Graphtransformationen zur Systemstabilitätsanalyse

Stephan Epp

11. Januar 2026

Inhaltsverzeichnis

1 Einführung

1.1 Problemstellung

Die Analyse komplexer Softwaresysteme erfordert Methoden zur Modellierung und Verifikation von Systemzuständen und deren Übergängen. Traditionelle Ansätze stoßen bei der Identifikation von Stabilitätszuständen, Zyklen und Ruhelagen an ihre Grenzen, insbesondere wenn Systeme durch strukturelle Transformationen beschrieben werden.

Die zentrale Problemstellung dieser Arbeit ist: **Wie lassen sich Systemzustände effizient vergleichen und analysieren, um Stabilitätseigenschaften, Ruhelagen und zyklische Verhalten in transformationsbasierten Systemen zu identifizieren?**

1.2 Motivation

Graphtransformationen sind eine Methode zur Modellierung von Zustandsübergängen in Softwaresystemen. Dabei werden:

- Systemzustände als Graphen $G = (V, E)$ dargestellt
- Zustandsübergänge durch farbcodierte Transformationsregeln $L \to R$ beschrieben
- Kontextelemente (schwarz), zu löschende Elemente (rot) und neue Elemente (grün) unterschieden

Ein praktisches Beispiel ist die Modellierung einer Ampelkreuzung, bei der jeder Ampelzustand (Grün, Gelb, Rot, Rot-Gelb) als Graph repräsentiert wird und Übergänge zwischen den Zuständen als Transformationsregeln definiert sind.

Die Systemstabilitätsanalyse ermöglicht es:

- Ruhelagen zu identifizieren (Zustände, die sich nicht mehr ändern)
- Zyklen zu erkennen (wiederkehrende Zustandsmuster)
- Längste Subgraph-Sequenzen zu finden (monotone Systemerweiterungen)
- Aussagen über Performance und Sicherheit zu treffen

Die Effizienz dieser Analyse hängt maßgeblich vom verwendeten Subgraph Algorithmus ab, der in $O(n^3)$ Zeit entscheiden kann, ob ein Graph in einem anderen enthalten ist.

2 Grundlagen

2.1 Definitionen

Definition 1 (Farbcodierter Graph)**.** Ein farbcodierter Graph ist ein Tripel $G = (V, E, c)$ mit einer Knotenmenge $V = \{v_1, v_2, \ldots, v_n\}$, einer Kantenmenge $E \subseteq V \times V$ und einer Farbfunktion $c : (V \cup E) \to \{\text{schwarz, rot, grün}\}$.

Die Farbcodierung hat folgende Bedeutung: **Schwarz**: Kontextelemente, die unverändert bleiben, **rot**: Elemente, die gelöscht werden, **grün**: Elemente, die neu erzeugt werden.

Definition 2 (Graphtransformation)**.** Eine Graphtransformation ist eine Regel $t : L \to R$ mit:

- Linke Seite L (Vorbedingung): enthält schwarze und rote Elemente
- Rechte Seite R (Nachbedingung): enthält schwarze und grüne Elemente
- Schwarze Elemente in L und R sind identisch

Definition 3 (Systemzustand)**.** Ein Systemzustand S_i zum Zeitpunkt i ist charakterisiert durch:

$$S_i = (i, t_i, G_i, T_{i-1}, A_i)$$

wobei i die Schrittnummer, t_i der Zeitstempel, G_i der Graph des Zustands, T_{i-1} die angewendete Transformation und A_i die Adjazenzmatrix von G_i sind.

Definition 4 (Subgraph-Sequenz)**.** Eine Subgraph-Sequenz ist eine Folge von Systemzuständen

$$\text{Seq} = (S_i, S_{i+1}, \ldots, S_j)$$

mit der Eigenschaft, dass $G_k \subseteq G_{k+1}$ für alle $k \in \{i, \ldots, j-1\}$.

2.2 Graphtransformationen

2.2.1 Anwendung einer Transformation

Die Anwendung einer Transformation $t : L \to R$ auf einen Graph G erfolgt in drei Schritten:

Algorithm 1 Anwendung einer Graphtransformation

Require: Graph G, Transformation $t = (L, R)$
Ensure: Transformierter Graph G'
1: **Schritt 1:** Prüfe ob L mit G übereinstimmt
2: **if** nicht übereinstimmend **then**
3: **return** Fehler
4: **end if**
5: **Schritt 2:** $G' \leftarrow$ Kopie von G
6: **Schritt 3:** Entferne alle roten Knoten aus L in G'
7: **Schritt 4:** Entferne alle roten Kanten aus L in G'
8: **Schritt 5:** Füge alle grünen Knoten aus R zu G' hinzu
9: **Schritt 6:** Füge alle grünen Kanten aus R zu G' hinzu
10: **return** G'

2.2.2 Matching-Bedingung

Eine Transformation $t : L \to R$ ist auf Graph G anwendbar, wenn:

$$\forall v \in V_L^{\text{schwarz}} : v \in V_G \tag{1}$$
$$\forall v \in V_L^{\text{rot}} : v \in V_G \tag{2}$$
$$\forall e \in E_L^{\text{schwarz}} : e \in E_G \tag{3}$$
$$\forall e \in E_L^{\text{rot}} : e \in E_G \tag{4}$$

Beispiel 1 (Ampeltransformation: Grün → Gelb)**. Linke Seite** L**:** Schwarze Knoten: `ampel`, `nord`, `süd`, rote Knoten: `grün`, rote Kanten: `ampel`→`grün`, `grün`→`nord`, `grün`→`süd`. **Rechte Seite** R**:** Schwarze Knoten: `ampel`, `nord`, `süd`, grüne Knoten: `gelb`, grüne Kanten: `ampel`→`gelb`, `gelb`→`nord`, `gelb`→`süd`.

