2: $processList2$
3: $pairs, bisimilarPairs \leftarrow EmptyList$
4: **for** $process1 \in processList1$ **do**
5: **for** $process2 \in processList2$ **do**
6: $fuege\ (process1, process2)\ in\ pairs\ zu.$
7: **end for**
8: **end for**
9: **while** *true* **do** ▷ Bisimilarität Überprüfung
10: $entleere\ bisimilarPairs$
11: $bisimilarPairs \leftarrow strongBisimilar(pairs)$
12: **if** $!AraEqual(pairs, bisimilarPairs)$ **then**
13: $entleere\ pairs$
14: $kopiere\ bisimilarPairs\ in\ pairs$
15: **else** *break*
16: **end if**
17: **end while**

5 Evaluation

In diesem Abschnitt zeige ich 7 Beispiele von reaktiver Bisimilarität [vG20, S. 3–6] und die Überprüfungsergebnisse sowie Statistiken über den ursprünglichen und verbesserten Algorithmus.
Hinweis: Erster Eintrag in (x, y) in allen Beispiele gehört zum System1 und zweiter zum System2.
Hinweis: $a \rightarrow b \rightarrow c$ ist eine Sequenz von Aktionen beginnend von Prozess 0.
Hinweis: In allen Beispiele steht System1 auf der linken Seite und System2 auf der rechten.

Algorithm 3 strongBisimilar

Require:

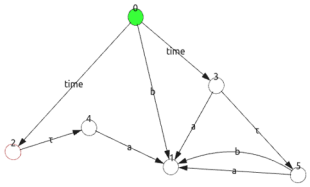
```
1: pairs
2: xStepBisimilar ← EmptyList
3: for pair ∈ pairs do
4:     if checkSimulation(false, pairs, pair) and checkSimulation(true, pairs, pair) then
5:         fuege pair in xStepBisimilar zu
6:     end if
7: end for
8: return xStepBisimilar
```

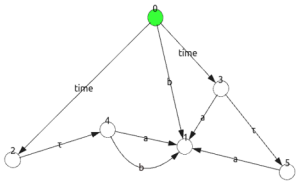
Algorithm 4 checkSimulation

Require:

```
1: leftToRight (check (p, q) or (q, p))
2: pairs
3: Prozess p from pair
4: Prozess q from pair
5: found ← false
6: for t1 ∈ p.getTransitions() do
7:     for t2 ∈ q.getTransitions() do
8:         if t1 == t2 then
9:             found ← areSuccessorsInR(leftToRight, pairs, t1.getSuccessor(), t2.getSuccessor())
10:            if found then
11:                break
12:            end if
13:        end if
14:    end for
15:    if !found then
16:        returnfalse
17:    end if
18:    found ← false
19: end for
20: returntrue
```

5.1 Beispiel 1

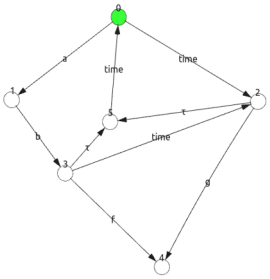


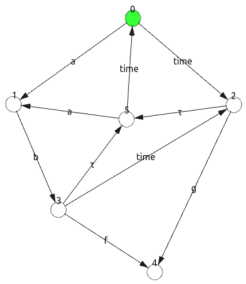


	Ursprüngliche Version	Verbesserte Version
(0, 0)	true	true
(i, j)	(0, 0), (1, 1), (4, 5), (5, 4)	(0, 0), (1, 1), (4, 5), (5, 4)
Prozesse-Anzahl	60	36
Ausführungszeit	120,3 ms	117 ms

Das Paar (0,0) ist reaktiv bisimilar, da die Zeit Aktionen von Prozess 0 in beiden Systeme die Ausführung von der Aktion b verbietet. 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.

