Auftragssysteme

Tim Jammer

Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik, Arbeitsbereich TGI
Proseminar Nebenläufigkeit SS 14

Zusammenfassung. In der vorliegenden Arbeit werden zuerst die theoretischen Grundlagen für Auftragssysteme erläutert. Auf dieser Basis wird dann die Serialisierbarkeit von Auftragssystem behandelt. Außerdem wird der Begriff eines funktionalen Auftragssystems vorgestellt. Um einen kurzen Einblick in die praktische Verwendung zu bekommen, werden anschließend auch noch Commit-Protokolle kurz vorgestellt.

Schlüsselwörter: Auftragssysteme, Serialiserbarkeit, funktionale Auftragssysteme, Commit-Protokolle

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Das Thema der Auftragssysteme befasst sich mit der allgemeinen Theorie über die nebenläufige Ausführung von Aufträgen. Die Nebenläufigkeit kommt dabei dadurch zustande, dass zwei Aufträge keinen direkten Zusamenhang haben, prinzipiell also auch gleichzeitig ausgeführt werden können.
Man beschäftigt sich dabei unter anderem damit, folgende Fragen im Kontext des gesamten Auftragssystems (und nicht nur für einen einzelnen Auftrag) zu betrachten:

- In welcher Reihenfolge können die Aufträge ausgeführt werden?
- Spielt die Reihenfolge der Ausführung eine Rolle?
- Gibt es eine optimale Reihenfolge?
- Wie beeinflussen die einzelnen Aufträge das Gesamtergebnis?

Im Folgenden wollen wir uns vorrangig mit der Reihenfolge der Ausführung von Aufträgen befassen.
Dabei sollen dem Leser die Begrifflichkeiten im Zusammenhang von Auftragssystemen näher gebracht werden, sodass man ein grundlegendes Verständnis von der allgemeinen Theorie bekommt.
Dabei werden wir uns im ersten Abschnitt mit der Einführung der allgemeinen Begrifflichkeiten befassen.
Im zweiten Abschnitt wird die Frage behandelt, inwiefern Auftragssysteme serialiserbar sind.
Der dritte Abschnitt widmet sich der Funktionalität von Auftragssystemen.
Im letzten werden Abschnitt Commit-Protokolle vorstellt, um einen Ausblick darauf zu geben, wie man dann die Atomrarität jedes einzelnen Auftrags gewährleisten kann.

2 Definition Auftragssystem

Definition 1. *Auftragssystem:*
Ein Auftragssystem $AS = (A, \prec)$ *besteht aus*
einer endlichen Menge A *von Aufträgen*
und einer Präzedenzrelation $\prec$ *, welche irreflexiv und transitiv ist.*

Dabei gibt die Präzendenzrelation $\prec$ an, welcher Auftrag die vorhergehende Ausführung welchen anderen Auftrags vorraussetzt.
$(a_1, a_2) \in \prec$ wird auch als $a_1 \prec a_2$ notiert, gesprochen wird dies als a_1 präzedent zu a_2. Gemeint ist damit, dass vor der Ausführung des Auftrags a_2 der Auftrag a_1 abgeschlossen sein muss.
Ein Auftrag a_1 ist *direkt präzendent* zu a_2, wenn $a_1 \prec a_2$ gilt, aber für kein $a_i \in A$ gilt: $a_1 \prec a_i \prec a_2$.
Die Relation $\prec$ ist irreflexiv, damit kein Auftrag seine eigene Ausführung voraussetzen kann. Die Transitivität leuchtet auch sofort ein: wenn ein Auftrag a_2

erst nach a_1 ausgeführt werden kann, und $a_2 \prec a_3$ gilt, ist die logische Folgerung, dass vor der Ausführung von a_1 keinesfalls a_3 ausgeführt werden kann, also auch $a_1 \prec a_3$ gilt. Aus der Irreflexivität und Transitivität ergibt sich, dass $\prec$ nicht symmetrisch sein kann. Auch dies ist anschaulich klar: da dann wenn $a_1 \prec a_2$ und $a_2 \prec a_1$ gilt, nie einer der beiden Aufträge ausgeführt werden kann.
Der Sonderfall $\prec = \emptyset$ spielt keine wesentliche Rolle, da dann die Reihenfolge der Ausführung gar keine Rolle spielt, kann also bei der weiteren Betrachtung weggelassen werden.
Man kann die Präzendenzrelasion auch als Präzendenzgraphen darstellen. Man verwendet allerdings häufiger die Konnektivitätsgraphen, in denen man nur die Direkten Präzendenzen einträgt, und die Kanten, die sich aus der Transitivität ergeben weglässt (Hasse Diagramm), da diese Form der Darstellung übersichtlicher und Kompakter ist. Um den kausalen Zusammenhang in der

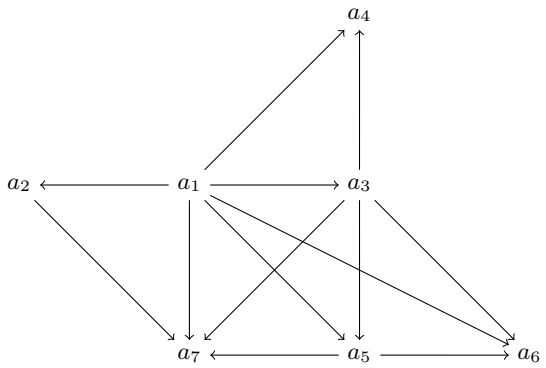

Abb. 1. Darstellung als Präzendenzgraph

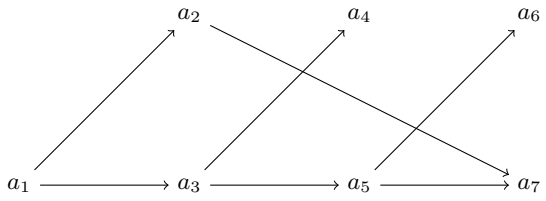

Abb. 2. Darstellung als Konnektivitätsgraph

Reihenfolge der Ausführung noch besser zu veranschaulichen, kann man ein Auftragssystem auch als Kausalnetz darstellen: Sofort einleuchtend ist, dass diese Form der Darstellungen äquivalent sind. Wir werden im folgenden Hasse-

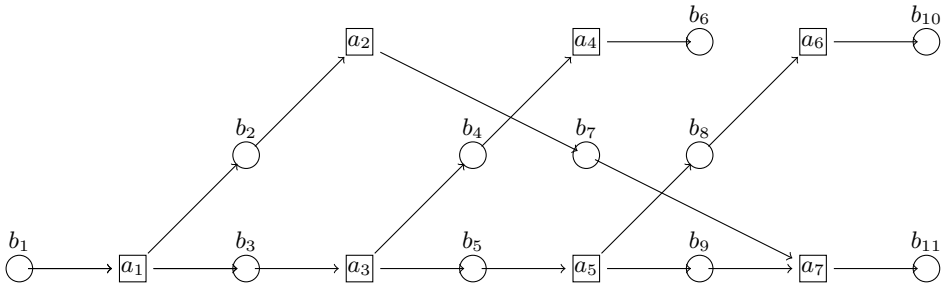

Abb. 3. Darstellung als Kausalnetz

Diagamme verwenden, da sie eine kompakte Darstellung ermöglichen.