3 Systemstabilitätsanalyse

In diesem Kapitel wird die Idee der Systemstabilitätsanalyse vorgestellt. Es werden Algorithmen zur Identifikation von Systemeigenschaften beschrieben.

Die Systemstabilitätsanalyse basiert auf der Beobachtung, dass Systeme durch wiederholte Anwendung von Transformationen bestimmte Verhaltensmuster zeigen:

1. **Monotone Erweiterung**: Jeder neue Zustand erweitert den vorherigen (Subgraph-Beziehung)
2. **Stabile Zustände**: Zustände, die sich nicht mehr ändern (Ruhelagen)
3. **Zyklisches Verhalten**: Zustände, die sich periodisch wiederholen

Für eine Sequenz von n Systemzuständen $S_0, S_1, \ldots, S_{n-1}$ wird eine Vergleichsmatrix $M \in \{0,1\}^{n \times n}$ berechnet:

$$M_{ij} = \begin{cases} 1 & \text{falls } G_i \subseteq G_j \\ 0 & \text{sonst} \end{cases} \tag{5}$$

Die Berechnung von M_{ij} erfolgt mittels des Subgraph Algorithmus in $O(n^3)$ Zeit.

Satz 1 (Komplexität der Vergleichsmatrix)**.** Die Berechnung der vollständigen Vergleichsmatrix für n Systemzustände erfordert $O(n^5)$ Zeit.

Beweis. Es sind n^2 Paare von Zuständen zu vergleichen. Jeder Vergleich mittels Subgraph Algorithmus benötigt $O(n^3)$ Zeit. Gesamtkomplexität: $O(n^2 \cdot n^3) = O(n^5)$. □

Satz 2 (Korrektheit der Sequenzberechnung)**.** Der Algorithmus 2 findet die längste zusammenhängende Subgraph-Sequenz in $O(n^2)$ Zeit.

Beweis. Die dynamische Programmierung garantiert, dass $dp[i]$ die Länge der längsten Sequenz speichert, die bei Zustand i endet. Die Rekonstruktion über die *parent*-Zeiger liefert die tatsächliche Sequenz. Die Zeitkomplexität ergibt sich aus den zwei verschachtelten Schleifen: $O(n^2)$. □

Die Identifikation der längsten Subgraph-Sequenz erfolgt durch dynamische Programmierung. Algorithmus 2 beschreibt die Arbeitsweise zur Ermittlung der längsten Subgraph-Sequenz.

Algorithm 2 Längste Subgraph-Sequenz

Require: Vergleichsmatrix $M \in \{0,1\}^{n \times n}$
Ensure: Längste Subgraph-Sequenz
$dp[i] \leftarrow 1$ für alle $i \in \{0, \ldots, n-1\}$
$parent[i] \leftarrow -1$ für alle $i \in \{0, \ldots, n-1\}$
for $i = 1$ to $n-1$ **do**
    for $j = 0$ to $i-1$ **do**
        if $M[j][i] = 1$ und $j \neq i$ **then**
            if $dp[j] + 1 > dp[i]$ **then**
                $dp[i] \leftarrow dp[j] + 1$
                $parent[i] \leftarrow j$
            end if
        end if
    end for
end for
$max\_length \leftarrow \max(dp)$
Rekonstruiere Sequenz über *parent*-Zeiger
return Sequenz

Definition 5 (Ruhelage)**.** Ein Zustand S_i ist stabil (Ruhelage), wenn:

$$G_i = G_{i+1} \quad \text{und} \quad M[i][i+1] = 1 \quad \text{und} \quad M[i+1][i] = 1 \tag{6}$$

Algorithmus 3 beschreibt die Arbeitsweise zur Identifikation von stabilen Zuständen.

Algorithm 3 Identifikation stabiler Zustände

Require: Vergleichsmatrix M, Systemzustände $\{S_0, \ldots, S_{n-1}\}$
Ensure: Menge stabiler Zustände
$stable \leftarrow \emptyset$
for $i = 0$ to $n-2$ **do**
    if $M[i][i+1] = 1$ und $M[i+1][i] = 1$ **then**
        if $A_i = A_{i+1}$ **then** ▷ Adjazenzmatrizen sind identisch
            $stable \leftarrow stable \cup \{i\}$
        end if
    end if
end for
return *stable*

Definition 6 (Zyklus)**.** Ein Zyklus liegt vor, wenn ein Zustand S_i zu einem späteren Zeitpunkt S_j $(j > i)$ identisch wiederkehrt:

$$\text{Zyklus}(i,j) \Leftrightarrow G_i = G_j \quad \text{für } j > i \tag{7}$$

Die Zykluslänge ist $l = j - i$.

Algorithmus 4 beschreibt die Arbeitsweise zur Identifikation von Zyklen im System.

Algorithm 4 Identifikation von Zyklen

Require: Systemzustände $\{S_0, \ldots, S_{n-1}\}$
Ensure: Menge von Zyklen (i, j)

```
cycles ← ∅
for i = 0 to n − 2 do
    for j = i + 1 to n − 1 do
        if A_i = A_j then                    ▷ Adjazenzmatrizen sind identisch
            cycles ← cycles ∪ {(i, j)}
        end if
    end for
end for
return cycles
```

4 Subgraph Algorithmus

In diesem Kapitel wird auf den Subgraph Algorithmus eingegangen und verdeutlicht, wie dieser Algorithmus bei der Systemstabilitätsanalyse eingesetzt wird.

