Selbstorganisierende Verteilung von zeitlich abhängigen Tasks in verteilten Systemen

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachbereich Informatik und Mathematik
der Johann Wolfgang Goethe-Universität
in Frankfurt am Main

von

Mathias Pacher

aus Karlsruhe

Frankfurt 2010
(D30)

vom Fachbereich Informatik und Mathematik der
Johann Wolfgang Goethe-Universität als Dissertation angenommen.

Dekan:	Prof. Dr. Tobias Weth
Gutachter:	Prof. Dr. Uwe Brinkschulte Prof. Dr. Lars Hedrich
Datum der Disputation:	15. November 2010

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Eingebettete Systeme des Fachbereichs Informatik und Mathematik der Goethe-Universität Frankfurt am Main und am Institut für Prozessrechentechnik, Automation und Robotik der Fakultät für Informatik der Universität Karlsruhe (TH).

Ich möchte Herrn Prof. Dr. Uwe Brinkschulte für die Möglichkeit zur Durchführung dieser Arbeit und für die zahlreichen anregenden Diskussionen und seine Hilfe während der Entstehung dieser Arbeit sehr herzlich danken. Ebenfalls danke ich Herrn Prof. Dr. Lars Hedrich für seine hilfreichen Anmerkungen und seine Bereitschaft, die Zweitgutachterschaft für diese Arbeit zu übernehmen.

Mein Dank gilt auch meinen Kollegen Alexander von Renteln, Michael Bauer, Manuel Nickschas und Linda Stapleton für ihre wertvolle Unterstützung und Hilfe.

Schließlich danke ich meiner Familie und meinen Freunden Aaron, Saskia und Karsten, die mich während der Durchführung dieser Arbeit immer unterstützten und an mich glaubten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Liste der Algorithmen

1. Einleitung

Die Anwendungfelder eingebetteter Systeme sind in den letzten Jahren drastisch angewachsen. Während sie früher hauptsächlich in industriellen Anwendungen vorkamen, haben sie heute ihren Weg ins alltägliche Leben von Menschen gefunden: Sie sind im Alltag in Mobiltelefonen (meist mit Internetanbindung), Personal Digital Assitants (PDAs), Haushaltsgeräten wie Kaffee- und Waschmaschinen und in modernen Autos vertreten. Allein in Deutschland wird das Marktvolumen für eingebettete Systeme im Jahr 2010 bei knapp 19 Milliarden Euro liegen [BIT10b].

Die Gründe für dieses Wachstum sind zum einen die stetige Miniaturisierung der benötigten Hardware-Komponenten, wie Mikroprozessoren und Mikrocontrollern, bei steigender Leistungsfähigkeit dieser Geräte, wodurch diese hochintegrierten eingebetteten Systeme überhaupt erst ermöglicht werden. Zum anderen besteht in den industrialisierten Ländern eine starke Nachfrage nach elektronischen Konsumgütern, die eingebettete Systeme enthalten.

Als Beispiel betrachte man ein Oberklasse-Kraftfahrzeug, welches heute typischerweise mehr als 30 Prozessoren besitzt, siehe [BU07]. Allein ihre Programmierung dauert mehrere Stunden, was unter anderem an der großen Anzahl von Konfigurationen für einen einzigen Fahrzeugtyp liegt (mehr als 1000 Konfigurationen). Als Folge dieser Komplexität ist die Entwicklung und Wartung solcher Systeme hochgradig kompliziert und äußerst fehleranfällig.

Ein anderes Beispiel sind Sensornetze mit mehreren Hundert einfachen Knoten, die koordiniert werden müssen [DFG09b]. Die Koordination, ob zentral oder dezentral, ebenso wie die Inbetriebhaltung des Netzes und der Aktualisierung der eingesetzten Software stellt ein ein schwieriges Problem dar.

In der näheren Zukunft ist abzusehen, dass immer mehr solcher eingebetteten Systeme im täglichen Leben auftreten werden, und dass ihre Programmierung und Handhabung immer komplexer werden wird. Aktuelle Zahlen über Jahresproduktion und Code-Umfang von typischen eingebetteten Systemen kann man der Tabelle in Abbildung 1.1 entnehmen. Weiterhin steigt die Integrationsdichte dieser Systeme stetig wodurch ihre Zuverlässigkeit immer kleiner wird, so dass schon Effekte wie kosmische Strahlung bei der Fehlersuche berücksichtigt werden müssen.

Man erkennt an diesen Beispielen, dass eingebettete Systeme immer komplexer und damit schwerer handhabbar werden. Damit sind sie langfristig nur von Experten bedienbar, was im Widerspruch zu ihrem Einsatz in jedermanns Alltag steht. Zusätzlich besitzen sie aufgrund der schwierigen Herstellungsprozesse unzuverlässige Komponenten und müssen sich an variable Umgebungen anpassen, was ihre

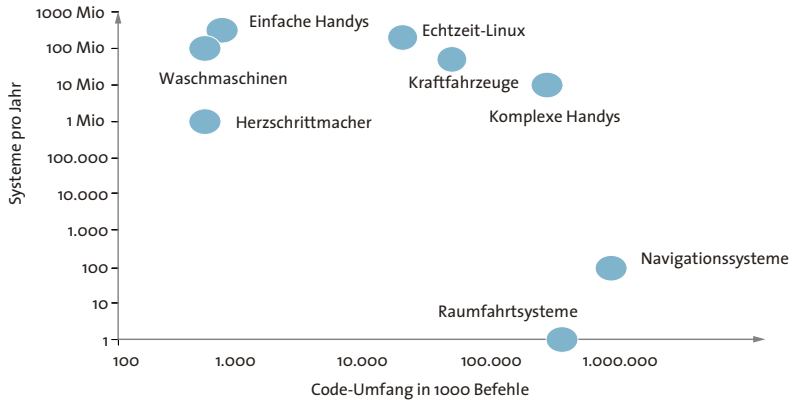


Abbildung 1.1.: Eingebettete Systeme mit Umfang und Jahresproduktion (Stand 2008). Quelle: [BIT10a]

Benutzung weiter erschwert.

Eine Idee, diesen Problemen zu begegnen, bietet das *Organic Computing*, welches in jüngster Zeit intensiv erforscht wird, und was im nächsten Kapitel beschrieben wird.

1.1. Einführung in Organic Computing

Die folgende Beschreibung des Organic Computing lehnt sich an die Beschreibung in [BU07] an. Die Idee des Organic Computing ist technische Systeme mit „lebensähnlichen“ Eigenschaften auszustatten. So soll die Beherrschbarkeit komplexer eingebetteter Systeme sowie die Erhöhung der Robustheit gegenüber Fehlern durch Prinzipien gewährleistet werden, wie sie im Bereich organischer Lebensformen vorzufinden sind. Daher wurde im Jahr 2003 der Forschungsschwerpunkt „Organic Computing“ ins Leben gerufen. Der Begriff geht auf einen Workshop der GI/ITG-Expertenrunde „Zukunftsthemen der Informatik“ zurück [VDE03]. Dabei sollen jedoch keine biologischen Rechensysteme zum Einsatz kommen; die eingebetteten Systeme bestehen nach wie vor aus normalen Rechenknoten wie Mikrocontrollern und Mikroprozessoren. Es sollen vielmehr in der Biologie erfolgreiche Organisationsprinzipien auf Rechnersysteme übertragen werden. Die wichtigsten dieser Prinzipien sind:

Selbstorganisation: Das eingebettete System organisiert sich selbst ohne Einfluss einer äußeren, steuernden Instanz. Alle internen Betriebsabläufe stellen sich eigenständig durch Wechselwirkung der einzelnen Komponenten des Systems untereinander ein.

Selbstkonfiguration: Das eingebettete System findet selbsttätig in Abhängigkeit äußerer Gegebenheiten und Anforderungen (z.B. aktuelle Aufgabenstellung,

Umgebungstemperatur) sowie interner Zustände (z.B. Anzahl und Energievorrat der einzelnen Rechenelemente) eine initiale, arbeitsfähige Konfiguration.

Selbstoptimierung: Ausgehend von der initialen Konfiguration versucht das System ständig, sich selbst zu verbessern und sich an geänderte äußere und innere Bedingungen (z.B. Änderungen in der Aufgabenstellung, der Temperatur, des Energievorrats) anzupassen.

Selbstheilung: Das eingebettete System kann ohne Eingriff eines Administrators einen Großteil auftretender Fehler beheben. Dies beinhaltet sowohl Softwarefehler wie auch permanente oder transiente Ausfälle der Hardware.

Selbstschutz: Angriffe von außen (z.B. eine Denial-of-Service-Attacke) werden selbsttätig erkannt und entsprechende Gegenmaßnahmen (z.B. Abschottung, Rekonfiguration der Kommunikationskanäle) autonom eingeleitet.

Selbstbewusstsein: Das eingebettete System ist sich seiner eigenen Fähigkeiten (z.B. Rechenkapazität, verfügbare Sensorik und Aktorik) sowie seines Kontextes (z.B. gegenwärtiger Aufenthaltsort, Umgebung, Datum, Uhrzeit) bewusst und bezieht diese Informationen in die Ablauforganisation mit ein.

Die aufgezählten Prinzipien werden auch *Selbst-X Eigenschaften* (Self-X) genannt. Durch Anwendung und technische Nutzung dieser Eigenschaften soll die Entwicklung komplexer eingebetteter Systeme erleichtert werden.