3 Serialisierbarkeit

Für viele Kontexte ist es hilfreich zu wissen, ob ein Auftragssystem serialisierbar ist, z.B. für Datenbanksysteme. Daher werden wir uns im folgendem Abschnitt mit der Serialisierung von Auftragssystemen befassen.
Hierzu müssen wir zunächst einige weitere Begrifflichkeiten einführen:

3.1 Schematisches AS

Definition 2. *Schematisches Auftragssystem:*
Ein Auftragssystem $AS = (A, \prec)$ *ist schematisch, wenn*
die Auftragsmenge aus Aufträgen der Form $\{l_i, s_i\}$ *besteht, sodass* $A = \{l_1, s_1, \ldots, l_n, s_n\}$ *mit* $n \geq 1$. *Und außerdem gilt:*
l_i, s_i *sind für alle* $i \in \mathbb{N}$ *direkt präzendent, und sonst nur direkte Präzendenzen der Form* $s_i \prec l_j$ *existieren.*
Ferner sei eine Menge V *von Variablen definiert, so dass zu jedem* $l_i \in A$ *eine Menge* $ein_i \subseteq V$ *von Eingangsvariabeln und für jedes* $s_i \in A$ *eine Menge* $aus_i \subseteq V$ *von Ausgangsvariabeln definiert ist.*

Die Aufträge l_i sind die Leseaufträge. Die Aufträge s_i bezeichnet man als Schreibaufträge. Die Lese- und Schreibaufträge sind jeweils direkt präzendent, sodass die Ausführung eines Schreibauftrags immer die des dazugehörigen Leseauftrags voraussetzt. Da sonst nur direkte Präzendenzen zwischen einem Lese- und dem "nächsten" Schreibauftrag existieren, können l_i und s_i als ein Auftrag a_i aufgefasst werden (Vergröberung).

Definition 3. *Vergröberung:*
Die Vergröberung $\widehat{AS} = \widehat{A}, \prec$ *eines schematischen Auftragssystems ist durch* $\widehat{A} = \{a_1, \ldots, a_n\}$

$\prec = \{(a_i, a_j)\}^+$ *wenn* $s_i \prec l_j$ *gilt*[1]
gegeben, wobei die Mengen ein_i *und* aus_i *beibehalten werden.*

Das vergröberte Auftragssystem ist äquivalent zum schematischen AS, da keine Information "verloren" geht. Man kann ein schematisches Auftragssystem auch

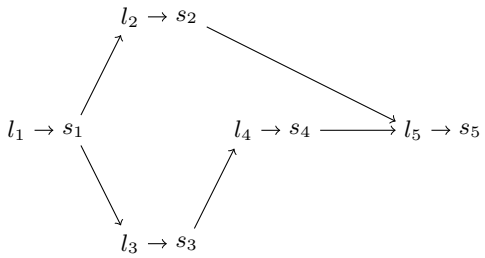

Abb. 4. Ein Schematisches Auftragssystem als Hasse Diagramm

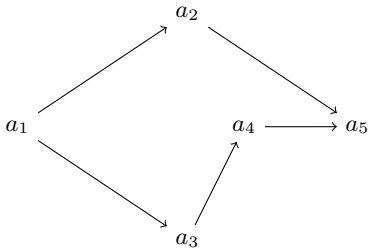

Abb. 5. Die dazugehörige Vergröberung als Hasse Diagramm

durch die Schreibweise $A = \{l_1[ein_1], s_1[aus_1], \ldots, l_n[ein_n], s_n[aus_n]\}$ und die Angabe von $\prec$ eindeutig darstellen.

In der weiteren Betrachtung (und auch in der Anwendung) ist es nützlich, vor allen anderen Aufträgen einen Initialisierungsauftrag zu haben, der alle Variablen beschreibt (also initialisiert). Und außerdem einen Ausgabeauftrag, der zuletzt alle Variablen liest, um sie im Anwendungskontext auszugeben z.B. auf dem Bildschirm oder auf einem Drucker. Ein solches Auftragssystem nennt man ein vollständiges AS.

Definition 4. *Vollständiges Auftragssystem:*
Ein Auftragssystem ist vollständig, wenn gilt:

[1] Mit R^+ ist die transitive Hülle der Relation R bezeichnet. vgl. [And13]

$ein_1 = \emptyset$ *und* $aus_1 = V$
$ein_n = V$ *und* $aus_n = \emptyset$
Außerdem muss für alle $l_i \in A$ *gelten:* $s_1 \prec l_i$.
Und für alle $s_i \in A$ *muss gelten:* $s_i \prec l_n$,
mit Ausnahme von $l_1 \prec s_1$ *und* $s_n \prec l_n$

Sinnvollerweise wird die Definition nur für $n \geq 3$ betrachtet, da ein Auftragssystem, dass nur aus Initialisierung und Ausgabe besteht in der Praxis nicht relevant ist.

Beispiel 1. [2]
$AS = (A, \prec)$ mit
$A = \{l_1[\emptyset], s_1[ab], l_2[a], s_2[b], l_3[b], s_3[a], l_4[a], s_4[a], l_5[ab], s_5[\emptyset]\}$
$\prec = \{(l_1, s_1), (l_2, s_2), (l_3, s_3), (l_4, s_4), (l_5, s_5), (s_1, l_2), (s_1, l_3), (s_2, l_5), (s_3, l_4), (s_4, l_5)\}^+$

Ist das vollständiges AS aus Abb. 4 mit der Vergröberung von Abb. 5:
$\widehat{AS} = \widehat{A}, \prec$
$\widehat{A} = \{a_1, a_2, a_3, a_4, a_5\}$
$\prec = \{(a_i, a_j)\}^+$ wenn $s_i \prec l_j = \{(a_1, a_2), (a_1, a_3), (a_3, a_4), (a_4, a_5), (a_2, a_5)\}^+$

Ein schematisches Auftragssystem nennt man auch uninterpretiert, da es nur die statischen Abhängigkeiten der Aufträge zueinander beschreibt.