Der Subgraph Algorithmus ist das zentrale Element der Systemstabilitätsanalyse. Er entscheidet in einer Laufzeit von $O(n^3)$, ob eine Subgraph-Beziehung $G_i \subseteq G_j$ existiert. Die Analyse der Effizienz hängt direkt von der Effizienz des Subgraph Algorithmus ab:

- **Vergleichsmatrix**: Die Ermittlung der Vergleichsmatrix erfordert $O(n^2)$ Aufrufe des Subgraph Algorithmus.
- **Gesamtlaufzeit**: Der Subgraph Algorithmus hat eine Laufzeit von $O(n^3)$, damit ergibt sich eine Gesamtlaufzeit von $O(n^5)$.

Um den Subgraph Algorithmus zu nutzen, müssen die erforderlichen Datenstrukturen für die Anwendung des Subgraph Algorithmus bereit gestellt werden. Die Graphklasse bietet Methoden zur Konvertierung zwischen Graph-Repräsentation und Adjazenzmatrix:

Listing 1: Graph-Adjazenzmatrix Konvertierung

```python
def to_adjacency_matrix(self) -> Tuple[np.ndarray, Dict[str, int]]:
        """Konvertiert Graph in Adjazenzmatrix."""
        node_list = sorted(self._nodes.keys())
        n = len(node_list)
        node_to_idx = {node_id: idx for idx, node_id in enumerate(node_list)}

        matrix = np.zeros((n, n), dtype=int)
        for edge in self._edges:
                i = node_to_idx[edge.from_node]
                j = node_to_idx[edge.to_node]
                matrix[i][j] = 1

        return matrix, node_to_idx
```

5 Algorithmus zur Systemstabilitätsanalyse

In diesem Kapitel wird der Algorithmus zur Systemstabilitätsanalyse beschrieben.

5.1 Arbeitsweise

Es folgt der vollständige Algorithmus zur Systemstabilitätsanalyse.

Algorithm 5 Systemstabilitätsanalyse

Require: Anfangszustand G_0, Transformationen $\{t_1, \ldots, t_m\}$, Schritte k
Ensure: Analyseergebnisse

$states \leftarrow [S_0]$ mit $S_0 = (0, \text{now}(), G_0, \text{null}, A_0)$
for $i = 1$ to k **do**
    $t \leftarrow t_{((i-1) \mod m)+1}$ ▷ Zyklische Transformation
    $G_i \leftarrow t.\text{apply}(G_{i-1})$
    $S_i \leftarrow (i, \text{now}(), G_i, t, A_i)$
    $states \leftarrow states \cup \{S_i\}$
end for

$M \leftarrow \text{Matrix}(|states| \times |states|)$
for $i = 0$ to $|states| - 1$ **do**
    for $j = 0$ to $|states| - 1$ **do**
        if $i = j$ **then**
            $M[i][j] \leftarrow 1$
        else
            $M[i][j] \leftarrow \text{Subgraph}(A_i, A_j)$
        end if
    end for
end for

$sequences \leftarrow \text{FindLongestSequences}(M)$
$stable \leftarrow \text{FindStableStates}(M, states)$
$cycles \leftarrow \text{FindCycles}(states)$

return $(M, sequences, stable, cycles)$

Satz 3 (Laufzeit)**.** Die Systemstabilitätsanalyse für Transformationsschritte mit Graphen der Größe n hat eine Laufzeit von $O(n^5)$, die von der Anzahl der Transformationsschritte abhängt.

Beweis. Der Beweis geht direkt aus der Analyse der Effizienz aus Kapitel 4 hervor. □

6 Softwaresysteme

Softwaresysteme sind heute in vielen Bereichen des Alltags notwendig für die erfolgreiche und zuverlässige Ausführung unterschiedlicher Abläufe und Aufgaben. In diesem Kapitel wird das Softwaresystem definiert und in seinen Eigenschaften betrachtet.

Definition 7 (Softwaresystem)**.** Ein Softwaresystem M besteht aus einer Menge von Zuständen Q, die miteinander verbunden sind und auf Eingaben für das Softwaresystem reagieren.

Definition 8 (Sprache eines Softwaresystems)**.** Die Sprache eines Softwaresystems $L(M)$ enthält alle Worte, die durch die Verbindung der Zustände durch die Eingabe für das Softwaresystem erzeugt werden.

Definition 9 (Dynamisches Softwaresystem)**.** Ein Softwaresystem M ist dynamisch, wenn der aktuelle Zustand Q_t, in dem sich M zum Zeitpunkt t befindet, abhängt von mindestens einem Zeitpunkt $t-1$ und mindestens einem Zustand Q'_{t-1} mit $Q'_{t-1} \neq Q_t$.

Definition 10 (Minimales Softwaresystem)**.** Ein Softwaresystem M ist minimal, wenn durch Wegnahme eines weiteren Zustandes Q die Sprache $L(M)$ verändert wird.

Von besonderem Interesse sind nur die Zustände eines Softwaresystems M, die den *Kern* des Softwaresystems bilden, ohne dabei $L(M)$ zu verändern. Minimal dynamische Softwaresysteme haben den Vorteil, dass sie leichter änderbar sind. Veränderungen an Zuständen wird dadurch weniger die Möglichkeit gegeben, die Sprache $L(M)$ zu verändern entgegen der eigentlichen Spezifikation.

Definition 11 (Verifizierbarkeit von Softwaresystemen)**.** Zwei Softwaresysteme M und M' sind verifizierbar auf $L(M) = L(M')$, wenn M und M' minimal sind.

Dadurch, dass bei der Verifikation nur noch der entscheidende Kern von Zuständen sowohl bei M und M' betrachtet werden muss, ist die Verifikation beider Systeme M und M' effizient möglich.