1.2. Organic Computing und Middleware

Middleware ist eine Softwareschicht oberhalb der Betriebssystemebene, welche die Entwicklung verteilter Systeme unterstützt [BW05]. Dabei verdeckt die Middleware die verschiedenen Rechen-Ressourcen eines verteilten Systems für den Anwender, der somit nur die Sicht auf eine homogene Plattform hat, auf der Dienste laufen. Auf diesen Diensten kann er Aufträge abarbeiten lassen. Für den Anwender ist es dabei unerheblich auf welcher Ressource ein Dienst läuft: Er übergibt einen Auftrag an die Middlware und spezifiziert den Dienst, auf der er ausgeführt werden soll. Die Übermittlung an die Ressource geschieht dann durch die Middleware. Dies wird durch Abbildung 1.2 dargestellt.

Bei einer klassischen Middleware muss der Anwender für jeden Auftrag angeben, auf welchem Dienst er ausgeführt werden soll und muss im Fehlerfall, falls ein Dienst oder eine Ressource ausfällt, reagieren und den Auftrag neu erteilen. Weil die Middleware jedoch auf allen Ressourcen verteilt ist und Zugriff auf sie hat, ist sie hervorragend für den Einsatz mit Organic Computing geeignet: Sie soll Selbst-X-Eigenschafen erhalten, so dass sie selbstständig in der Lage ist, Aufträge an Dienste zu vergeben, dabei eine Optimierung vorzunehmen und Aufträge bei einem Dienst-

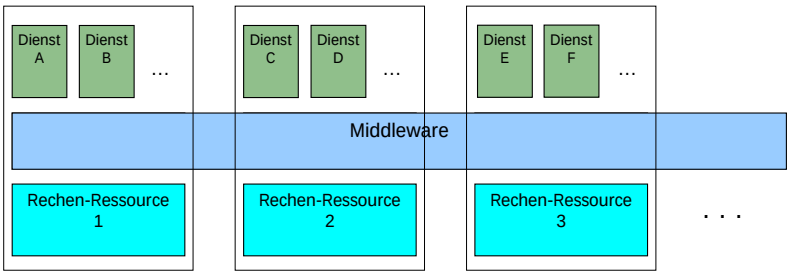


Abbildung 1.2.: Middleware als transparente Softwareschicht in einem verteilten System

oder Ressourcenausfall neu zu vergeben. Die Möglichkeiten des Einsatzes einer solchen organischen Middleware werden in den nächsten Kapitel erörtert.

1.3. Ziele und Konzepte der Arbeit

In dieser Arbeit wird die Verteilung von kausal (zeitlich) abhängigen Tasks in einem verteilten System unter den Gesichtspunkten des Organic Computing untersucht. Sie leistet Beiträge zur Theorie des Schedulings und zur selbstorganisierenden Verteilung solcher abhängiger Tasks unter Echtzeitbedingungen. Dies wird im Folgenden genauer erläutert.

Im Bereich der eingebetteten Systeme und Automatisierung müssen komplexe Aufgaben beziehungsweise Tasks ausgeführt werden. Diese Tasks hängen oft zeitlich voneinander ab, man denke beispielsweise an eine Fertigungsstrasse in der Automobilherstellung, in der nacheinander Montageschritte von Menschen oder Robotern durchgeführt werden.

Die Idee in dieser Arbeit ist, dass die Tasks auf Ressourcen unter Einhaltung der oben erwähnten Selbst-X-Eigenschaften verteilt werden sollen. Als Beispiel kann man sich mobile Roboter vorstellen, die Teile zu einer Fertigungsstrasse liefern. Dabei können Ausfälle von Leitrechnern oder den mobilen Robotern selbst vorkommen, die behoben werden müssen, wenn nicht das ganze System selbst ausfallen soll. Auch können die mobilen Roboter unterschiedliche Fähigkeiten besitzen oder in verschiedenen Zuständen sein (zum Beispiel Batterieladung, maximale Fahrgeschwindigkeit, aktuelle Position), was bei der selbstorganisierenden Verteilung der Tasks berücksichtigt werden muss.

Wie in Kapitel 1.2 beschrieben wurde, ist Middleware hervorragend geeignet, um solch eine Verteilung von Tasks auf Ressourcen vorzunehmen. Da sie jedoch selber auf einer sehr abstrakten Schicht angesiedelt ist, ist es sinnvoll die oftmals sehr komplexen Tasks in Folgen elementarer Aufträge zu zerlegen, die dann durch die organische Middleware auf die Ressourcen verteilt werden können. So kann ein Palettierungsauftrag in die Folge von elementaren Aufträgen *Fahre zur Position 1*, *hole*

dort die Werkstücke, *fahre an die Abladeposition* und *entlade die Werkstücke dort* zerlegt werden. An diesem Palettierungsauftrag wird klar, dass diese Folge von Aufträgen dann von einer einzigen Ressource ausgeführt werden muss, da es beispielsweise keinen Sinn ergibt, wenn eine Ressource zur Position 1 fährt und Werkstücke auflädt, während eine andere zur Abladeposition fährt und versucht die Werkstücke abzuladen. Weiterhin ist auch klar, dass die Aufträge in der beschriebenen Reihenfolge ausgeführt werden müssen, um den Palettierungsauftrag zur erfüllen.

Der Vorteil solch einer Zerlegung ist, dass man viele komplexe Tasks durch Folgen von einfachen Aufträgen realisieren kann und die Middleware keine Informationen über die Semantik der Tasks benötigt.

Daher werden in dieser Arbeit Tasks als Folgen von elementaren Aufträgen (sogenannte *Pfade* betrachtet, siehe Definition 2.2.1), wobei die elementaren Aufträge direkt von den ausführenden Ressourcen bearbeitet werden können. Dabei liegt der Fokus nicht auf der Zerlegung der Tasks, denn dies kann durch einen Benutzer oder durch einen Compiler, der sich an einem modellbasierten Ansatz [KWB03] orientiert, oder durch die Erstellung und Nutzung von Planbibliotheken [RN03] geleistet werden und ist stark von der jeweiligen Anwendung (Domäne) abhängig. Diese Arbeit konzentriert sich vielmehr auf zwei Schwerpunkte:

1. Ausgehend von einer Menge von Pfaden, im Folgenden *System von Pfaden* genannt, werden die zeitlichen Abhängigkeiten auf der Ebene der Aufträge und Pfade formalisiert. Dies ist zwingend erforderlich um Nachweise zu führen, ob das System unter Einhaltung der zeitlichen Abhängigkeiten ausführbar ist und wieviele Ressourcen für seine Ausführung benötigt werden, und ob und wie es möglich ist, diese Anzahl zu reduzieren.

2. Es werden dezentrale Mechanismen, das einstufige und zweistufige künstliche Hormonsystem (KHS), vorgestellt und untersucht, durch die die Pfade eines Systems auf Ressourcen verteilt werden können. Diese Mechanismen berücksichtigen die oben genannten Self-X-Eigenschaften der Selbstorganisation, Selbstkonfiguration und Selbstheilung, und bieten so einen Ansatz, die Komplexität der Verteilung und Bearbeitung von Tasks in einem verteilten System zu vereinfachen.

Die Entwicklung der beiden Schwerpunkte zeigt, dass das Konzept der Pfade und der dezentralen Mechanismen zu deren Verteilung hervorragend für den Einsatz in Middleware geeignet ist. Weiterhin wird für jeden Algorithmus in dieser Arbeit eine obere Zeitschranke für seine Ausführung berechnet, was zeigt, dass die vorgestellten Konzepte und Ideen auch in Echtzeitsystemen mit harten Zeitschranken eingesetzt werden können.

1.4. Aufbau der Arbeit

Diese Arbeit ist wie folgt strukturiert: Kapitel 1 motiviert diese Arbeit vor dem Hintergrund des Organic Computing. In Kapitel 2 wird das Konzept der Pfade als eine Folge von Aufträgen eingeführt, die ununterbrechbar auf einer Ressource ausgeführt werden sollen. Zwischen den Aufträgen der Pfade können kausale Abhängigkeiten existieren, die bei einer Ausführung berücksichtigt werden müssen. Es werden Techniken entwickelt, mit denen ein gegebenes System von Pfaden auf seine Ausführbarkeit hin untersucht werden kann. In Kapitel 3 wird untersucht, wie man einen Ausführungsplan für ein System von Pfaden berechnen kann. Weitergehend werden Untersuchungen angestellt, die es ermöglichen, die Anzahl der für die Ausführung der Pfade benötigten Ressourcen zu reduzieren. In Kapitel 4 werden Rekursionen zur Berechnung der Ausführungszeit eines Systems von Pfaden angegeben und anaylsiert. Kapitel 5 beschreibt zwei Ansätze, wie Pfade auf Ressourcen verteilt werden können, die die oben genannten Eigenschaften des Organic Computing respektieren und zusätzlich für den Einsatz in Echtzeitsystemen geeignet sind. Diese beiden Ansätze werden in Kapitel 6 getestet und evaluiert. In Kapitel 7 wird auf den aktuellen Stand der Technik im Forschungsfeld dieser Arbeit eingegangen, und Kapitel 8 fasst die Arbeit zusammen. Im Anhang A werden wichtige Details beschrieben, die über die Inhalte der Arbeit hinausgehen.