3.2 Interpretation

Im vorhergehenden Abschnitt haben wir uns mit schematischen AS befasst, nun wollen wir diese interpretieren:

Definition 5. *Interpretation eines vergröberten AS:*
Eine Interpretation I wird durch eine Wertemenge D, eine Funktion $f : (a_i, z) \rightarrow z$ *und einen Startzustand* z_{start} *dargestellt.*
Wobei z der Zustand ist und durch einen Vektor $d \in D^p$ *der Länge p dargestellt wird, so dass jedem* $v_i \in V$ *ein Wert* $d_i \in D$ *zugewiesen wird.*

Betrachtet man vollständige AS, so kann der Startzustand weggelassen werden, da er ja sowieso durch den Initialisierungsauftrag überschrieben wird.
Diese Definition ist für die Vergröberung eines schematischen AS gültig, wenn die Funktion, die jedem Tupel aus Auftrag und Zustand einen Folgezustand zuordnet, nur die Eingangsvariablen ein_i beachtet, und nur die Ausgangsvariablen aus_i ändert.
Im allgemeinen Fall ist die Definition komplexer [VJ87, S.151], aber obige Fassung enthält alle benötigten Informationen, um eine Ausführungsfolge zu beschreiben. Definition 5 ist angelehnt an die Definition von Zustandsübergängen in der Automatentheorie vgl. [KBH14] und ermöglicht so ein intuitives Verständnis, dass jeder Auftrag einfach die Belegung der Variablen ändert.

[2] Mit $\{\}^+$ ist die Transitive hülle gemeint vgl. [And13]

Definition 6. *Ausführungsfolge:*
Eine Ausführungsfolge w wird durch eine (geordnete) Abfolge von Aufträgen bestimmt, dabei müssen die Präzendenzen eingehalten werden.
Die bezüglich der Interpretation I zugehörige Zustandsfolge ist die Abfolge der Zustände die sich ergibt, wenn man f (wiederholt) auf den aktuellen Zustand und den aktuellen Auftrag anwendet.
Man definiert res(w,I) als letzten Zustand dieser Abfolge.

Dazu ein Beispiel:

Beispiel 2. Gegeben sei das Auftragssystem aus Beispiel 1. Sei I gegeben durch: $D = \{0, 1\}$

$$\begin{aligned} f(a_1, z) &\to (0, 1) \\ f(a_2, (0, d)) &\to (0, 0) \\ f(a_2, (1, d)) &\to (1, 1) \\ f(a_3, (d, 0)) &\to (0, 0) \\ f(a_3, (d, 1)) &\to (1, 1) \\ f(a_4, (0, d)) &\to (1, d) \\ f(a_4, (1, d)) &\to (0, d) \\ f(a_5, (d, f)) &\to (d, f) \end{aligned}$$

mit $d, f \in D$

Dann ist w= a_1, a_3, a_4, a_2, a_5 eine Ausführungsfolge und besitzt bezüglich I folgende Zustandsfolge:

$$\begin{aligned} z_1 &= (0, 1)\ (a_1 \text{ ausgeführt}) \\ z_2 &= (1, 1)\ (a_3 \text{ ausgeführt}) \\ z_3 &= (0, 1)\ (a_4 \text{ ausgeführt}) \\ z_4 &= (0, 0)\ (a_2 \text{ ausgeführt}) \\ z_5 &= (0, 0)\ (a_5 \text{ ausgeführt}) \end{aligned}$$

Daher ist $res(w, i) = (0, 0)$

Für den allgemeinen Fall eines schematischen Auftragssystems, der betrachtet werden soll, reicht Definition 5 jedoch nicht aus. Betrachtet werden im Folgenden vollständige schematische AS.

Definition 7. *Interpretation eines schematischen AS:*
eine Interpretation I ist gegeben durch eine Wertemenge D und Funktionen $f_l : (l_i, eingabe_i) \to lok_i$ *und* $f_s : (s_i, lok_i, z) \to z$
Wobei z der Zustand ist und durch einen Vektor $d \in D^p$ *der Länge p dargestellt wird, so dass jedem* $v_i \in V$ *ein Wert* $d_i \in D$ *zugewiesen wird.*

Und lok_i eine Menge an lokalen Variablen des Auftrags a_i mit $|lok_i| = |aus_i|$
Und $eingabe_i$, einem Vektor über D, der nur diejenigen Werte des Vektors z enthält, die durch ein_i festgelegt sind.
Dabei ist die Funktion f_s so definiert, dass sie die Stellen des Vektors z, die durch aus_i festgelegt sind, mit den Werten lok_i überschreibt.

In Anlehnung an Definition 5 drückt diese Definition nun auch aus, dass der Leseautrag nicht unbedingt dem Schreibauftrag direkt vorausgehen muss.

3.3 Äqivalenz von Ausführungsfolgen

Definition 8. *Äquivalenz von Ausführungsfolgen:*
*zwei Ausführungsfolgen w_1 und w_2 sind äquivalent, wenn für **alle** Interpretationen I gilt:*
$res(w_1, I = res(w_2, I)$

Um nun über die Äquivalenz von Ausführungsfolgen zu entscheiden, muss man alle möglichen Interpretationen betrachten. Stattdessen wollen wir einen Satz herleiten, mit dem man sehr einfach verifizieren kann, dass zwei Ausführungsfolgen äquivalent sind. Dazu benötigen wir noch die Begriffe der Werteübergangsrelation und von relevanten Aufträgen.

Definition 9. *Werteübergangsrelation:*
Die Werteübergangsrelation $W\ddot{U}(w)$ einer Ausführungsfolge ist definiert durch:
$W\ddot{U}(w) = \{(a_j, a_i)\}$ *für alle (a_j, a_i) für die gilt, dass es eine Variable v gibt, die l_i von s_j liest.*
Wobei l_i liest v von s_j gilt wenn:
s_j in w vor l_i ist, v in ein_i und aus_j ist, und für keinen anderen Auftrag s_k , der in w zwischen s_j und v_i ist, gilt, dass $v \in aus_k$

Mit l_i liest v von s_j ist also gemeint, dass l_i als Wert in der Variable v genau den Wert liest, der vorher von s_j geschrieben wurde. Die Werteübergangsrelation muss nicht transitiv sein, da ja vorrausgesetzt ist, dass kein anderer Auftrag die Variable zwischen l_i und s_j beschreibt.[3]

Definition 10. *Relevante Aufträge*
Ein Auftrag a_i ist für eine Ausführungsfolge w relevant, wenn gilt, dass $(a_i, a_j) \in W\ddot{U}$ und a_j ein relevanter Auftrag ist. Dabei ist der Ausgabeauftrag immer relevant.
Ein Auftrag ist relevant, wenn er für mindestens eine Ausführungsfolge des AS relevant ist, ansonsten ist er nutzlos.
Ein Auftragssystem ist relevant, wenn alle Aufträge relevant sind.