7 Stabilitätsanalyse von Markov-Ketten

In diesem Kapitel wird gezeigt, wie die entwickelte Methodik zur Systemstabilitätsanalyse auf Markov-Ketten angewendet werden kann. Markov-Ketten sind stochastische Prozesse, die sich natürlich durch Graphen darstellen lassen, wobei Zustände als Knoten und Übergangswahrscheinlichkeiten als gewichtete Kanten repräsentiert werden.

7.1 Grundlagen der Markov-Ketten

Definition 12 (Markov-Kette)**.** Eine zeitdiskrete Markov-Kette ist ein stochastischer Prozess $(X_t)_{t\geq 0}$ mit endlichem Zustandsraum $S = \{s_1, s_2, \ldots, s_n\}$ und der Markov-Eigenschaft:

$$P(X_{t+1} = s_j \mid X_t = s_i, X_{t-1}, \ldots, X_0) = P(X_{t+1} = s_j \mid X_t = s_i) = p_{ij} \tag{8}$$

wobei p_{ij} die Übergangswahrscheinlichkeit von Zustand s_i nach s_j ist.

Definition 13 (Übergangsmatrix)**.** Die Übergangsmatrix $P = (p_{ij})$ einer Markov-Kette ist eine stochastische Matrix mit:

$$p_{ij} \geq 0 \quad \text{und} \quad \sum_{j=1}^{n} p_{ij} = 1 \quad \text{für alle } i \tag{9}$$

Definition 14 (Stationäre Verteilung)**.** Eine Wahrscheinlichkeitsverteilung $\pi = (\pi_1, \ldots, \pi_n)$ heißt stationäre Verteilung, wenn:

$$\pi^T = \pi^T P \quad \text{und} \quad \sum_{i=1}^{n} \pi_i = 1 \tag{10}$$

7.2 Graphrepräsentation von Markov-Ketten

Eine Markov-Kette lässt sich als gewichteter gerichteter Graph $G_{MC} = (V, E, w)$ darstellen:

- Knoten $V = S$ repräsentieren die Zustände
- Kante $(s_i, s_j) \in E$ existiert genau dann, wenn $p_{ij} > 0$
- Gewichtsfunktion $w : E \to (0, 1]$ mit $w((s_i, s_j)) = p_{ij}$

Die Adjazenzmatrix A des Graphen entspricht der binären Version der Übergangsmatrix:

$$A_{ij} = \begin{cases} 1 & \text{falls } p_{ij} > 0 \\ 0 & \text{sonst} \end{cases} \tag{11}$$

7.3 Transformation von Markov-Ketten

Änderungen in Markov-Ketten können als Graphtransformationen modelliert werden. Typische Transformationen umfassen:

1. **Hinzufügen neuer Übergänge**: Neue Kanten mit Wahrscheinlichkeiten > 0 werden eingefügt (grüne Kanten), bestehende Wahrscheinlichkeiten werden renormalisiert
2. **Entfernen von Übergängen**: Kanten werden gelöscht (rote Kanten), Wahrscheinlichkeiten werden auf verbleibende Übergänge umverteilt
3. **Zustandsaggregation**: Mehrere Zustände werden zu einem einzigen Zustand zusammengefasst
4. **Zustandsverfeinerung**: Ein Zustand wird in mehrere Unterzustände aufgeteilt

Beispiel 2 (Transformation einer Drei-Zustands-Markov-Kette)**.** Betrachte eine Markov-Kette mit Zuständen $\{A, B, C\}$ und Übergangsmatrix:

$$P = \begin{pmatrix} 0.7 & 0.3 & 0 \\ 0.2 & 0.5 & 0.3 \\ 0.1 & 0.4 & 0.5 \end{pmatrix}$$

Transformation: Hinzufügen eines Übergangs $A \to C$ mit Wahrscheinlichkeit 0.1
Linke Seite L: Schwarze Knoten: A, B, C; bestehende Kanten wie in P
Rechte Seite R: Schwarze Knoten: A, B, C; grüne Kante: $A \to C$ mit Gewicht 0.1
Nach Renormalisierung:

$$P' = \begin{pmatrix} 0.64 & 0.26 & 0.1 \\ 0.2 & 0.5 & 0.3 \\ 0.1 & 0.4 & 0.5 \end{pmatrix}$$

7.4 Stabilitätskriterien für Markov-Ketten

Die Stabilitätsanalyse von Markov-Ketten untersucht folgende Eigenschaften:

7.4.1 Irreduzibilität

Definition 15 (Irreduzibilität)**.** Eine Markov-Kette heißt irreduzibel, wenn jeder Zustand von jedem anderen Zustand aus erreichbar ist. Graphtheoretisch bedeutet dies, dass der zugehörige Graph stark zusammenhängend ist.

Die Irreduzibilität lässt sich durch Analyse der Adjazenzmatrix prüfen. Nach Anwendung von Transformationen kann die Irreduzibilität verloren gehen oder wiederhergestellt werden.

Algorithm 6 Prüfung der Irreduzibilität

Require: Adjazenzmatrix $A \in \{0,1\}^{n \times n}$
Ensure: Boolean: Kette ist irreduzibel
1: Berechne Erreichbarkeitsmatrix $R = A + A^2 + \cdots + A^n$
2: **if** alle Einträge $R_{ij} > 0$ **then**
3: **return** true ▷ Alle Zustände erreichbar
4: **else**
5: **return** false ▷ Nicht irreduzibel
6: **end if**

7.4.2 Periodizität

Definition 16 (Periode)**.** Die Periode eines Zustands s_i ist:

$$d(i) = \gcd\{n \geq 1 : P_{ii}^{(n)} > 0\} \tag{12}$$

wobei $P^{(n)}$ die n-Schritt-Übergangsmatrix ist. Ein Zustand mit $d(i) = 1$ heißt aperiodisch.