2. Konzeption und theoretisches Modell

Im letzten Kapitel wurden die Einsatzmöglichkeiten von Organic Computing diskutiert. Darauf aufbauend wird nun eine spezielle Architektur vorgestellt und analysiert, mittels derer wesentliche Eigenschaften des Organic Computing im Bereich der Middleware realisiert werden können.

Dieses Kapitel ist wie folgt aufgebaut: In Unterkapitel 2.1 werden die zeitlichen (kausalen) Abhängigkeiten zwischen den Aufträgen nochmals genauer motiviert. In Unterkapitel 2.2 werden die zeitlichen Abhängigkeiten zwischen Aufträgen formal gefasst und in Unterkapitel 2.3 wird darauf aufbauend ein Ausführbarkeitsbegriff für ein System aus Pfaden definiert und ein Ausführbarkeitskriterium dafür entwickelt. Weiterhin wird der Begriff der zeitlichen Abhängigkeiten zwischen Aufträgen auf Pfade (Pfad-Relation) übertragen und es ergibt sich die Frage, wie man einen Ausführungsplan für ein System von Pfaden gewinnt. In Unterkapitel 3.1 wird diese Frage beantwortet, wenn die Pfad-Relation irreflexiv auf den Pfaden ist. In Unterkapitel 3.2 wird diese Frage beantwortet für den Fall, dass die Pfad-Relation nicht irreflexiv auf den Pfaden ist. Es müssen dafür kompliziertere Techniken zur Beantwortung entwickelt werden als im irreflexiven Fall. In Unterkapitel 3.3 wird die Vorgehensweise zur Gewinnung eines Ausführungsplanes zusammengefasst. Die benötigte Zeit für die Ausführungsplanung wird in Unterkapitel 3.4 berechnet und durch Beispiele verdeutlicht. In Unterkapitel 3.5 werden Untersuchungen zur Laufzeiteinsparung vorgenommen, die man durch den Einsatz der in Unterkapitel 3.2 entwickelten Techniken erhält. Das Kapitel wird mit Unterkapitel 3.6 abgeschlossen, in dem über den Fokus dieser Arbeit hinaus die Verwendbarkeit eines existierenden Ausführungsplanes untersucht wird, wenn Pfade und zeitliche Abhängigkeiten aus einem Pfadsystem entfernt werden oder zu einem Pfadystem hinzugefügt werden.

2.1. Motivation

Im Bereich der Automatisierung treten komplexe Tasks auf. Wenn man sich beispielsweise einen Palettierungsauftrag für einen mobilen Roboter anschaut, dann bestehen dieser mindestens aus den folgenden Einzelbefehlen (Aufträgen):

Beispiel 2.1.1.

1. *Fahren(Position1)*
2. *Greifen(Werkstücke)*
3. *Fahren(Abladeposition)*
4. *Abladen(Werkstücke)*

Durch den ersten Auftrag wird der mobile Roboter zu einer definierten Position geschickt, an der sie die zu transportierenden Werkstücke abholen kann. Dies wird mit dem zweiten Auftrag durch das Greifen der Werkstücke erledigt. Durch den dritten Auftrag werden nun die Werkstücke zur Zielposition transportiert und vermöge des vierten Auftrages dort abgeladen. An der Darstellung der Aufträge sieht man nochmals deutlich, dass die selbstorganisierende Middleware sie erstens in der beschriebenen Reihenfolge und zweitens auf einem einzigen mobilen Roboter ausgeführt werden müssen, um den Palettierungsauftrag zu erfüllen.

Wenn ein Anwender also solch komplexe Tasks erstellt, dann resultieren daraus im Allgemeinen eine Folge von *elementaren Aufträgen*, deren sequentielle Abarbeitung auf einer Ressource die Task realisieren. Dabei ist klar, dass die elementaren Aufträge auf der bearbeitenden Instanz, der *Ressource*, wieder in Teilbefehle zerfallen können: Allerdings kapseln sie eine elementare Funktionalität, zum Beispiel Greifen oder Fahren, und die Zusammenhänge zwischen Aufträgen wie Reihenfolge der Ausführung und Nutzung exklusiver Ressourcen können durch sie vollständig beschrieben werden, was für den Kontext dieser Arbeit wesentlich ist.

Weiterhin ist klar, dass ein (technisches) System, welches komplexe Tasks bearbeiten kann, auch in der Lage sein sollte gleichzeitig mehrere davon auszuführen. Als Beispiel kann man sich die Fertigungsstrasse zusammen mit den mobilen Robotern vorstellen. Darin können gleichzeitig Palettierungsaufgaben und Schweißarbeiten zum Verbinden der angelieferten Werkstücke ausgeführt werden. In den folgenden Unterkapiteln werden nun die zeitlichen Abhängigkeiten zwischen Aufträgen formal gefasst und es werden Bedingungen angegeben, unter denen die Aufträge ausführbar sind. Des Weiteren werden Optimierungen entwickelt, durch die die Anzahl der zur Ausführung von komplexen Befehlen benötigten Ressourcen verkleinert werden kann.

2.2. Die kausale Relation auf Aufträgen

Man betrachte noch einmal das Beispiel 2.1.1 aus dem letzten Unterkapitel:

1. Fahren(Position1)

2. Greifen(Werkstücke)
3. Fahren(Abladeposition)
4. Abladen(Werkstücke)

Die wesentliche Eigenschaft, die diese Folge von Aufträgen kennzeichnet, ist, dass diese *nacheinander* und von *einer Person oder Ressource* ausgeführt werden müssen. Die vier Aufträge hier müssen nacheinander von einem Roboter ausgeführt werden. Denn es wäre sinnlos, wenn ein Roboter, der nicht die ersten beiden Aufträge ausgeführt hat, den dritten Auftrag ausführt, also zur Abladeposition fährt (denn er hat ja nichts zum Abladen). Dieses Beispiel motiviert die folgende Definition:

Definition 2.2.1 (Pfad)**.** *Ein Pfad ist eine endliche Menge von Aufträgen, die nacheinander in einer festen Reihenfolge auf einer Ressource ausgeführt werden müssen. Ein Pfad muss atomar auf einer Ressource ausgeführt werden.*

Bemerkung 2.2.1. Wegen dem oben Gesagten wird in dieser Arbeit davon ausgegangen, dass sich jede Task als Pfad darstellen lässt. Im Titel dieser Arbeit wird trotzdem der Begriff Task zum einfacheren Verständnis für einen Leser, der nicht mit der Materie dieser Arbeit vertraut ist, verwendet.

Aufbauend auf Definition 2.2.1 wird die kausale Relation zwischen Aufträgen nun wie folgt definiert (analog zu [Lam78]):

Definition 2.2.2 (Kausale Relation)**.** *Die Relation $<_{\mathfrak{A}}$ auf der Menge der Aufträge von Pfaden eines Systems $\mathcal{S}$ ist die kleinste Relation für die gilt:*

- *Wenn ein Auftrag x vor einem Auftrag y auf einem Pfad liegt, dann gilt $x <_{\mathfrak{A}} y$.*
- *Wenn x und y Aufträge auf zwei verschiedenen Pfaden sind, und x vor y ausgeführt werden muss, dann gilt $x <_{\mathfrak{A}} y$.*
- *Wenn für drei Aufträge x, y und z gilt $x <_{\mathfrak{A}} y$ und $y <_{\mathfrak{A}} z$, dann gilt auch $x <_{\mathfrak{A}} z$.*

Bemerkung 2.2.2. Durch die Forderung nach der kleinsten Relation wird sofort deren *Eindeutigkeit* gewährleistet. Wenn man je zwei in Relation stehende Aufträge x und y als Tupel (x, y) auffasst, so besagt Definition 2.2.2, dass durch $<_{\mathfrak{A}}$ die Menge der Tupel mit der kleinsten Kardinalität beschrieben wird, die alle Forderungen der Definition erfüllt. Die Eindeutigkeit dieser Menge ist klar: Es existieren eine oder mehrere kleinste Mengen, die die Forderungen der Definition erfüllen (man kann zumindest eine per Hand konstruieren, siehe Satz 2.3.1). Wenn nun mehrere verschiedene solcher Mengen mit gleicher Kardinalität existieren würden, so betrachte man zwei davon und bezeichne sie mit M_1 und M_2. M_1 enthält nun ein Tupel von

Aufträgen (x_1, y_1), welches nicht in M_2 enthalten ist, und umgekehrt. Da (x_1, y_1) aber gemäß Definition 2.2.2 entstanden ist, muss es auch in M_2 liegen. Dasselbe gilt auch umgekehrt. Dies zeigt nun die Eindeutigkeit der Relation $<_{\mathfrak{A}}$.

Definition 2.2.3. *Von nun an werde die Menge der Aufträge, die in den Pfaden eines Systems $\mathcal{S}$ liegen, mit $\mathcal{S}_{\mathfrak{A}}$ bezeichnet.*

Bemerkung 2.2.3. Es kann natürlich auch Aufträge x und y aus $\mathcal{S}_{\mathfrak{A}}$ geben, die überhaupt nicht zueinander in Relation stehen. Das bedeutet, dass weder $x <_{\mathfrak{A}} y$ noch $y <_{\mathfrak{A}} x$ gilt. Als Notation für diesen Fall gelte $x||_{\mathfrak{A}} y$.