Diese Definition drückt aus, dass ein Auftrag relevant ist, wenn er einen Einfluss auf den Ausgabeauftrag hat.

[3] Natürlich kann WÜ trotzdem transitiv sein, da es möglicherweise mehrere Variablen gibt, zwischen denen die WÜ ja nicht unterscheidet.

3.4 Satz über Äqivalenz von Ausführungsfolgen

Nun können wir einen Satz über die Äquivalenz von Ausführungsfolgen eines vollständigen vergröberten AS betrachten:

Satz 1 *Äquivalenz von Ausführungsfolgen*
Zwei Ausführungsfolgen w_1 und w_2 sind äquivalent, wenn
a) sie die gleiche Menge an relevanten Aufträgen besitzen
b) für alle relevanten Aufträge und alle Variablen v gilt: a_i liest v von a_j in w_1 genau dann, wenn a_i liest v von a_j in w_2

Dieser Satz ist anschaulich einleuchtend, da das Resultat gleich ist, wenn die Reihenfolge der relevanten Aufträge gleich ist.
Der Beweis gelingt mithilfe von Herbrand-Interpretationen:[4]

Beweis. Die Wertemenge D der Herbrand-Interpretation ist die Menge aller Terme über den verwendeten Funktionen. Die Funktionsanwendung bedeutet die Termerweiterung.

Sei nun $AS = (A, \prec)$ ein vollständiges vergröbertes Auftragssystem. Seien w_1 und w_2 nun zwei Ausführungsfolgen dieses AS, für die a) und b) gilt, so ist zu zeigen, dass diese Folgen äquivalent sind.

Es seien n Funktionen f_i ($i = 0$ bis n) definiert mit $n = |A|$.
Die Funktion $f(a_i, z) \rightarrow z$ ist dann gegeben durch: Wende auf alle Variablen in z, die durch aus_i gegeben sind die Funktion f_i an. Die Folgen sind nun äquivalent, wenn der Endzustand exakt die gleichen Terme enthält.[5]

Zu zeigen ist also, wenn die Folgen a) und b) erfüllen, so haben sie die gleichen Terme im Endzustand.
Die Termbildung erfolgt entlang der "liest von" Beziehung, da in der "liest von" Beziehung ja vorausgesetzt wurde, dass in der Zwischenzeit kein anderer Auftrag den Wert der Variable überschreibt, also kein weiterer Term hinzu kommt. Die Bildung der Terme enthält keine nutzlosen Aufträge, da nach Definition 10 kein relevanter Auftrag von ihnen schreibt.
Wenn nun also a) und b) gelten, so müssen die resultierenden Terme gleich sein, da sie entlang der "Liest von" Beziehungen gebildet werden. □

Der Satz gilt auch für nicht vergröberte AS. Der Beweis ist etwas komplexer, folgt aber der gleichen Idee. [VJ87, S.153]

3.5 Serialisierbare Ausführungsfolgen

Definition 11. *Serielle Ausführungsfolge*
Eine Ausführungsfolge w ist seriell, wenn für alle $w_i \in w$ gilt, dass
$w_i = l_k \Rightarrow w_{i+1} = s_k$

[4] siehe auch [Luc79]
[5] Da die Form der Funktionen f_i nicht festgelegt wurde, gilt dann $w_1 \Leftrightarrow w_2$ für alle Funktionen f_i.

Gemeint ist, dass Lese- und Schreibauftrag ungeteilt (seriell) hintereinander ausgeführt werden.

Definition 12. *Serialisierung einer Ausführungsfolge*
w ist eine Serialisierung von w′ wenn gilt:
$w \Leftrightarrow w'$ *und w ist seriell.*

w′ heißt serialiserbar, wenn eine Serialiserung existiert.
w′ heißt schwach serialiserbar, wenn w′ äquivalent zu einer seriellen Ausführungsfolge eines Unterauftragssystems vom AS ist.

Oft wird bei schwach serialiserbaren Systemen das Unterauftragssystem aller relevanten Aufträge betrachtet, da es möglich ist, dass die nutzlosen Aufträge eine Serialisierung verhindern.

Um zu entscheiden, ob es eine Serialisierung gibt, bildet man einen Serialisierungsgraphen. Falls eine Serialisierung von w_1 existiert, so muss sie eine Ausführungsfolge w_2 des vergröberten AS sein[6]. Nach Satz 1 muss w_2 die gleiche "liest von" Beziehungen wie w_1 haben, dann sind diese folgen äquivalent, und damit ist w_2 eine Serialiserung von w_1. Einen Serialiserungsgraphen konstruiert man, in dem man zuerst die Präzendenzen als Kanten aufnimmt. Danach nimmt man die Werteübergangsrelation von w_1 hinzu. Sollte dabei ein Widerspruch, also ein Zyklus im Graphen, entstehen, ist w_1 nicht serialiserbar. Andernfalls kann es aber noch weitere Konflikte geben:
Angenommen es gibt eine Variable v mit a_i liest (in w_1) v von a_j und a_k beschreibt v ($c \in aus_k$), so gibt es zwei mögliche Fälle: $a_j \prec_{w2} a_i \prec_{w2} a_k$ oder $a_k \prec_{w2} a_j \prec_{w2} a_i$. a_k kann v also vor der Ausführung von a_j und a_i beschreiben oder danach. Man nimmt nun *einen* der beiden Fälle zum Serialiserungsgrapen hinzu. Falls ein Widerspruch (also ein Zyklus im Graphen) entsteht, betrachtet man den anderen Fall. Wenn der Graph keinen Zyklus enthält beschreibt er also eine serielle Ausführungsfolge w_2.
Durch die Fallunterscheidung erhält man also eine Menge von Relationen, falls von denen eine Relation zyklusfrei ist, besitzt w_1 eine Serialiserung.[7]

Definition 13. *Serialisierungsgraphen*
Ein Serialiserungsgraph SG(w) ist definiert durch die Relation $\prec_o$ *über die Auftragsmenge, wobei* $\prec_o$ *definiert durch die transitive Hülle von* $\prec \cup \prec_1 \cup \prec_2$.
Wobei $\prec$ *die Präzendenzrelation ist*
und $\prec_1 := W\ddot{U}(w_1)$ *die Werteübergangsrelation*
und $\prec_2$ *definiert durch: wenn* a_i *liest* v *von* a_j *und* $v \in aus_k$ *gilt, dann ist entweder* $(a_k, a_i) \in \prec_2$ *oder* $(a_i, a_k) \in \prec_2$

[6] Da im vergröberten As die ungeteilte Ausführung impliziert ist.
[7] Diese serielle Ausführungsfolge w_2 kann man bei Bedarf aus dem Graphen ablesen. Da alle "Präzendenzen" des Grapen eingehalten werden müssen bleibt kein weiterer "Spielraum".