Eine Markov-Kette ist aperiodisch, wenn alle Zustände aperiodisch sind. Dies ist äquivalent dazu, dass der zugehörige Graph Selbstschleifen hat oder keine bipartite Struktur aufweist.

7.4.3 Ergodizität

Definition 17 (Ergodizität)**.** Eine Markov-Kette heißt ergodisch, wenn sie irreduzibel und aperiodisch ist.

Ergodische Markov-Ketten haben eine eindeutige stationäre Verteilung π, gegen die die Kette unabhängig vom Startzustand konvergiert:

$$\lim_{n \to \infty} P^{(n)} = \begin{pmatrix} \pi^T \\ \vdots \\ \pi^T \end{pmatrix} \tag{13}$$

7.5 Analyse von Transformationssequenzen

Betrachte eine Sequenz von Markov-Ketten $MC_0, MC_1, \ldots, MC_k$, die durch Transformationen $t_1, \ldots, t_k$ auseinander hervorgehen:

$$MC_0 \xrightarrow{t_1} MC_1 \xrightarrow{t_2} \cdots \xrightarrow{t_k} MC_k \tag{14}$$

Die Systemstabilitätsanalyse kann folgende Fragen beantworten:

1. **Erhaltung der Irreduzibilität**: Bleibt die Kette nach Transformationen irreduzibel?
2. **Konvergenz zur Ergodizität**: Führen die Transformationen zu einer ergodischen Kette?
3. **Stabilität der stationären Verteilung**: Wie stark ändert sich π unter Transformationen?
4. **Strukturelle Stabilität**: Welche Subgraph-Beziehungen existieren zwischen den Zustandsgraphen?

7.6 Anwendung des Subgraph Algorithmus

Der Subgraph Algorithmus wird auf die Adjazenzmatrizen der Markov-Ketten angewendet. Für zwei Markov-Ketten MC_i und MC_j mit Adjazenzmatrizen A_i und A_j gilt:

$$A_i \subseteq A_j \quad \Leftrightarrow \quad \text{Jeder Übergang in } MC_i \text{ existiert auch in } MC_j \tag{15}$$

Dies bedeutet: Wenn $A_i \subseteq A_j$, dann hat MC_j mindestens alle Übergänge von MC_i (möglicherweise mit unterschiedlichen Wahrscheinlichkeiten).

Satz 4 (Monotone Erweiterung der Erreichbarkeit)**.** Sei $(MC_0, MC_1, \ldots, MC_k)$ eine Subgraph-Sequenz mit $A_i \subseteq A_{i+1}$ für alle i. Dann gilt:

1. Wenn MC_0 irreduzibel ist, dann sind alle MC_i irreduzibel
2. Die Menge der erreichbaren Zustände ist monoton nicht-abnehmend

Beweis. Wenn $A_i \subseteq A_{i+1}$, dann enthält MC_{i+1} alle Kanten von MC_i und möglicherweise weitere. Daher bleibt die Erreichbarkeit zwischen Zuständen erhalten oder wird erweitert. Wenn MC_0 stark zusammenhängend ist, bleibt dies in allen nachfolgenden Ketten erhalten. □

7.7 Berechnung der Vergleichsmatrix für Markov-Ketten

Für eine Sequenz von k Markov-Ketten wird die Vergleichsmatrix $M \in \{0,1\}^{k \times k}$ analog zur allgemeinen Systemstabilitätsanalyse berechnet:

$$M_{ij} = \begin{cases} 1 & \text{falls } A_i \subseteq A_j \\ 0 & \text{sonst} \end{cases} \tag{16}$$

Zusätzlich können gewichtsspezifische Vergleiche durchgeführt werden:

Definition 18 (Gewichtete Subgraph-Beziehung)**.** Eine Markov-Kette MC_i mit Übergangsmatrix P_i ist ein gewichteter Subgraph von MC_j mit Übergangsmatrix P_j, geschrieben $MC_i \subseteq_w MC_j$, wenn:

$$A_i \subseteq A_j \quad \text{und} \quad p_{ij}^{(i)} \leq p_{ij}^{(j)} \quad \text{für alle } (s_i, s_j) \in E_i \tag{17}$$

7.8 Beispiel: Wetter-Markov-Kette

Beispiel 3 (Evolution einer Wetter-Markov-Kette)**.** Betrachte ein einfaches Wettermodell mit Zuständen {sonnig, bewölkt, regnerisch}.

Anfangszustand MC_0:

$$P_0 = \begin{pmatrix} 0.7 & 0.3 & 0 \\ 0.4 & 0.4 & 0.2 \\ 0 & 0.5 & 0.5 \end{pmatrix}, \quad A_0 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Transformation 1: Hinzufügen direkter Übergang sonnig $\rightarrow$ regnerisch

$$P_1 = \begin{pmatrix} 0.65 & 0.25 & 0.1 \\ 0.4 & 0.4 & 0.2 \\ 0 & 0.5 & 0.5 \end{pmatrix}, \quad A_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Transformation 2: Hinzufügen Übergang regnerisch $\rightarrow$ sonnig

$$P_2 = \begin{pmatrix} 0.65 & 0.25 & 0.1 \\ 0.4 & 0.4 & 0.2 \\ 0.2 & 0.4 & 0.4 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Analyse:

- Subgraph-Sequenz: $A_0 \subseteq A_1 \subseteq A_2$
- MC_0 ist nicht irreduzibel (sonnig und bewölkt können nicht nach regnerisch übergehen direkt)
- MC_1 ist nicht irreduzibel (regnerisch kann nicht nach sonnig übergehen)
- MC_2 ist irreduzibel und aperiodisch (vollständig verbunden mit Selbstschleifen) $\Rightarrow$ ergodisch

Die Vergleichsmatrix:

$$M = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Dies zeigt eine monotone Erweiterung: Jeder Zustand ist Subgraph aller nachfolgenden Zustände.