Beispiel 2.2.1 (Unabhängigkeit von Aufträgen)**.** *Man betrachte die beiden Aufträge „Fahre(Position1)“ im Rahmen eines Palettierungsauftrages und „Sende(Aktuelle _Zeit)“ um einen neu ins System gekommenen Roboter zu initialisieren. Diese beiden Aufträge sind unabhängig und können daher in einer beliebigen Reihenfolge ausgeführt werden.*

Wenn man die strukturellen Eigenschaften der kausalen Relation $<_{\mathfrak{A}}$ auf $\mathcal{S}_{\mathfrak{A}}$ untersucht, ist es sinnvoll anzunehmen, dass die Relation *irreflexiv* ist, das heißt es gilt nicht $x <_{\mathfrak{A}} x$ für einen beliebigen Auftrag x. Das ist auch anschaulich klar, denn ein Auftrag wird zu einem festen Zeitpunkt ausgeführt und kann daher nicht zu sich selber in Relation stehen. Im nächsten Unterkapitel wird deutlich, dass die Irreflexivität von wesentlicher Bedeutung ist für die Formulierung von Ausführbarkeitsbedingungen.

Die Relation $<_{\mathfrak{A}}$ ist nach Definition *transitiv* auf $\mathcal{S}_{\mathfrak{A}}$. Diese Eigenschaft folgt bereits aus den ersten beiden Forderungen der Definition der kausalen Relation, wird aber durch die dritte Forderung explizit erwähnt. Die Transitivität entspricht auch der Anschauung, die man im täglichen Leben hat:

Beispiel 2.2.2 (Transitivität von $<_{\mathfrak{A}}$)**.** *Man betrachte wieder Beispiel 2.1.1. Offensichtlich gilt Fahre(Position1)$<_{\mathfrak{A}}$ Greife(Werkstücke), da es keinen Sinn macht, nach den Werkstücken zu greifen, solange der Roboter noch nicht in Position ist. Außerdem gilt Greife(Werkstücke)$<_{\mathfrak{A}}$ Fahre(Abladeposition), da die Werkstücke an der Abladeposition deponiert werden sollen. Wenn diese Bedingungen eingehalten werden sollen, dann gilt aber offensichtlich auch Fahre(Position1)$<_{\mathfrak{A}}$ Fahre(Abladeposition).*

Bemerkung 2.2.4. Es ist interessant an dieser Stelle den Ausführungszeitpunkt eines Auftrages ins Spiel zu bringen.[1] Es ist klar, dass für zwei Aufträge x und y gilt:

$$x <_{\mathfrak{A}} y \quad \Longrightarrow \quad \text{Ausführungszeitpunkt}(x) < \text{Ausführungszeitpunkt}(y)$$

[1] Damit ist der relative oder absolute Zeitpunkt gemeint, an dem ein Auftrag ausgeführt wird.

Wenn also x vor y ausgeführt werden muss, dann ist auch der Ausführungszeitpunkt von x früher als der von y. Die Umkehrung gilt dagegen nicht: Wenn Ausführungszeitpunkt(x)<Ausführungszeitpunkt(y), dann muss nicht zwangsläufig gelten $x <_{\mathfrak{A}} y$. Der Grund liegt darin, dass $x||_{\mathfrak{A}} y$ gelten kann und x durch das Schedule vor y ausgeführt wurde.

Definition 2.2.4. *Das Tupel* ($\mathcal{S}$, $\mathcal{S}_{\mathfrak{A}}$, $<_{\mathfrak{A}}$) *bezeichne eine System* $\mathcal{S}$ *von Pfaden,* $\mathcal{S}_{\mathfrak{A}}$ *die darin vorkommende Menge von Aufträgen und* $<_{\mathfrak{A}}$ *eine auf* $\mathcal{S}_{\mathfrak{A}}$ *definierte Relation gemäß Definition 2.2.2.*

Es werde vereinbart, dass im Folgenden das Tupel ($\mathcal{S}$, $\mathcal{S}_{\mathfrak{A}}$, $<_{\mathfrak{A}}$) gemeint ist, wenn von einem System $\mathcal{S}$ gesprochen wird. Das Tupel wird nur noch explizit angegeben, wenn es für den Kontext benötigt wird.

Die Eigenschaften der kausalen Relation $<_{\mathfrak{A}}$ sind keine speziellen Eigenschaften der Aufträge der Middleware. Sie wurden bereits in ähnlicher Form von Leslie Lamport beschrieben und im Kontext auf sogenannte logische Uhren untersucht [Lam78]. Auch Friedemann Mattern nutzt diese Definition in [Mat89, CBMT96].

2.3. Ausführbarkeit eines Systems von Pfaden

Gegeben sei ein (von einem Anwender erstelltes) System $\mathcal{S}$ von Pfaden zusammen mit den kausalen Relationen zwischen den beinhalteten Aufträgen. Die Pfade sollen von einer Anzahl von Ressourcen bearbeitet werden. Hierbei stellen sich die folgenden Fragen:

1. **Ist das System $\mathcal{S}$ von Pfaden ausführbar?** Das heißt, gibt es eine Anordnung der Pfade, die ohne eine Verklemmung (Deadlock) bearbeitet werden kann? Satz 2.3.2 wird eine Antwort auf diese Frage geben.

2. **Wieviele Ressourcen werden mindestens benötigt, um die durch $\mathcal{S}$ gegebenen Pfade zu bearbeiten?** Unter realen Bedingungen stehen eine beschränkte Anzahl von Ressourcen für die Bearbeitung der Pfade zur Verfügung. Daher benötigt man einen Algorithmus, durch den man feststellen kann, wieviele Ressourcen für die Bearbeitung mindestens benötigt werden. Wenn sich dann herausstellt, dass mehr Ressourcen als unbedingt benötigt zur Verfügung stehen, kann man diese für Optimierungszwecke oder als redundante Ressourcen nutzen.

Um die erste Frage zu beantworten muss man sich klar machen, wie $\mathcal{S}$ durch einen Anwender definiert wird: Er wird dafür in der Regel die zugehörigen Pfade definieren, siehe Beispiel 2.3.1.

Beispiel 2.3.1.

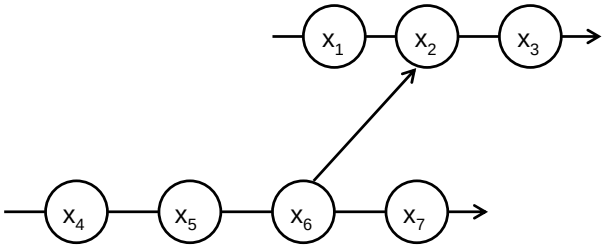


Abbildung 2.1.: Pfade mit Abhängigkeiten

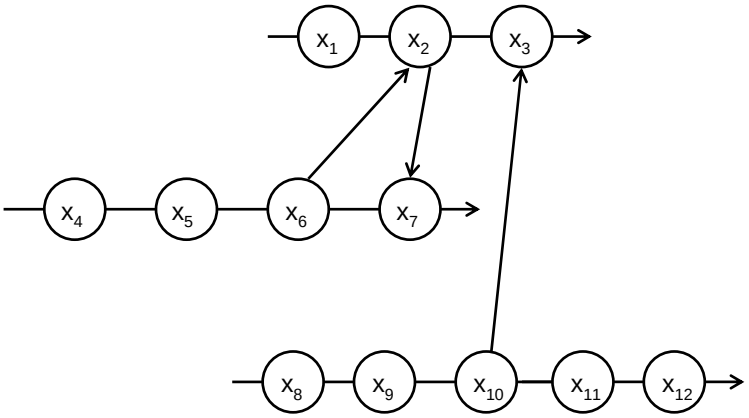


Abbildung 2.2.: Pfade mit mehreren Abhängigkeiten

Die „Hintereinander"-Abhängigkeiten in den Diagrammen, dargestellt durch die Pfeile zwischen den Pfaden, entsprechen aber *nicht* den durch $<_{\mathfrak{A}}$ geforderten Relationen, da die die Transitivität nicht dargestellt wird: In Abbildung 2.1 sieht man nur, dass etwa der Auftrag x_2 auf den Auftrag x_6 wartet. Damit wartet x_2 aber auch auf die Aufträge x_4 und x_5, was der Übersichtlichkeit halber nicht dargestellt wird.

Um die Relation $<_{\mathfrak{A}}$ auf den Aufträgen zu gewinnen werden die paarweisen „Hintereinander"-Abhängigkeiten in einer Adjazenzmatrix beschrieben. Alle Aufträge werden sowohl in der ersten Zeile, als auch in der ersten Spalte aufgezählt, und es gelte folgende Vereinbarung für die Einträge a_{ij} der Adjazenzmatrix:

$$a_{ij} = \begin{cases} 1, & \text{falls Auftrag } x_j \text{ direkt nach Auftrag } x_i \text{ ausgeführt werden muss} \\ 0, & \text{sonst.} \end{cases}$$

Beispiel 2.3.2. *Die Adjazenzmatrix zu Beispiel 2.3.1 erstes Pfadpaar sieht dann wie folgt aus:*

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_1	*0*	*1*	*0*	*0*	*0*	*0*	*0*
x_2	*0*	*0*	*1*	*0*	*0*	*0*	*0*
x_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*
x_4	*0*	*0*	*0*	*0*	*1*	*0*	*0*
x_5	*0*	*0*	*0*	*0*	*0*	*1*	*0*
x_6	*0*	*1*	*0*	*0*	*0*	*0*	*1*
x_7	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Nun stellt sich die Frage, in welchem Zusammenhang die hier eingeführte „Hintereinander“-Relation und $<_{\mathfrak{A}}$ stehen. Sei mit $<_H$ die Relation gekennzeichnet, die man durch Bilden der transitiven Hülle[2] der „Hintereinander“-Relation erhält.