Die Menge aller dieser Graphen sei mit $MSG(w)$ bezeichnet.

Satz 2 *Eine Ausführungsfolge w ist genau dann serialiserbar, wenn $MSG(w)$ mindestens einen zyklusfreien Graphen enthält.*

Dieser Satz folgt aus der Konstruktion der Serialiserungsgraphen. [VJ87, S.157]

3.6 NP-Vollständigkeit

Theorem 1. *Sei w eine Ausführungsfolge: das Problem, ob w serialisierbar ist ist NP-vollständig. Dieses Problem sei mit SR bezeichet. vgl. [Pap79]*

Beweis. a) $SR \in NP$:
Man kann ein gegebenes Zertifikat, also eine serielle Ausführungsfolge mit Satz 1 in Polynominialzeit verifizieren. Dafür muss nur entschieden werden, ob die beiden Folgen äquivalent sind, die "liest von" Beziehung auf Gleichheit zu prüfen erfordert nur lineare Zeit.
b) SR ist NP-Schwer:
Aus Satz 2 folgt, dass das Problem äquivalent ist, zu Folgendem: Gegeben sei eine Menge von Graphen (über der gleichen Knotenmenge): gibt es einen azyklischen Graph? (Jeder azyklische Graph ist dann eine serielle Ausführungsfolge)
Es soll nun eine Reduktion von 3SAT auf das Graphenproblem angegeben werden:

Gegeben sei eine Probleminstanz von 3SAT mit n Variablen $x_1, \ldots, x_n$ und m Klauseln $C_1, \ldots, C_m$ mit je 3 Literalen $\lambda_{i1} \vee \lambda_{i2} \vee \lambda_{i3}$.
Man konstruiert daraus eine Menge an Graphen wie folgt:
für jede Variable x_j nimmt man die Knoten a_j, b_j, c_j und für jedes Literal λ_{ik} die Knoten y_{ik}, z_{ik} hinzu.
Für jede Variable x_i nimmt man (a_i, b_i) als Kante mit auf, und *entweder* (b_i, c_i) *oder* (c_i, a_i)[8]. Für jede Klausel werden die Kanten (y_{i1}, z_{i2}) ,(y_{i2}, z_{i3}) und (y_{i3}, z_{i1}).
Wenn $\lambda_{ik} = x_j$ (nicht negiert), dann nehme man die Kanten (c_j, y_{ik}), (b_j, z_{ik}) sowie *entweder* (z_{ik}, y_{ik}) *oder* (y_{ik}, b_j) hinzu.
Wenn $\lambda_{ik} = \neg x_j$ (negiert), dann nehme man die Kanten (z_{ik}, c_j), (y_{ik}, a_j) sowie *entweder* (a_j, z_{ik}) *oder* (z_{ik}, y_{ik}) hinzu.

Diese Probleminstanz ist n^4 mal größer als eine von 3SAT, da man bei der Konstruktion zweimal die Möglichkeit hat, etwas auszuwählen. Daher erfordert die Konstruktion auch nur Polinominialzeit.

Für den Beweis ist noch zu zeigen, dass die Formel F erfüllbar ist, genau dann, wenn es in der konstruierten Menge (mindestens) einen azyklischen Graphen

[8] Bei jedem *entweder/oder* ist wie bei der Konstruktion der Serialiserungsgraphen gemeint, dass die Menge einen Graphen mit der *entweder*-Kante und einen, mit der *oder*-Kante enthält.

gibt.
Angenommen es gibt einen azyklischen Graphen:
Nach Konstruktion ist für jede Variable x_j entweder (b_j, c_j) oder (c_j, a_j) in jedem Graphen[9] vorhanden.
Wenn die Kante (c_j, a_j) vorhanden ist, ist damit gemeint, dass x_j mit *true* belegt ist. Wenn ein Literal den Wert *false* hat, ist dafür die Kante (z_{ik}, y_{ik}) vorhanden. Andernfalls wäre (im Widerspruch zur Annahme) ein Zyklus – entweder (c_j, y_{ik}, b_j) oder (z_{ik}, c_j, a_j).
Die einzige Form einen Zyklus der Form $(z_{i1}, y_{i1}, z_{i2}, y_{i2}, z_{i3}, y_{i3})$ zu vermeiden ist, dass mindestens ein Litral in jeder Klausel wahr ist[10], was bedeutet, dass die Formel erfüllbar ist.

Andersherum: Angenommen F ist erfüllbar, so enthält die Menge einen azyklischen Graphen:
Nach der Konstruktion ist die einzige Möglichkeit einen zyklischen Graphen zu erhalten, ein Zyklus der Form $(z_{i1}, y_{i1}, z_{i2}, y_{i2}, z_{i3}, y_{i3})$.[11] Wie oben bereits Feststellt, existiert ein solcher Zyklus nur dann, wenn die Formel (also mindestens ein Literal) nicht erfüllbar ist.

Aus a) und b) folgt: $SR \in NPC$ □

4 (Spur)Funktionale AS

4.1 Was sind funktionale AS

In der Anwendung möchte man auch bei nebenläufiger Ausführung der Aufträge ein eindeutiges Resultat haben. Dieses Resultat kann dann als Funktion der Eingangswerte angesehen werden. Daher der Begriff funktional.

Definition 14. *Funktionale AS*
Ein schematisches AS heißt funktional, wenn alle Ausführungsfolgen zueinander äguivalent sind.

Nun ist es "egal" welche Ausführungsfolge bei der nebenläufigen Ausführung der Aufträge konkret ausgeführt wird. Da sie ja alle äquivalent sind, liefern sie auch immer das gleiche Resultat.

Definition 15. *Störungsfreie Auftgäge*
Zwei Aufträge a_i und a_j sind störungsfrei, wenn
entweder $a_i \prec a_j$
oder $a_j \prec a_i$
oder $dis(a_i, a_j)$ gilt.