7.9 Stabilität stationärer Verteilungen

Ein wichtiger Aspekt bei der Transformation von Markov-Ketten ist die Veränderung der stationären Verteilung. Für ergodische Ketten kann die Sensitivität von π bezüglich Transformationen analysiert werden.

Definition 19 (Stabilität der stationären Verteilung)**.** Die stationäre Verteilung heißt ϵ-stabil unter Transformation t, wenn:

$$\|\pi_{nach} - \pi_{vor}\|_1 \leq \epsilon \tag{18}$$

wobei $\|\cdot\|_1$ die L^1-Norm bezeichnet.

Algorithm 7 Berechnung der Stabilitätsmetrik

Require: Sequenz von Übergangsmatrizen $\{P_0, P_1, \ldots, P_k\}$
Ensure: Stabilitätsmetriken
1: *stability* $\leftarrow$ []
2: **for** $i = 0$ to k **do**
3: Berechne stationäre Verteilung π_i aus P_i
4: **if** $i > 0$ **then**
5: $\Delta_i \leftarrow \|\pi_i - \pi_{i-1}\|_1$
6: *stability*.append(Δ_i)
7: **end if**
8: **end for**
9: **return** *stability*

7.10 Zusammenfassung

Die Systemstabilitätsanalyse lässt sich erfolgreich auf Markov-Ketten anwenden. Die Graphrepräsentation ermöglicht es, den Subgraph Algorithmus zur Analyse struktureller Veränderungen einzusetzen. Wichtige Erkenntnisse sind:

- Transformationen können als farbcodierte Graphregeln modelliert werden
- Der Subgraph Algorithmus identifiziert monotone Erweiterungen der Übergangsmöglichkeiten
- Irreduzibilität und Ergodizität können durch Transformationen erreicht oder verloren werden
- Die Analyse von Transformationssequenzen zeigt die Evolution stochastischer Systeme

Die Kombination von Markov-Theorie und Graphtransformationen eröffnet neue Möglichkeiten für die Analyse stochastischer Systeme, insbesondere im Bereich der Performance-Modellierung, Zuverlässigkeitsanalyse und adaptiven Systeme.

8 Zusammenfassung

Die vorliegende Arbeit präsentiert eine umfassende Methodik zur Systemstabilitätsanalyse durch Graphtransformationen. Im Zentrum steht die Verbindung von formaler Graphtheorie, effizienten Algorithmen und praktischer Anwendbarkeit auf unterschiedliche Systemtypen.

8.1 Kernbeiträge

Die wesentlichen Beiträge dieser Arbeit lassen sich in vier Bereiche gliedern:

Formale Grundlagen: Es wurde eine formale Basis für die Modellierung von Systemzuständen durch farbcodierte Graphen entwickelt. Die Farbcodierung (schwarz für Kontextelemente, rot für zu löschende Elemente, grün für neu zu erzeugende Elemente) ermöglicht eine intuitive und gleichzeitig präzise Beschreibung von Zustandsübergängen. Die Definition von Transformationsregeln als Paare $L \to R$ folgt etablierten Konzepten der algebraischen Graphtransformation und bietet eine solide theoretische Fundierung.

Effizienter Algorithmus: Die entwickelte Systemstabilitätsanalyse basiert maßgeblich auf dem Subgraph Algorithmus, der in $O(n^3)$ Zeit entscheiden kann, ob eine Subgraph-Beziehung zwischen zwei Graphen existiert. Diese Effizienz ist entscheidend für die praktische Anwendbarkeit, da für eine Sequenz von k Systemzuständen insgesamt $O(k^2 \cdot n^3) = O(n^5)$ Laufzeit erreicht wird. Ohne den effizienten Subgraph Algorithmus wäre die Analyse realistischer Systeme mit naiven Ansätzen ($O(n!)$ für Permutationsprüfung) nicht durchführbar.

Umfassende Analysemöglichkeiten: Die Methodik ermöglicht die Identifikation verschiedener Systemeigenschaften. Durch die Berechnung der Vergleichsmatrix M können längste Subgraph-Sequenzen identifiziert werden, die monotone Systemerweiterungen repräsentieren. Die Erkennung stabiler Zustände (Ruhelagen) erfolgt durch Prüfung auf identische aufeinanderfolgende Systemzustände. Zyklen werden durch paarweisen Vergleich aller Zustände in der Sequenz erkannt. Diese drei Analysetypen liefern komplementäre Einblicke in das Systemverhalten.

Anwendung auf Markov-Ketten: Die Erweiterung der Methodik auf Markov-Ketten demonstriert die Vielseitigkeit des Ansatzes. Durch Interpretation von Übergangswahrscheinlichkeiten als Kantengewichte und Verwendung der binären Adjazenzmatrix für strukturelle Vergleiche lassen sich stochastische Systeme analysieren. Die Untersuchung von Irreduzibilität, Periodizität und Ergodizität zeigt, wie graphtheoretische Eigenschaften mit stochastischen Konzepten verknüpft werden können. Die Analyse der Stabilität stationärer Verteilungen unter Transformationen eröffnet neue Perspektiven für die Bewertung adaptiver stochastischer Systeme.

8.2 Methodische Stärken

Die entwickelte Methodik zeichnet sich durch mehrere charakteristische Stärken aus:

Die **Skalierbarkeit** des Ansatzes ermöglicht die Analyse von Systemen unterschiedlicher Größe. Während kleine Systeme (wenige Knoten, wenige Transformationsschritte) in Sekundenbruchteilen analysiert werden können, bleiben auch Systeme mit mehreren hundert Zuständen noch praktikabel analysierbar. Die polynomielle Laufzeit ($O(n^5)$) steht im starken Kontrast zu exponentiellen oder faktoriellen Komplexitäten naiver Ansätze.