Satz 2.3.1. *Für zwei Aufträge x und y aus $\mathcal{S}_{\mathfrak{A}}$ gilt: $x <_{\mathfrak{A}} y$ genau dann wenn $x <_H y$.*

Beweis.
„$\Rightarrow$“: Gelte also $x <_{\mathfrak{A}} y$ für zwei Aufträge x und y. Dann sind entweder x und y Aufträge desselben Pfades, und y muss nach x ausgeführt werden. Folglich gilt $x <_H y$, unter Beachtung, dass $<_H$ durch Bildung der transitiven Hülle der „Hintereinander“-Relation entstanden ist. Oder x und y sind Aufträge auf verschiedenen Pfaden. Dann ist diese Relation durch Anwendung des zweiten oder dritten Punktes von Definition 2.2.2 entstanden und liegt daher in der transitiven Hülle der „Hintereinander“-Relation, also $x <_H y$.
„$\Leftarrow$“: Analog zu „$\Rightarrow$“. □

Satz 2.3.1 sagt also aus, dass man Pfade wie in Beispiel 2.3.1 beschreiben kann; durch Bilden der transitiven Hülle der zugehörigen Adjazenzmatrix erhält man dann $<_{\mathfrak{A}}$. Daher wird im Folgenden nur noch auf $<_{\mathfrak{A}}$ Bezug genommen. Die Adjazenzmatrix von $<_{\mathfrak{A}}$ wird dann durch die Berechnung der zugehörigen Adjazenzmatrix der $<_H$-Relation gewonnen.

Beispiel 2.3.3 (Gewinnung der Relation $<_{\mathfrak{A}}$)**.** *In der folgenden Tabelle wird die zur Relation $<_{\mathfrak{A}}$ gehörige Adjazenzmatrix bezüglich Beispiel 2.3.2 dargestellt, die durch*

[2]Zur Definition der transitiven Hülle einer Relation siehe Anhang A.1

Berechnung der transitiven Hülle der „Hintereinander“-Relation gewonnen wird.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_1	*0*	*1*	*1*	*0*	*0*	*0*	*0*
x_2	*0*	*0*	*1*	*0*	*0*	*0*	*0*
x_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*
x_4	*0*	*1*	*1*	*0*	*1*	*1*	*1*
x_5	*0*	*1*	*1*	*0*	*0*	*1*	*1*
x_6	*0*	*1*	*1*	*0*	*0*	*0*	*1*
x_7	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Die folgenden Definitionen charakterisieren die Bereitschaft eines Auftrages zur Ausführung beziehungsweise das Ausführen eines Auftrages selber.

Definition 2.3.1 (Ausführbarkeit eines Auftrages)**.**
Gegeben sei das Tupel $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}})$. Ein Auftrag $x \in \mathcal{S}_{\mathfrak{A}}$ eines Pfades in $\mathcal{S}$ ist ausführbar genau dann wenn es entweder keine Aufträge $y \in \mathcal{S}_{\mathfrak{A}}$ von Pfaden in $\mathcal{S}$ gibt mit $y <_{\mathfrak{A}} x$ oder wenn für alle diese Aufträge y mit $y <_{\mathfrak{A}} x$ gilt: y ist ausgeführt.

Definition 2.3.2. *Gegeben sei das Tupel $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}})$. Ein Auftrag $x \in \mathcal{S}_{\mathfrak{A}}$ eines Pfades in $\mathcal{S}$ ist ausgeführt genau dann wenn x ausführbar ist und von einer Ressource bearbeitet wurde.*

Definition 2.3.3 (Ausführbarkeit eines Systems von Pfaden)**.**
Das Tupel $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}})$ ist ausführbar genau dann wenn für jeden der Aufträge $x \in \mathcal{S}_{\mathfrak{A}}$ ein endlicher Zeitpunkt existiert zu dem x ausführbar ist.

Es werde vereinbart, dass im Folgenden gemeint ist, dass das Tupel $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}})$ ausführbar ist, wenn von einem ausführbaren System $\mathcal{S}$ gesprochen wird. Das Tupel wird nur noch explizit angegeben, wenn es für den Kontext benötigt wird.

Auf Basis der bis jetzt gewonnenen Erkenntnisse kann man nun ein Ausführbarkeitskriterium formulieren.

Satz 2.3.2. *Ein System $\mathcal{S}$ von Pfaden ist ausführbar genau dann wenn die Relation $<_{\mathfrak{A}}$ auf $\mathcal{S}_{\mathfrak{A}}$ irreflexiv ist.*

Beweis.
„$\Rightarrow$“: Kontraposition: Die Relation $<_{\mathfrak{A}}$ auf $\mathcal{S}_{\mathfrak{A}}$ ist nicht irreflexiv. Das bedeutet, dass es mindestens einen Auftrag gibt, der zu sich selbst in Relation steht. Daher kann es für ihn keinen endlichen Zeitpunkt geben zu dem er ausführbar ist.
„$\Leftarrow$“: Wieder Kontraposition: $\mathcal{S}$ ist nicht ausführbar. Das kann nur daran liegen, dass es auf einem Pfad einen (in der $<_{\mathfrak{A}}$-Relation ersten) wartenden Auftrag gibt, der nicht ausführbar ist. Dann muss es aber mindestens einen Auftrag geben, auf

den gewartet wird und der nicht ausführbar ist und auf einem anderen Pfad liegt. Dies kann aber nur der Fall sein, wenn dieser Auftrag der erste des neuen Pfades ist, der nicht ausführbar ist oder es davor ein einen wartenden Auftrag gibt, der nicht ausführbar ist. Da es nur endlich viele Pfade gibt, wird schließlich nach maximal n Schritten (n gleich Anzahl der Pfade) der Kreis (rückwärts) geschlossen und es existiert ein Zyklus. Folglich ist die Relation $<_{\mathfrak{A}}$ nicht irreflexiv auf $\mathcal{S}_{\mathfrak{A}}$. □

Die Aussage des Satzes lässt sich nun dazu benutzen, einen Test auf Ausführbarkeit einer Menge von Pfaden durchzuführen: Es gilt nämlich offensichtlich, dass ein System $\mathcal{S}$ von Pfaden (zusammen mit $<_{\mathfrak{A}}$ zwischen ihren Aufträgen) ausführbar ist, wenn die Adjazenzmatrix der Relation $<_{\mathfrak{A}}$ keine Einträge in der Hauptdiagonalen enthält. Diese Adjazenzmatrix lässt sich einfach durch durch das Bilden der transitiven Hülle der „Hintereinander"-Relation gewinnen. Dazu kann man beispielsweise den Algorithmus von Warshall nutzen [War62]. Der Aufwand dafür liegt maximal bei k^2 Vergleichen und k^3 arithmetischen Operationen, wobei k der Anzahl aller Aufträge im System entspricht. Das bedeutet, dass man die Hauptdiagonale der Adjazenzmatrix durchsuchen muss, und wenn sie nur Nullen enthält, dann ist $<_{\mathfrak{A}}$ irreflexiv. Der Aufwand für diese Suche beträgt maximal k Vergleichsoperationen.

Zur Bildung der transitiven Hülle und der Prüfung der Irreflexivität von $<_{\mathfrak{A}}$ wird unter Benutzung dieser Algorithmen also folgender Aufwand benötigt:

- $k^2 + k$ Vergleiche
- k^3 arithmetische Operationen

Man kann den Satz 2.3.2 auch wie folgt formulieren: Ein System von Pfaden ist *nicht* ausführbar genau dann wenn die Relation $<_{\mathfrak{A}}$ auf $\mathcal{S}_{\mathfrak{A}}$ nicht irreflexiv ist.

Was folgt aber daraus, wenn ein System von Pfaden nicht ausführbar ist? Dazu betrachte man Definition 2.2.2: Es ist sinnvoll, davon auszugehen, dass ein Benutzer einen Auftrag innerhalb eines Pfades a priori so konzipiert, dass er nicht zu sich in Relation steht bezüglich $<_{\mathfrak{A}}$. Wenn nun ein System von Pfaden nicht ausführbar ist, folgt dann aber aus eben getroffener Annahme, dass es mindestens zwei Aufträge in verschiedenen Pfaden gibt, die durch die Relation $<_{\mathfrak{A}}$ in gegenseitiger Beziehung stehen und so mittels der Transitivität ein Zyklus definiert ist. Diese Tatsache wird im Weiteren angenommen und benutzt.