5.2 Beispiel 2

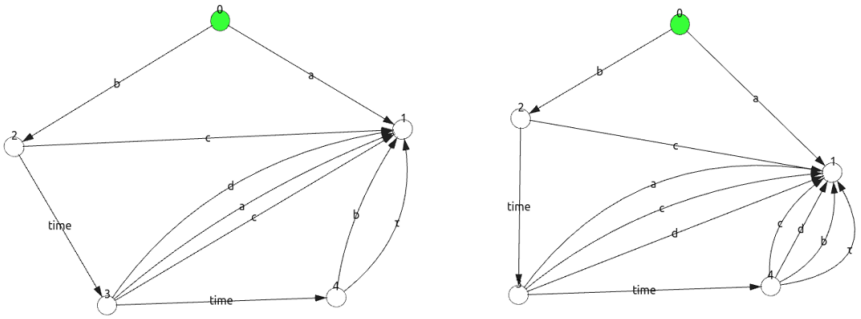




Das Paar (0,0) ist nicht reaktiv bisimilar, da im System2 einen anderen Weg anderes als $time \to \tau \to a$ gibt, wo die Aktion a erreicht wird und zwar: $a \to b \to \tau \to a$. Das Paar (4, 4) ist ein Blatt und dementsprechend reaktiv bisimilar.

	Ursprüngliche Version	Verbesserte Version
(0, 0)	false	false
(i, j)	(4, 4)	(4, 4)
Prozesse-Anzahl	204	84
Ausführungszeit	130,6 ms	122,9 ms

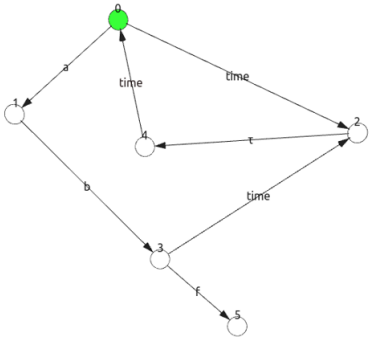
5.3 Beispiel 3

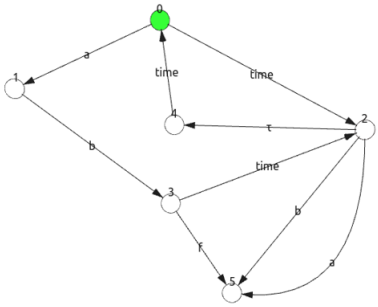


	Ursprüngliche Version	Verbesserte Version
(0, 0)	true	true
(i, j)	(0, 0), (1, 1), (2, 2), (3, 3)	(0, 0), (1, 1), (2, 2), (3, 3)
Prozesse-Anzahl	170	60
Ausführungszeit	123 ms	117,9 ms

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

5.4 Beispiel 4

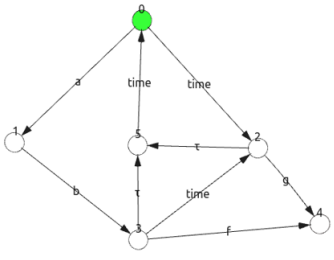


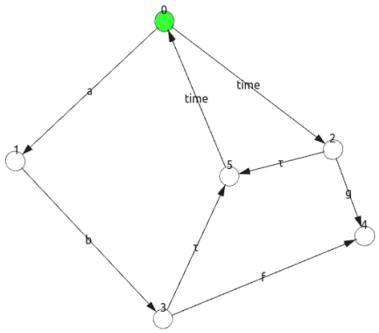


	Ursprüngliche Version	Verbesserte Version
(0, 0)	false	false
(i, j)	(5, 5)	(5, 5)
Prozesse-Anzahl	108	84
Ausführungszeit	114,7 ms	116,5 ms

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

5.5 Beispiel 5

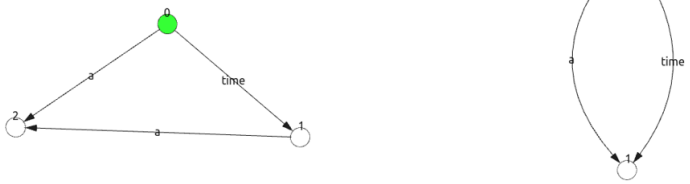




	Ursprüngliche Version	Verbesserte Version
(0, 0)	true	true
(i, j)	(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5)	(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5)
Prozesse-Anzahl	204	84
Ausführungszeit	133,7 ms	119,2 ms

Hier sind die beiden Systeme gleich, da die Zeit Aktion im System1 in Prozess 3 wegen der τ Aktion nicht betrachtet wird.