Die **Modularität** der Implementierung erlaubt es, einzelne Komponenten unabhängig voneinander zu entwickeln und zu testen. Die Graphklasse, die Transformationslogik, der

Subgraph Algorithmus und die Stabilitätsanalyse sind sauber getrennt und über klar definierte Schnittstellen verbunden. Dies erleichtert Erweiterungen und Anpassungen erheblich.

Die **Verifikation** der Ergebnisse ist durch die formale Fundierung gewährleistet. Sätze über die Korrektheit der Algorithmen (Sequenzberechnung, Subgraph-Erkennung) garantieren, dass keine falschen Positiven oder falschen Negativen auftreten. Die Vergleichsmatrix M liefert eine vollständige und korrekte Übersicht über alle Subgraph-Beziehungen in der Zustandssequenz.

Die **Interpretierbarkeit** der Ergebnisse unterstützt praktische Anwendungen. Die Vergleichsmatrix, Subgraph-Sequenzen, stabile Zustände und Zyklen können direkt interpretiert werden und liefern unmittelbar verwertbare Aussagen über Systemeigenschaften. Dies unterscheidet den Ansatz von Black-Box-Methoden, deren Ergebnisse oft schwer nachvollziehbar sind.

8.3 Vergleich mit verwandten Ansätzen

Die entwickelte Methodik positioniert sich im Kontext verwandter Forschungsrichtungen:

Im Vergleich zu **klassischen Graphtransformationssystemen** (z.B. nach Ehrig et al.) bietet der Ansatz eine spezialisierte Fokussierung auf Stabilitätsanalyse. Während klassische Systeme primär die Anwendbarkeit und Terminierung von Transformationen untersuchen, liegt hier der Schwerpunkt auf der Analyse von Zustandssequenzen und der Identifikation von Verhaltensmustern.

Gegenüber **Model-Checking-Verfahren** für Zustandsautomaten unterscheidet sich der Ansatz durch die graphbasierte Repräsentation und die explizite Subgraph-Analyse. Model-Checker arbeiten typischerweise mit temporaler Logik und symbolischen Methoden, während hier strukturelle Grapheigenschaften im Vordergrund stehen. Die Ansätze sind komplementär und könnten kombiniert werden.

Im Vergleich zu **Markov-Analyse-Werkzeugen** (PRISM, STORM) bietet die Methodik eine explizite Transformationsperspektive. Statt nur eine gegebene Markov-Kette zu analysieren, werden Sequenzen von Ketten und deren Evolution betrachtet. Dieses Vorgehen ermöglicht Aussagen über die Entwicklung stochastischer Systeme unter strukturellen Änderungen.

8.4 Beitrag zur Forschung

Die vorliegende Arbeit leistet mehrere Beiträge zur Forschung im Bereich formaler Methoden und Systemanalyse:

Methodisch wird die Verbindung von Graphtransformationen und effizienter Subgraph-Erkennung zur Stabilitätsanalyse etabliert. Dies schließt eine Lücke zwischen theoretischen Graphtransformationssystemen und praktisch anwendbaren Analysemethoden.

Algorithmisch demonstriert die Arbeit, wie der Subgraph Algorithmus mit $O(n^3)$ Laufzeit als Grundbaustein für komplexere Analysen dienen kann. Die Reduktion von faktorieller auf polynomielle Komplexität durch Nutzung zyklischer Rotationen statt vollständiger Permutationen ist zentral für die Praktikabilität.

Anwendungsbezogen zeigt die Erweiterung auf Markov-Ketten, wie die Methodik auf stochastische Systeme übertragen werden kann. Die Analyse von Irreduzibilität, Ergodizität und Stabilität stationärer Verteilungen unter Transformationen eröffnet neue Perspektiven für adaptive stochastische Modelle.

8.5 Ausblick

Für die Weiterentwicklung der Systemstabilitätsanalyse durch Graphtransformationen ergeben sich vielfältige Forschungsrichtungen, die sich in drei Bereiche gliedern lassen: Optimierungen des bestehenden Algorithmus, konzeptionelle Erweiterungen sowie neue Anwendungsgebiete.

Im Bereich der Optimierungen bietet die Parallelisierung der Berechnung der Vergleichsmatrix erhebliches Potenzial. Da die einzelnen Subgraph-Vergleiche voneinander unabhängig sind, können sie problemlos auf mehrere Prozessorkerne verteilt werden. Dies würde bei modernen Multi-Core-Systemen eine nahezu lineare Beschleunigung ermöglichen und die Gesamtlaufzeit deutlich reduzieren.

Ein weiterer vielversprechender Ansatz ist die inkrementelle Analyse. Anstatt bei jedem neuen Systemzustand die gesamte Vergleichsmatrix neu zu berechnen, könnten nur die Vergleiche des neuen Zustands mit allen bestehenden Zuständen durchgeführt werden. Dies würde die Laufzeit für jeden zusätzlichen Schritt reduzieren und die Analyse von langfristigen Systemverläufen erheblich beschleunigen.

Für sehr große Systeme mit mehr als tausend Zuständen könnten approximative Verfahren eingesetzt werden. Diese würden einen bewussten Trade-off zwischen Genauigkeit und Laufzeit eingehen, indem sie beispielsweise nur eine Stichprobe von Zuständen vergleichen oder heuristische Abbruchkriterien verwenden. Dadurch ließe sich die Analyse auch auf Systeme anwenden, für die eine vollständige Berechnung zu zeitaufwändig wäre.