Beispiel 2.3.4. *Die Adjazenzmatrix von $<_{\mathfrak{A}}$ auf $\mathcal{S}_{\mathfrak{A}}$ zu Beispiel 2.3.1 oberes Pfadpaar wird in Beispiel 2.3.3 berechnet. Da die Hauptdiagonale nur mit Nullen besetzt ist, sieht man, dass $<_{\mathfrak{A}}$ auf $\mathcal{S}_{\mathfrak{A}}$ irreflexiv ist. Somit können die Pfade nach Satz 2.3.2 ausgeführt werden. Wenn man zwei Ressourcen verwendet, dann werden beide Pfade quasi-parallel ausgeführt (der Pfad mit x_2 muss auf die Ausführung von x_6 warten). Bei Einsatz von nur einer Ressource wird zuerst der Pfad mit x_4 und dann der Pfad mit x_1 ausgeführt.*

Beispiel 2.3.5. *Man betrachte nun ein System bestehend aus den folgenden beiden Pfaden:*

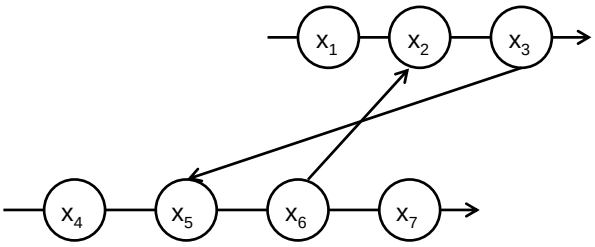


Abbildung 2.3.: Pfade mit Deadlock

Die Pfade in diesem Beispiel unterscheiden sich zu denen aus Beispiel 2.3.1 Abbildung 2.1 unter anderem durch die zusätzliche Relation $x_3 <_{\mathfrak{A}} x_5$. Die Adjazenzmatrix von $<_{\mathfrak{A}}$ sieht hier wie folgt aus.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_1	*0*	*1*	*1*	*0*	*1*	*1*	*1*
x_2	*0*	*1*	*1*	*0*	*1*	*1*	*1*
x_3	*0*	*1*	*1*	*0*	*1*	*1*	*1*
x_4	*0*	*1*	*1*	*0*	*1*	*1*	*1*
x_5	*0*	*1*	*1*	*0*	*1*	*1*	*1*
x_6	*0*	*1*	*1*	*0*	*1*	*1*	*1*
x_7	*0*	*0*	*0*	*0*	*0*	*0*	*0*

In dieser Adjazenzmatrix gilt $a_{22} = a_{33} = a_{55} = a_{66} = 1$. Nach Satz 2.3.2 gilt nun, dass es kein ausführbares Schedule gibt. Dies ist auch anschaulich klar, denn es müsste beispielsweise Auftrag x_2 vor den Aufträgen x_3, x_5 und x_6 ausgeführt werden. Der Auftrag x_6 wiederum muss allerdings vor Auftrag x_2 ausgeführt werden. Daran sieht man, dass eine Ausführung dieser beiden Pfade mit der hier gegebenen Relation $<_{\mathfrak{A}}$ nicht möglich ist.

Soweit ist also die Frage nach der *Ausführbarkeit* beantwortet. Im Weiteren wird daher vorausgesetzt, dass das System $\mathcal{S}$ der Pfade ausführbar ist, das heißt dass $<_{\mathfrak{A}}$ irreflexiv auf den zugehörigen Aufträgen ist. Denn sollte sie das nicht sein, kann nur der Anwender, der die Semantik der Pfade kennt, die Verklemmung beheben.

Wie findet man nun aber eine konkrete Ausführungsreihenfolge? Eine triviale Antwort darauf kann man geben, wenn man davon ausgeht, dass mindestens soviele Ressourcen wie Pfade existieren. Dann kann jeder Pfad einer eigenen Ressource zugeordnet und dort ausgeführt werden.

Im Regelfall werden aber nicht beliebig viele Ressourcen zur Verfügung stehen. Daraus ergibt sich die Frage, ob es auch möglich ist, mit einer geringeren Ressour-

cenanzahl auszukommen. Dies ist in einigen Fällen sehr wohl der Fall. Dazu wird eine weitere Relation auf der Menge der Pfade eines Systems definiert.

Definition 2.3.4. *Die (Pfad-)Relation $<$ auf dem System $\mathcal{S}$ der Pfade ist die kleinste Relation, für die gilt:*

- *Wenn für zwei Pfade X und Y aus $\mathcal{S}$ $(X \neq Y)$ gilt, dass es Aufträge x und y gibt mit $x \in X$ und $y \in Y$ mit $x <_{\mathfrak{A}} y$, dann gilt $X < Y$.*
- *Wenn für drei Pfade X, Y und Z aus $\mathcal{S}$ gilt $X < Y$ und $Y < Z$, dann gilt auch $X < Z$.*

Bemerkung 2.3.1. Salopp gesagt wird durch $X < Y$ ausgedrückt, dass ein Teil eines Pfads Y auf die Fertigstellung eines Auftrages eines anderen Pfades X warten muss. Für den Fall, dass diese beiden Pfade auf derselben Ressource ausgeführt werden können muss notwendigerweise gelten, dass X vor Y auf dieser Ressource ausgeführt wird, da die Pfade atomar ausgeführt werden müssen (siehe Definition 2.2.2).

Auch in dieser Definition wird die *Eindeutigkeit* der Relation $<$ durch die Forderung nach ihrer Minimalität gewährleistet. Der Beweis geht analog zu dem in den Bemerkungen zu Definition 2.2.2.

Die *Existenz* der Relation ist auch klar; auch sie kann von Hand konstruiert werden und in Matrixform dargestellt werden: Man kann sie gewinnen, indem man eine „Hintereinander"-Relation mittels der Relation $<_{\mathfrak{A}}$ zwischen jeweils zwei Pfaden des Systems aufstellt (analog zur Definition der „Hintereinander"-Relation auf Aufträgen) und dann die transitive Hülle bildet. Offensichtlich wird so die vollständige Relation $<$ gewonnen. Unter der Annahme, dass $\mathcal{S}$ genau n Pfade enthält, sieht ein Algorithmus dazu wie folgt aus:

Algorithmus 1 : Gewinnung der Relation $<$ auf den Pfaden eines Systems $\mathcal{S}$

Eingabe : $k \times k$-Matrix $\mathfrak{A}$, die die Relation $<_{\mathfrak{A}}$ zwischen den Aufträgen darstellt. Leere $n \times n$-Matrizen A und D, die mit Nullen besetzt sind

Ausgabe : Matrix A, die die Relation $<$ zwischen den Pfaden eines Systems $\mathcal{S}$ präsentiert

Beginn

  für alle *Zeilen der Matrix* $\mathfrak{A}$ **tue**

    Laufe durch die Zeile. Für jeden Eintrag mit einer 1 in $\mathfrak{A}$ kennzeichne in der Matrix D die zu diesen Aufträgen gehörenden Pfade mit einer 1. Dabei wird der Pfad, der zum Auftrag der Zeile aus $\mathfrak{A}$ gehört in der entsprechenden Zeile in D gekennzeichnet.

  Ende

  $A :=$ Transitive Hülle(D)

Ende

Nach dem bereits Gesagten ist die *Korrektheit* des Algorithmus klar. Die *Terminierung* folgt aus der Tatsache, dass die Matrix $\mathfrak{A}$ maximal einmal durchsucht wird, und der Terminierung des Algorithmus von Warshall zur Berechnung der transitiven Hülle. Der *Aufwand*, der für diesen Algorithmus benötigt wird, ist ebenfalls einfach zu berechnen: Die Matrix $\mathfrak{A}$ wird höchstens einmal durchsucht, das heißt, dass maximal k^2 Vergleichsoperationen auf ihr durchgeführt werden. In der Matrix D werden daraufhin maximal k^2 Einser gesetzt. Zum Berechnen der transitiven Hülle von D werden nun maximal n^2 Vergleiche und n^3 arithmetische Operationen durchgeführt.

Die Relation $<$ ist per definitionem *transitiv*. Eine weitere wichtige Eigenschaft ist wieder, ob Zyklen auftreten können, also ob $<$ irreflexiv ist. Dazu kann die Hauptdiagonale der Adjazenzmatrix von $<$ durchsucht werden, und wenn diese nur Nuller enthält, ist die Relation irreflexiv. Der Aufwand beträgt n Vergleichoperationen bei n Pfaden.

Insgesamt ergeben sich also folgende Aufwände zur Berechnung der Relation $<$ und der Prüfung auf ihre Irreflexivität:

- $k^2 + n^2 + n$ Vergleiche
- $k^2 + n^3$ arithmetische Operationen

Bemerkung 2.3.2. Durch die Relation $<$ auf den Pfaden wird die Relation $<_{\mathfrak{A}}$ auf diese erweitert. Wenn für zwei Pfade X und Y also genau $X < Y$ gilt, so können sie in dieser Reihenfolge auf einer Ressource ausgeführt werden. Somit ermöglicht das Aufstellen der Relation $<$ auf den Pfaden eine Erstellung eines Ausführplanes.

Definition 2.3.5. *Das Tupel* $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ *bezeichne ein System* $\mathcal{S}$ *von Pfaden,* $\mathcal{S}_{\mathfrak{A}}$ *die darin vorkommende Menge von Aufträgen,* $<_{\mathfrak{A}}$ *eine auf* $\mathcal{S}_{\mathfrak{A}}$ *definierte Relation gemäß Definition 2.2.2 und* $<$ *eine auf* $\mathcal{S}$ *definierte Relation gemäß Definition 2.3.4.*

Es werde vereinbart, dass im Folgenden das Tupel $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ gemeint ist, wenn von einem System $\mathcal{S}$ gesprochen wird. Das Tupel wird nur noch explizit angegeben, wenn es für den Kontext benötigt wird.

Definition 2.3.6. *Gegeben seien das Tupel* $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}})$ *und das daraus resultierende Tupel* $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$*. Dann ist das Tupel* $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ *ausführbar genau dann wenn das Tupel* $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}})$ *ausführbar ist.*

Es werde vereinbart, dass im Folgenden gemeint ist, dass das Tupel $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ ausführbar ist, wenn von einem ausführbaren System $\mathcal{S}$ gesprochen wird. Das Tupel wird nur noch explizit angegeben, wenn es für den Kontext benötigt wird.