5.6 Beispiel 6



Dieses Beispiel zeigt einen interessanten Fall in der reaktiven Bisimilarität.

Wir wissen bereits, dass die Ausführung der Zeit Aktion t bestimmte Aktionen durch eine Umgebung beschränkt. Wenn wir uns das obige Beispiel angucken, stellen wir fest, dass beide Systeme eine Zeit Aktion t haben, deren Ausführung jeweils die Aktion a verbietet. Das heißt, würde die Zeit Aktion t in beiden Systemen genommen, dann wären wir in Zustand 1 in beiden Systemen. Nun hat System2 sowieso keine weitere Aktionen. Bei System1 wäre die Aktion a möglich, wenn die Zeit 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 der Definition der reaktiven Bisimilarität:

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

Somit müssen die beiden Prozessen 1 reaktiv bisimilar sein. Mein Werkzeug gibt hier natürlich $false$ aus.

5.7 Beispiel 7

Das Beispiel ist ist aus [vG21b, 46].

	Ursprüngliche Version	Verbesserte Version
Prozesse-Anzahl	21074	1148
Ausführungszeit	151,958 s	466,5 ms

Das Ziel dieses Beispiel ist zu zeigen, wie effizient die verbesserte Version des Algorithmus im Fall von großen Systemen ist. Wir sehen hier, dass die

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

6 Verwendungsweise

Mein Werkzeug ist einfach zu verwenden. Alles was man braucht ist den Ordner all_in _one in mein GitHub Repo. Dort gibt es das Bash-skript bin.sh, durch es man mein Werkzeug zur Nutze machen kann.

Anforderungen:

- JAVA $\geq$ v11.

Das Skript bekommt 3 Parameters:

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

Parameter 1 und 2 (System1 und System2) sind die Systeme, die auf die reaktive Bisimilarität [vG20, S. 3–6] überprüft werden sollen.
Parameter 3 kann eine beliebige Kombination aus der Buchstaben **i, p, a und s**, womit:

- i: Verwende der verbesserte Algorithmus bedeutet (i $\rightarrow$ improved).
- p: Vergleiche alle nicht reflexive, symmetrische Paare der beiden Systeme bedeutet (p $\rightarrow$ pairs).
- a: Wende nur eine Umgebungsaktion (a $\rightarrow$ assumption).
- s: Visualisiere die beiden Systeme durch mCRL2 bedeutet (s $\rightarrow$ show).

Beispiel ist der folgende Aufruf:
`./bin.sh system1.mcrl2 system2.mcrl2 ips`. Damit wird den verbesserten Algorithmus verwandt, alle nicht reflexive Symmetrische Paare verglichen und die Systeme angezeigt.
`./bin.sh system1.mcrl2 system2.mcrl2 ias`. Damit wird den verbesserten Algorithmus verwandt, eine Umgebungsaktion angewandt und die Systeme angezeigt.

7 Fazit

In dieser Arbeit habe ich ein Werkzeug zur Überprüfung der reaktiven Bisimilarität entwickelt. Das Werkzeug kann nur ein Paar oder alle Paare auf die reaktive Bisimilarität [vG20, S. 3–6] überprüfen. Das Werkzeug verwendet einen Algorithmus, dessen Aufwand exponentiell wächst. Eine verbesserte Version davon entwickelte ich auch, die aber nur einen linearen Aufwand hat. Mein Werkzeug

mit allen im Abschnitt Verwendungsweise erwähnten Features sind durch nur ein Skript zu verwenden. Mein Werkzeug kann in den wissenschaftlichen Arbeiten verwandt werden, die Studierende beim Erlernen der reaktiven Bisimilarität helfen und große Systeme durch die verbesserte Version des Algorithmus effizienter testen.

Die Korrektheit der verbesserten Version des Algorithmus ist nicht bewiesen. Damit könnte der Beweis dessen Korrektheit eine mögliche damit verwandte zukünftige Arbeit sein.

Eine andere mögliche verwandte zukünftige Arbeit ist der Beweis von dem, was ich im Unterabschnitt Eine einzige Umgebungsaktion $E_{\{\dots\}}$ anstatt von verschiedenen erkläre.

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.