Konzeptionell eröffnen sich zahlreiche Erweiterungsmöglichkeiten des Grundansatzes. Die Integration gewichteter Graphen würde es ermöglichen, quantitative Eigenschaften von Systemzuständen zu modellieren. Kantengewichte könnten beispielsweise Wahrscheinlichkeiten, Kosten oder Kapazitäten repräsentieren. Der Subgraph Algorithmus müsste entsprechend erweitert werden, um nicht nur strukturelle Gleichheit, sondern auch die Verträglichkeit von Gewichten zu prüfen.

Attributierte Graphen stellen eine weitere wichtige Erweiterung dar. Dabei würden Knoten und Kanten mit Attributen versehen, die bei den Vergleichen berücksichtigt werden müssen. Dies ist besonders relevant für die Modellierung realer Systeme, bei denen Objekte durch vielfältige Eigenschaften charakterisiert sind. Der Vergleichsalgorithmus müsste dann sowohl strukturelle als auch attributbasierte Übereinstimmungen prüfen.

Probabilistische Transformationen würden die Modellierung nicht-deterministischer Systeme ermöglichen. Anstatt deterministischer Regeln könnten Transformationen mit Wahrscheinlichkeiten versehen werden, was die Analyse stochastischer Systeme erlaubt. Die Stabilitätsanalyse müsste dann um statistische Methoden erweitert werden, um Aussagen über die erwartete Systemdynamik treffen zu können.

Die Unterstützung hierarchischer Graphen würde Systeme mit mehreren Abstraktionsebenen zugänglich machen. Knoten könnten selbst wieder Graphen repräsentieren, was die Modellierung komplexer Systemarchitekturen erheblich vereinfachen würde. Der Subgraph Algorithmus müsste rekursiv auf verschiedenen Hierarchieebenen operieren können.

Schließlich wäre die Integration von Echtzeitaspekten für viele praktische Anwendungen essentiell. Durch die Berücksichtigung von Zeitbedingungen und Deadlines bei Transformationen könnten zeitkritische Systeme analysiert werden. Die Stabilitätsanalyse müsste dann auch temporale Eigenschaften wie maximale Antwortzeiten oder Periodizität untersuchen.

Die entwickelte Methodik eröffnet vielfältige neue Anwendungsgebiete. Im Bereich der Cyber-Physical Systems könnten eingebettete Systeme und IoT-Anwendungen analysiert werden. Die Wechselwirkung zwischen physikalischen Prozessen und Software-Steuerung

ließe sich durch Graphtransformationen elegant modellieren, wobei die Stabilitätsanalyse kritische Systemzustände identifizieren könnte.

Ein besonders aktuelles Anwendungsfeld ist die Blockchain-Technologie. Die Verifikation von Smart Contract Zustandsübergängen stellt eine große Herausforderung dar, da Fehler in Contracts zu erheblichen finanziellen Verlusten führen können. Durch Modellierung der Contract-Zustände als Graphen und der Transaktionen als Transformationen könnte die Stabilitätsanalyse unerwünschte Zustände oder Endlosschleifen frühzeitig erkennen.

In der Systembiologie würde die Modellierung von Gen-Regulationsnetzwerken profitieren. Gene und ihre Interaktionen lassen sich natürlich als Graphen darstellen, während Regulationsprozesse durch Transformationen beschrieben werden können. Die Stabilitätsanalyse könnte helfen, stabile Expressionsmuster oder oszillierende Genaktivitäten zu identifizieren.

Schließlich bieten soziale Netzwerke ein interessantes Anwendungsgebiet für die Analyse von Netzwerkdynamiken und Informationsfluss. Die Evolution sozialer Strukturen durch das Hinzufügen oder Entfernen von Verbindungen ließe sich als Folge von Graphtransformationen modellieren. Die Stabilitätsanalyse könnte stabile Communities identifizieren oder die Ausbreitung von Informationen durch das Netzwerk verfolgen.

8.6 Fazit

Die Kombination von Graphtransformationen und dem Subgraph Algorithmus bietet eine leistungsfähige Methode zur Systemstabilitätsanalyse. Die Effizienz des Ansatzes ermöglicht die Analyse realistischer Systeme, während die formale Fundierung korrekte und nachvollziehbare Ergebnisse garantiert.

Die praktische Anwendbarkeit wurde durch die Implementierung in Python und Beispiele wie die Ampelkreuzung und Wetter-Markov-Kette demonstriert. Die Analyse liefert wertvolle Einblicke in Systemverhalten, Stabilitätseigenschaften und potenzielle Fehlerquellen. Die erfolgreiche Anwendung auf sowohl deterministische Systeme (Ampelsteuerung) als auch stochastische Systeme (Markov-Ketten) zeigt die Vielseitigkeit des Ansatzes.

Der Subgraph Algorithmus ist dabei das zentrale Element, das durch seine Effizienz und Korrektheit die gesamte Analyse erst praktikabel macht. Ohne diese effiziente Subgraph-Erkennung mit $O(n^3)$ Laufzeit wäre die paarweise Vergleichsmatrix-Berechnung für realistische Systemgrößen nicht durchführbar. Die Reduktion von faktorieller auf polynomielle Komplexität durch Verwendung zyklischer Rotationen statt vollständiger Permutationen ist der Schlüssel zum Erfolg.

8.7 Implementierung

Der Python Code für die Implementierung des Algorithmus der Systemstabilitätsanalyse und die Tests wurden mit Claude AI generiert. Der Code ist verfügbar unter: `https://github.com/hjstephan/graph-trans`. In diesem Repository befindet sich auch der generierte Code Coverage Report im HTML Format.