Wenn Zyklen in $\mathcal{S}$ bezüglich $<$ auftreten, dann kompliziert das die Zuordnung der Pfade zu Ressourcen, wie im folgendem Beispiel gezeigt wird.

Beispiel 2.3.6. *Man betrachte die Pfade eines Systems aus Abbildung 2.2. Sei* X_1 *der Pfad, der Auftrag* x_1 *enthält und* X_2 *der Pfad, der Auftrag* x_4 *enthält. Es gilt dann* $X_1 < X_2$ *und* $X_2 < X_1$ *(Und somit aufgrund der Transitivität* $X_1 < X_1$ *und* $X_2 < X_2$*. Die Reflexivität kommt also durch Zyklen in der <-Relation zustande.). Die beiden Pfade* X_1 *und* X_2 *können daher nicht von einer Ressource ausgeführt werden, aber sie können problemlos von zwei verschiedenen Ressourcen ausgeführt werden. Der Pfad mit dem Auftrag* x_8 *kann dann von beiden Ressourcen ausgeführt werden, allerdings dort entweder vor* X_1 *oder vor* X_2*.*

3. Ausführungspläne

In diesem Kapitel werden Methoden zur Erstellung von Ausführungsplänen auf den Pfaden eines ausführbaren System vorgestellt. Im folgenden Unterkapitel 3.1 wird zunächst ein Ansatz besprochen, der davon ausgeht, dass die Relation $<$ irreflexiv auf den Pfaden eines Systems ist. In Unterkapitel 3.2 werden dann Methoden zur Erstellung von Ausführungsplänen vorgestellt unter der Voraussetzung, dass $<$ nicht irreflexiv auf den Pfaden eines System ist. In Unterkapitel 3.3 werden die soweit entwickelten Methoden zusammengefasst, während ihre Laufzeiten in Unterkapitel 3.4 ermittelt werden. Die durch die entwickelten Sätze und Methoden erzielbaren Laufzeiteinsparungen werden in Unterkapitel 3.5 untersucht. In Unterkapitel 3.6 wird die Weiterverwendbarkeit von bereits existierenden Ausführungsplänen untersucht, falls aus einem System Pfade entfernt oder Pfade hinzugefügt werden.

3.1. Ausführungspläne auf Systemen mit irreflexiver Pfad-Relation

Wenn $<$ auf den Pfaden irreflexiv ist, dann ist kann man relativ einfach eine Reihenfolge (oder auch topologische Sortierung) auf ihnen festlegen. Die Irreflexivität garantiert nämlich eine Art „globale Zyklenfreiheit“ auf dem System, wodurch man sie in Ketten von Pfaden einteilen kann, die sogar von einer einzigen Ressource ausgeführt werden können.

Der Algorithmus 2 leistet eine Sequentialisierung der Pfade des Systems $\mathcal{S}$.

Mit der Abbruchbedingung der inneren Schleife, dass „$\mathcal{S}$ sich nicht mehr ändert“ ist gemeint, dass für alle verbliebenen Pfade in $\mathcal{S}$ nun gelten muss, dass sie nicht mehr in Relation zu einem Pfad in $\mathcal{S}_X$ stehen.

Satz 3.1.1. *In jedem Iterationsschritt von Algorithmus 2 sind die Pfade in $\mathcal{S}_X$ wohlgeordnet. Das bedeutet, dass die Pfade darin in einer sequentiellen Reihenfolge vorliegen und die Relation $<$ zwischen den Pfaden aus $\mathcal{S}_X$ trotzdem eingehalten wird.*

Beweis. Wird induktiv duchgeführt.
Induktionsanfang: Nur X ist Element von $\mathcal{S}_X$, also ist $\mathcal{S}_X$ wohlgeordnet.
Induktionsvoraussetzung: Sei $\mathcal{S}_X$ wohlgeordnet im n-ten Schritt der Iteration. Dabei sei n beliebig, aber fest.

Algorithmus 2 : Sequentialisierung eines Systems $\mathcal{S}$ von Pfaden

Eingabe : Tupel $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ mit $\mathcal{S} \neq \emptyset$

Ausgabe : Menge von Teilsystemen $\mathcal{S}_X$ von $\mathcal{S}$; die Pfade der Teilsysteme sind vollständig geordnet bezüglich $<$

solange $\mathcal{S} \neq \emptyset$ **tue**

  Wähle einen Pfad $X \in \mathcal{S}$, füge ihn zum leeren Teilsystem $\mathcal{S}_X$ hinzu und lösche ihn aus $\mathcal{S}$. Für alle verbleibenden Pfade $Y \in \mathcal{S}$

  wiederhole

    Gilt $Y < Z$ $(Z < Y)$ für $Z \in \mathcal{S}_X$, dann sortiere Y links (rechts) von Z ein und lösche ihn aus $\mathcal{S}$. Befindet sich links (rechts) von Y noch ein Pfad W mit $Y < W$ $(W < Y)$, dann verschiebe Y links (rechts) davon.

  bis *sich* $\mathcal{S}$ *nicht mehr ändert*;

Ende

Induktionsschluß: Sei $Y \in \mathcal{S}$. Wenn Y zu keinem Pfad in $\mathcal{S}_X$ in Relation steht, so bleibt $\mathcal{S}_X$ unverändert und die Aussage gilt.
Sonst gelte o.B.d.A. $Y < Z$ für ein $Z \in \mathcal{S}_X$. Dann existiert ein am weitesten links stehendes $V \in \mathcal{S}_X$ mit $Y < V$ und das Teilsystem wird erweitert zu $\mathcal{S}_X = \{\ldots, Y, V, \ldots\}$. Aufgrund Konstruktion gilt, dass keiner der Pfade W links von Y in Relation $Y < W$ steht. Das bedeutet, dass Y auf diese Weise auf der linken Seite von V widerspruchsfrei eingeordnet wurde. Bleibt nur noch die Frage, ob es ein W rechts von V geben kann mit $W < Y$ (beachte, dass $Y < V$ gilt)? Wenn es ein solches W gäbe würde aufgrund der Transitivität der Relation $<$ gelten $W < V$ und somit wäre W nicht widerspruchsfrei eingeordnet gewesen im Widerspruch zur Induktionsvoraussetzung. Also existiert kein solches W und das System $\mathcal{S}_X$ ist auch nach dem Einordnen von Y wohlgeordnet. □

Die Irreflexivität der Relation $<$ ist sehr wichtig für diesen Beweis, da durch sie eindeutig festgelegt ist, dass ein Pfad vor einem anderen ausgeführt werden muss. Nur auf dieser Basis ist es möglich durch Algorithmus 2 Ketten zu definieren.

Natürlich muss noch gezeigt werden, dass der Algorithmus 2 terminiert.

Beweis. Das System $\mathcal{S}$ ist nach Voraussetzung nicht leer aber endlich und enthalte $\|\mathcal{S}\| = n$ Pfade. Durch die äußere Schleife wird beim ersten Durchgang ein Pfad X ausgewählt, der zum Teilsystem $\mathcal{S}_X$ hinzugefügt wird. Nun wird die innere Schleife iteriert. Es müssen die verbliebenen $n-1$ Pfade von $\mathcal{S}$ mit dem Pfad in $\mathcal{S}_X$ verglichen werden. Für den Fall, dass nun ein Pfad gefunden wird, der zu X in Relation steht, müssen die verbliebenen $n-2$ Pfade mit den nun 2 in $\mathcal{S}_X$ befindlichen Pfaden verglichen werden, und so weiter. Für die Aufwandsabschätzung muss berücksichtigt werden, dass ein Vergleich bezüglich $<$ auf beiden Seiten eines Pfades durchgeführt wird.

Das bedeutet, dass in dem Schleifendurchgang für $\mathcal{S}_X$ maximal

$$\text{Anzahl Vergleiche} = 2\sum_{k=1}^{n-1} k(n-k) \tag{3.1}$$

durchgeführt werden müssen. Dann ist die innere und auch die äußere Schleife beendet.

Für den Fall, dass das System $\mathcal{S}$ der Pfade sich bereits früher nicht mehr ändert, wird die innere Schleife beendet und eine neue Iteration der äußeren Schleife eingeleitet mit einem neuen Teilsystem $\mathcal{S}_Z$. Dabei wird $\mathcal{S}$ wiederum um einen Pfad verkleinert. Wie oben wird jetzt die innere Schleife ausgeführt, die auch durch die restliche Größe von $\mathcal{S}$ beschränkt ist.

Daran sieht man nun, dass $\mathcal{S}$ durch beide Schleifen sukzessive verkleinert wird. Beide Schleifen terminieren spätestens, wenn $\mathcal{S}$ leer ist, also terminiert der Algorithmus.

Es ist auch einfach einzusehen, dass der durch Gleichung 3.1 gegebene Aufwand eine Obergrenze für den Aufwand des ganzen Algorithmus ist, da bei späteren Schleifendurchläufen der äußeren Schleife nur noch die restlichen Pfade aus $\mathcal{S}$ benutzt werden.

Weiter werden alle maximal alle n Pfade aus der Liste, die $\mathcal{S}$ repräsentiert, gelöscht und in neue Teillisten, die die Teilsysteme darstellen, übernommen. Das bedeutet, das weiterhin maximal $2n$ Änderungen an einer Liste durchgeführt werden. □

Insgesamt werden also

- $2\sum_{k=1}^{n-1} k * (n-k)$ Vergleichsoperationen und
- $2n$ Operationen auf Listen

zur Ausführung des Algorithmus benötigt.