[9] Insbesondere in dem betrachteten Graphen ohne Zyklus.
[10] Dann fehlt einmal die Kante (z_{ik}, y_{ik}), womit der Zyklus unterbrochen ist.
[11] Zur ein solcher Zyklus würde in allen Graphen der Menge auftauchen, gefragt ist ja nur nach einem Graphen ohne Zyklus.

Wobei $dis(a_i, a_j)$ gilt wenn:
$(aus_i \cap aus_j) \cup (ein_i \cap aus_j) \cup (aus_i \cap ein_j) = \emptyset$ *gilt.*

Ein AS ist störungsfrei, wenn alle seine Aufträge paarweise störungsfrei sind.

Zwei Aufträge sind dann störungsfrei, wenn kein Konfliktpotential zwischen ihnen besteht. Also wenn die Abfolge der Aufträge klar durch Präzendenzen geregelt ist, oder die Aufträge nicht in einer "liest von" Beziehung stehen und nicht die gleichen Variablen überschreiben.
Im zweiten Fall ist die Reihenfolge der Ausführung der beiden störungsfeien Aufträge unerheblich, da sie sich ja nicht gegenseitig beeinflussen können.[12]

Definition 16. *Verlustfreie Aufträge*
Ein Auftrag a_i ist verlustfrei, wenn $aus_i \neq \emptyset$

Ein AS ist verlustfrei, wenn alle seine Aufträge verlustfrei sind.

Ein verlustfreier Auftrag hat eine Ausgabe, kann also potentiell relevant sein.
Relevante Aufträge sind nicht verlustfrei.
Beweis:
Der Ausgabeauftrag ist per Definition verlustfrei.[13]
Ein anderer Auftrag kann nur relevant sein, wenn seine Ausgabe von einem anderen relevanten Auftrag gelesen wird. Dafür muss er eine Ausgabe haben, die gelesen werden kann.

Satz 3 *Funktionalität von Auftragssystemen*
Ein AS ist genau dann funktional, wenn alle relevanten Aufträge paarweise störungsfrei sind.

Beweis. Alle relevanten Aufträge sind Störungsfrei $\Rightarrow$ AS ist funktional:
Zu zeigen ist, dass zwei beliebige Ausführungsfolgen w_1 und w_2 äquivalent sind. Nach Satz 1 gilt dies, wenn für alle relevanten Aufträge die "liest von" Beziehungen gleich sind.
Da alle relevanten Aufträge störungsfrei sind, kann a_i liest v von a_j nur gelten, wenn $a_j \prec a_i$ gilt (nach Definition 14). Wenn die Reihenfolge der beiden Aufträge aber bereits durch die Präzendenzen festgelegt ist, so muss diese Reihenfolge auch in w_2 eingehalten werden. Daher sind w_1 und w_2 nach Satz 1 äquivalent.

AS ist funktional $\Rightarrow$ Alle relevanten Aufträge sind störungsfrei:
Seien a_i und a_j beliebige relevante Aufträge. Zu zeigen ist, dass sie störungsfrei sind. Wenn $(a_i, a_j) \notin \prec$ und $(a_j, a_i) \notin \prec$, ist zu zeigen, dass $dis(a_i, a_j)$ gilt.
Wir nehmen an, dass $v \in aus_i \cap ein_j \neq \emptyset$ gilt und führen dies zum Widerspruch:

[12] Natürlich nur dann, wenn durch die Präzendenzen nicht eine bestimmte Reihenfolge vorgegenben ist (was dann der Fall 1 wäre).

[13] Selbst wenn für den Ausgabeauftrag gilt: $aus_{ausgabe} = \emptyset$, so ist er verlustfrei, da seine Ausgabe "außerhalb" des Auftragssystems (z.B. auf einem Drucker) ausgegeben wird, und daher nicht unbedingt durch $aus_{ausgabe}$ modelliert sein muss.

Es gibt serielle Ausführungsfolgen w_1 und w_2 in denen gilt $a_i <_{w_1} a_j$ und $a_j <_{w_2} a_i$[14]. Da das AS funktional ist, müssen w_1 und w_2 äquivalent sein. Also kann nach Satz 1 nicht gelten, dass a_i v von a_j liest (und andersherum). Folglich muss für w_1 ein a_k mit $v \in aus_k$ existeren, für den $a_i <_{w_1} a_k <_{w_1} a_j$ gilt. Da dies für jede Ausführungsfolge $a_i <_w a_j$ gelten muss, folgt $a_i \prec a_k \prec a_j$. Im Widerspruch zur Annahme folgt dann $a_i \prec a_j$.
Der Fall, dass $v \in ein_i \cap aus_j \neq \emptyset$ gilt, ist analog.
Im Fall $v \in aus_i \cap aus_j \neq \emptyset$ wäre a_i in w_1 nicht relevant, womit w_1 nicht äquivalent zu w_2 sein kann. □

4.2 Spurfunktionalität

Definition 17. *Spur einer Variablen*
$spur(w, v, I)$ bezeichnet die Spur der Variablen v in der Ausführungsfolge w bei der Intapretation I.
Diese Spur ist genau diejenige Abfolge von Werten, die durch die Zustandsabfolge von w bezüglich I für die Variable v bestimmt ist.

$spur(w, I)$ ist ein Vektor mit den Einträgen $(spur(w, v_1, I), \ldots, spur(w, v_n, I))$

Die Spur einer Variablen ist einfach die Abfolge der Zustände, die diese Variable annimmt.

Definition 18. *Spuräquivalenz*
Zwei Ausführungsfolgen w_1 und w_2 sind spuräquivalent, wenn $spur(w_1, I) = spur(w_2, I)$ für alle *Interpretationen I gilt.*

Ein AS heißt spurfunktional,
wenn alle Ausführungsfolgen paarweise spuräquivalent sind.

Analog zum Satz 3 gilt:

Satz 4 *Spurfunktionalität von Auftragssystemen*
Ein AS ist genau dann spurfunktional, wenn alle verlustfreien Aufträge paarweise störungsfrei sind.

Da der Beweis analog zu dem von Satz 3 ist, soll an dieser Stelle auf ihn verzichtetet werden.[15]

4.3 Zusammenhänge

Die folgende Graphik soll die Zusammenhänge der Begriffe funktional, spurfunktional, relevant, störungsfrei und verlustfrei klarstellen:

[14] Diese Folgen existieren, da ja keine Präzendenzen die Reihenfolge von a_i und a_j vorgeben.
[15] vgl. [VJ87, S.169]

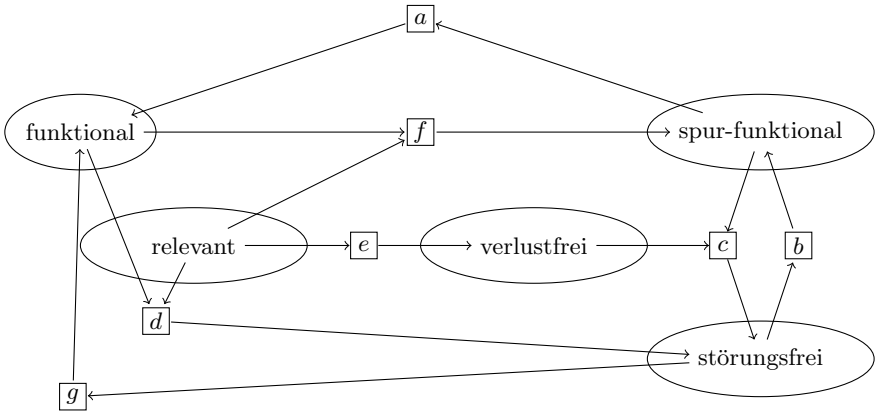

Abb. 6. Beziehungen der Begriffe

Die Beziehung

a) gilt, da spuräquivalente Folgen auch äquivalent sind
b) gilt, da wenn alle Aufträge störungsfrei sind, sind es im Besonderen auch die verlustfreien
c) folgt aus Satz 4
d) folgt aus Satz 3
e) gilt, da relevante Aufträge immer verlustfrei sind
f) folgt aus d) und b)
g) folgt aus a) und b)

5 Commit-Protokolle

In der Praxis kommt es häufig vor, dass ein Auftrag noch in weitere kleine Aufträge unterteilt ist, welche nicht unbedingt auf dem gleichen System ausgeführt werden müssen. Daher braucht man ein Instrument, das den erfolgreichen Abschluss eines Auftrags kennzeichnet, wenn dieser nicht nur auf einem System ausgeführt wird. In Verbindung mit Datenbanken spricht man jedoch nicht von Aufträgen, sondern von Transaktionen, wobei der Inhalt dieser Transaktionen meistens im Verändern der Datenbank besteht. Im Zusammenhang mit Datenbanken haben sich hierfür die Commit-Protokolle bewährt. Diese zeigen nicht nur das Ende einer Transaktion an, sodass man dann die nächste starten kann, sondern gewährleisten auch, dass Fehler innerhalb der Transaktion entdeckt werden, sodass man im Zweifel den Ursprungszustand wiederherstellen kann (z.B. um die fehlgeschlagene Transaktion erneut auszuführen).
Im Folgenden sollen zwei Varianten von Commit-Protokollen beispielhaft vorgestellt werden.

5.1 Zwei-Phasen-Commit

In der ersten Phase des Zwei-Phasen-Commit sendet der Koordinator eine "prepare to commit" Nachricht an alle anderen beteiligten Akteure. Jeder Teilnehmer muss dann dem Koordinator mitteilen, ob seine lokale Transaktion erfolgreich war. Nur wenn *alle* Teilnehmer mit "erfolgreich" stimmen, sendet der Koordinator "global-commit" an alle Teilnehmer, dann dürfen die Teilnehmer die Transaktion als erfolgreich ansehen und die Änderungen in die Datenbank eintragen. Falls aber ein Teilnehmer mit "abort" stimmt, wird die Transaktion abgebrochen und der Koordinator weist alle Teilnehmer an, die Transaktion abzubrechen. Um im Nachhinein (z.B. bei einem Ausfall) den Vorgang rekonstruieren zu können, werden diese Aktivitäten protokolliert.
Dieses Verhalten wird auch durch die Abb. 7 beschrieben.

Da das Protokoll beispielsweise bei einem Ausfall des Koordinators nicht immer eine Korrektheit garantieren kann, wurde das robustere Drei-Phasen-Commit Protokoll entwickelt.

5.2 Drei-Phasen-Commit

Zusätzlich zum Zwei-Phasen-Commit wird noch die Phase des Pre-Commits eingeführt. In dieser Phase wartet der Koordinator auf die Bestätigung aller Teilnehmer, um erst danach "global commit" zu senden.

Dadurch gewährleistet man, dass alle Teilnehmer von dem Erfolg der Transaktion mitbekommen, bevor der eigentliche Commit durchgeführt wird und man die Änderungen ausführt. Im Zwei-Phasen-Commit könnte ein Teilnehmer im Fehlerfall die Commit-Nachricht nicht erhalten und dann vermutlich nach einem Timeout von einem Abbruch ausgehen, sodass ein inkonsistenter Zustand erreicht würde.

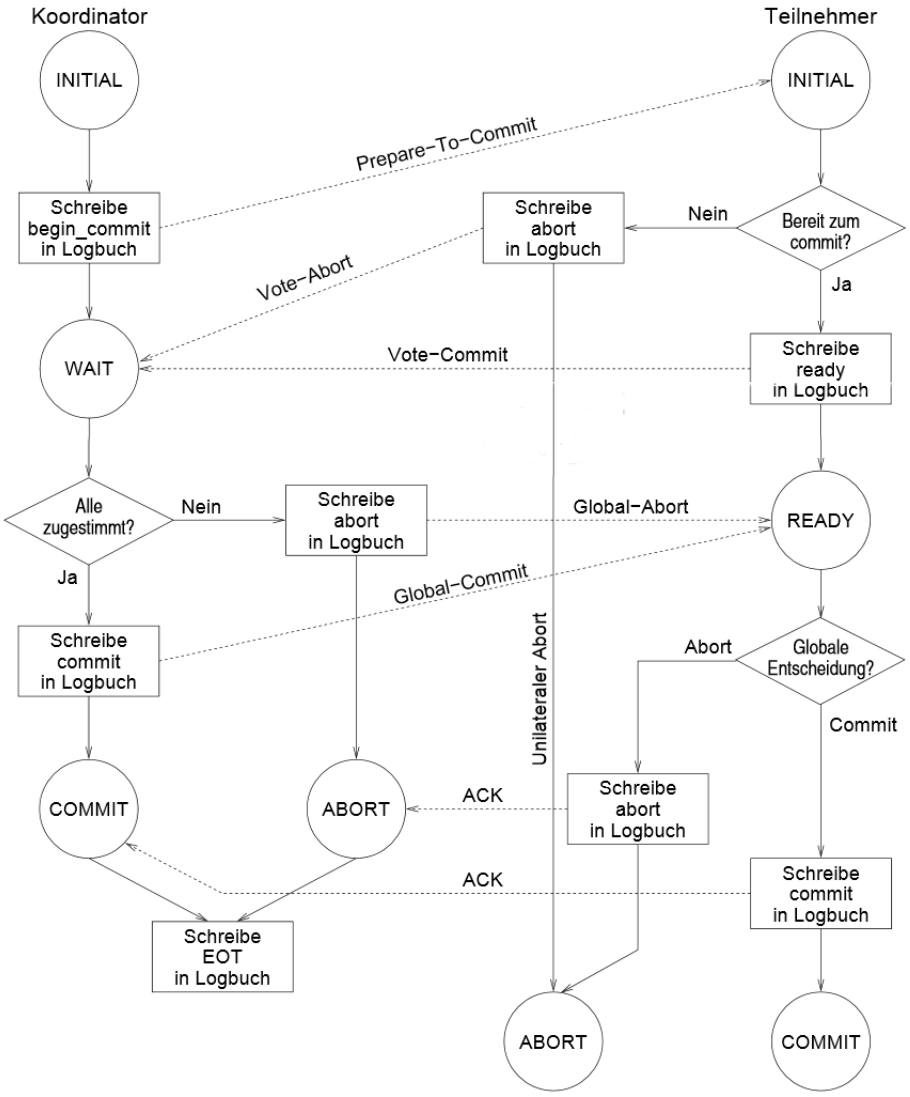

Abb. 7. Darstellung des 2 Phasen Commit aus [Sch06]

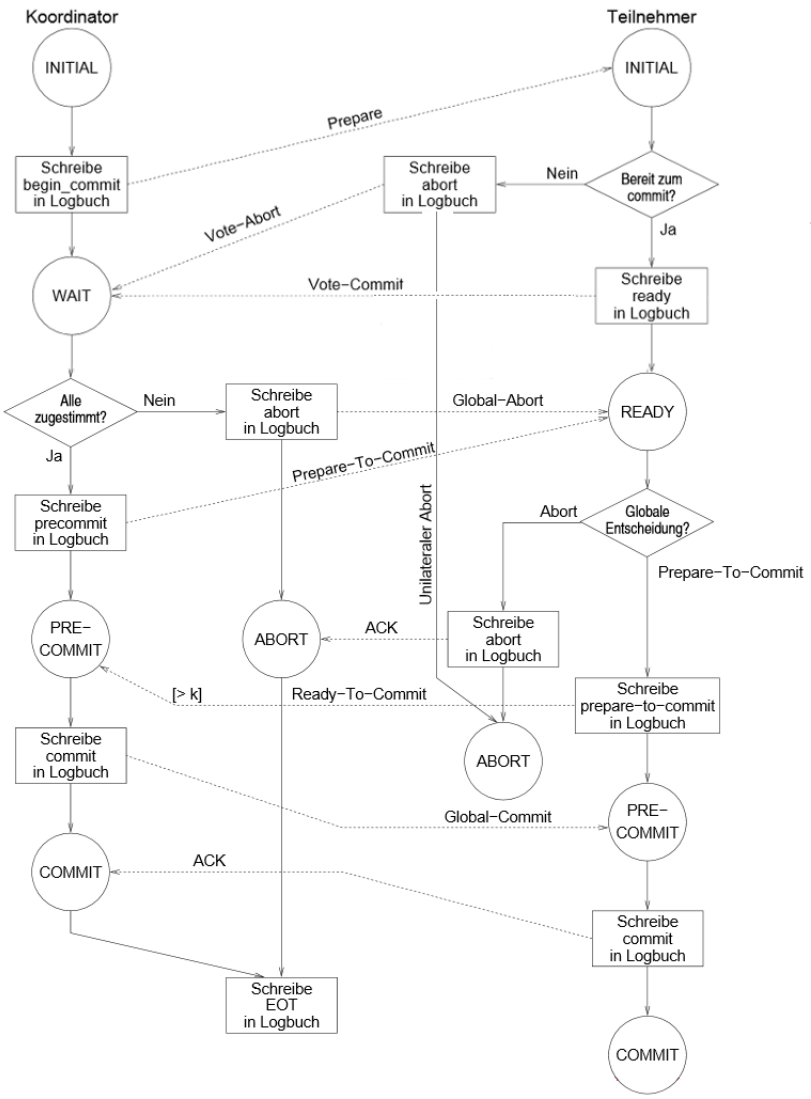

Abb. 8. Darstellung des 3 Phasen Commit aus [Sch06]

6 Fazit/Ausblick

Im ersten Abschnitt haben wir Auftragssysteme und ihre Darstellungen kennengelernt.

Danach behandelten wir im zweiten Abschnitt die Interpretation schematischer AS, damit wir damit Aussagen über die Äquivalenz von Ausführungsfolgen machen konnten. Wichtiges Fazit aus diesem Abschnitt ist daher der Satz 1, nach dem zwei Ausführungsfolgen dann äquivalent sind, wenn die relevanten Aufträge die gleichen "liest von" Beziehungen haben. Zuletzt wurde die Frage, wann ein Auftragssystem serialiserbar ist behandelt und als NP- vollständig nachgewiesen.

Im dritten Abschnitt ging es um die Funktionalität von Auftragssystemen, da man (gerade in der Praxis) immer an einem eindeutigen Ergebnis interessiert ist.

Der letzte Abschnitt stellte die Commit-Protokolle als Werkzeug zum gewährleisten der Konsistenz eines einzelnen Auftrags vor.

Wie im Abschnitt über die Commit-Protokolle bereits angerissen wurde, liegt eine wesentliche Anwendung der Theorie von Auftragssystemen in verteilten Datenbanksystemen.

Literatur

[And13] Thomas Andreae. Mathematik 1 für studierende der informatik und wirtschaftsinformatik. 2013.

[KBH14] Michael Köhler-Bußmeier and Frank Heitmann. Fromale grundlagen der informatik. 2014.

[Luc79] Luckham. On formalised computer program. *J Comp & Syst SCI*, 4:220–249, 1979.

[Pap79] Christos H. Papadimitriou. Serializability of concurrent database updates. 1979.

[Rah94] Erhard Rahm. *Mehrrechner-Datenbanksysteme*. Addison-Wesley-Verlag, 1994.

[Sch06] Eike Schallehn. Verteilte und föderierte datenbanken- kapitel: Verteilte transaktionen, 2006.

[VJ87] Rüdiger Valk and Eike Jessen. *Rechensysteme*. Springer-Verlag, 1987.