Satz 3.1.2. *Je zwei Teilsysteme $\mathcal{S}_X$ und $\mathcal{S}_Y$ mit $\mathcal{S}_X \neq \mathcal{S}_Y$, die durch den Algorithmus 2 konstruiert worden sind, sind disjunkt und es bestehen keinerlei Abhängigkeiten zwischen den Pfaden der beiden Teilmengen.*

Beweis. Klar aufgrund Konstruktion: Eine Iteration der inneren Schleife für das Teilsystem $\mathcal{S}_X$ wird beendet, sobald sich $\mathcal{S}$ nicht mehr ändert. Das bedeutet, dass sich in $\mathcal{S}$ nun keine Pfade mehr befinden, die zu einem beliebigen Pfad aus $\mathcal{S}_X$ in Relation stehen. Daraus folgt wiederum, dass $\mathcal{S}_X \cap \mathcal{S}_Y = \emptyset$. □

Als Folge ergibt sich, dass die Pfade, die in verschiedenen so konstruierten Teilsystemen liegen, in einer beliebigen Reihenfolge hintereinander auf einer Ressource oder aber auch parallel auf verschiedenen Ressourcen ausgeführt werden können.

Weiter ist auch klar, dass die Reihenfolge von Pfaden innerhalb eines Teilsystems $\mathcal{S}_X$, die durch den Algorithmus definiert wird, nicht eindeutig ist. Sie hängt vielmehr von X und der Struktur von $<$ ab. Dies wird in Beispiel 3.1.2 verdeutlicht.

Beispiel 3.1.1. *Sequentialisierung von Pfaden (1)*
Man betrachte in Beispiel 2.3.1 die Abbildung 2.1. Sei Y der Pfad, der den Auftrag x_1 enthält und X der Pfad, der x_4 enthält. Dann gilt offensichtlich $X < Y$ und die Relation $<$ ist irreflexiv und transitiv auf dem System $\mathcal{S} = \{X, Y\}$. Eine Anwendung von Algorithmus 2 auf $\mathcal{S}$ ergibt nun:

1. *Auswahl von Y in der äußeren Schleife, somit Initialisierung von $\mathcal{S}_Y$ und löschen von Y aus $\mathcal{S}$.*

2. *Durch die innere Schleife wird X vor Y eingefügt, da $X < Y$. Außerdem wird X aus $\mathcal{S}$ entfernt. Also gilt $\mathcal{S} = \emptyset$ und der Algorithmus wird beendet.*

Das Resultat der Anwendung des Algorithmus ist also, dass X und Y in dieser Reihenfolge ausgeführt werden müssen.

Beispiel 3.1.2. *Sequentialisierung von Pfaden (2)*
Nun wird der Algorithmus an einem komplizierten Beispiel vorgestellt. Dabei werden nicht mehr die kompletten Pfade gezeigt, sondern nur noch Kästchen zur Repräsentation der Pfade und ihre Abhängigkeiten, siehe Abbildung 3.1. Dabei bedeutet $X_2 \longrightarrow X_1$, dass Pfad X_1 auf Pfad X_2 warten muss (analog zur Notation bei den Aufträgen), also gilt Pfad X_2 <Pfad X_1.

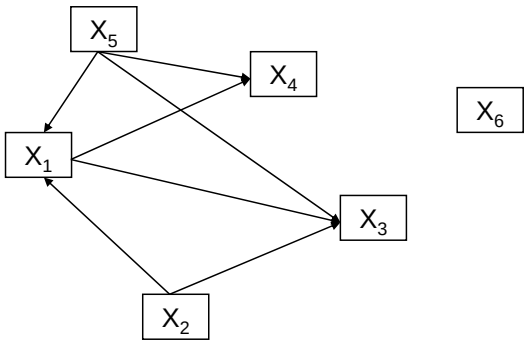


Abbildung 3.1.: Pfadsystem, das sequentialisiert werden soll

Die zu $<$ gehörende Adjazenzmatrix auf diesen Pfaden sieht dann wie folgt aus:

	X_1	X_2	X_3	X_4	X_5	X_6
X_1	*0*	*0*	*1*	*1*	*0*	*0*
X_2	*1*	*0*	*1*	*1*	*0*	*0*
X_3	*0*	*0*	*0*	*0*	*0*	*0*
X_4	*0*	*0*	*0*	*0*	*0*	*0*
X_5	*1*	*0*	*1*	*1*	*0*	*0*
X_6	*0*	*0*	*0*	*0*	*0*	*0*

Wieder sieht man an der Matrix, dass $<$ irreflexiv ist. Eine Anwendung von Algorithmus 2 sieht aus wie folgt, wobei o.B.d.A. $\mathcal{S} = \{X_1, X_2, X_3, X_4, X_5, X_6\}$ gelte:

- *Im ersten Schritt werde beispielsweise X_4 als Initialisierungspfad ausgewählt. Es gelte nun also $\mathcal{S} = \{X_1, X_2, X_3, X_5, X_6\}$ und $M_{\{X_4\}} = \{X_4\}$.*
- *Nun wird in der inneren Schleife gesucht, ob es noch Pfade gibt, die zu Pfad X_4 in Relation stehen oder zu denen X_4 in Relation steht.*

  Zunächst wird mit X_3 verglichen, der zu X_4 in keiner Relation steht. Daher bleiben $\mathcal{S}$ und $\mathcal{S}_{\{X_4\}}$ in dieser Iteration unverändert.
- *Nun wird mit Pfad X_2 verglichen. Es gilt $X_2 < X_4$ und somit gilt nach dem Algorithmus $\mathcal{S} = \{X_1, X_3, X_5, X_6\}$ und $\mathcal{S}_{\{X_4\}} = \{X_2, X_4\}$.*
- *Für X_1 gilt nun: $X_1 < X_4$ aber $X_1 \not< X_2$. Also: $\mathcal{S} = \{X_3, X_5, X_6\}$ und $\mathcal{S}_{\{X_4\}} = \{X_2, X_1, X_4\}$.*
- *Pfad X_6 steht in keinerlei Relation zu den Pfaden aus $\mathcal{S}_{\{X_4\}}$, also bleiben die Systeme in diesem Iterationsschritt unverändert.*
- *Für Pfad X_5 gilt nun $X_5 < X_4$ und $X_5 < X_1$, aber $X_5 \not< X_2$. Nach dem Algorithmus gilt nun: $\mathcal{S} = \{X_3, X_6\}$ und $\mathcal{S}_{\{X_4\}} = \{X_2, X_5, X_1, X_4\}$.*
- *Nun wird wieder mit Pfad X_3 verglichen und es gilt $X_1, X_2, X_5 < X_3$, aber $X_4 \not< X_3$. Also gilt: $\mathcal{S} = \{X_6\}$ und $\mathcal{S}_{\{X_4\}} = \{X_2, X_5, X_1, X_3, X_4\}$.*
- *Ein weiterer Vergleich mit Pfad X_6 zeigt keine neuen Abhängigkeiten auf. Also wird die innere Schleife beendet.*
- *Da $\mathcal{S} \neq \emptyset$ wird ein neues System $\mathcal{S}_{\{X_6\}} = \{X_6\}$ etabliert. Danach gilt $\mathcal{S} = \emptyset$.*
- *Daher wird die innere Schleife nun sofort wieder verlassen und die äußere Schleife terminiert genauso.*

Durch die Anwendung des Algorithmus wurden die beiden Teilsysteme $\mathcal{S}_{\{X_4\}} = \{X_2, X_5, X_1, X_3, X_4\}$ und $\mathcal{S}_{\{X_6\}} = \{X_6\}$ etabliert. In jedem Teilsystem ist eine Ausführungsreihenfolge der Pfade widerspruchsfrei definiert. Ein Scheduler hat nun die Freiheitsgrade beispielsweise alle Pfade auf einer einzigen Ressource auszuführen. Dabei kann er noch feiner unterscheiden, ob zuerst Pfad X_6 ausgeführt werden soll und dann die Pfade von $\mathcal{S}_{\{X_4\}}$ oder umgekehrt.

Alternativ kann er auch die Pfade von $\mathcal{S}_{\{X_4\}}$ auf einer Ressource ausführen und den Pfad X_6 auf einer anderen Ressource.

Eine weitere Möglichkeit ist natürlich, die Pfade aus $\mathcal{S}_{\{X_4\}}$ noch feiner unterteilt verschiedenen Ressourcen zuzuordnen solange die Reihenfolge eingehalten wird. Dies

hängt aber von der konkreten Wahl der Schedulingstrategie ab, auf die in Kapitel 5 eingegangen wird.

Insgesamt wurden $1+1+2+3+3+4+5 = 19$ *Vergleiche durchgeführt. Dies liegt deutlich unter der oberen Anzahl von Vergleichen, die durch Gleichung 3.1 gegeben ist mit* $\sum_{k=1}^{5} k * (6-k) = 35$ *Vergleichen.*

Neben dem in diesem Unterkapitel vorgestellten Algorithmus zur Sequentialisierung von Pfaden gibt es schon bekannte Verfahren, die eine topologische Sortierung auf einer Menge von Elementen (zusammen mit einer irreflexiven Relation auf diesen Elementen) erstellen, siehe [Las61]. Algorithmus 2 wurde trotzdem in dieser Arbeit eingeführt, da er eine größere Anzahl an Freiheitsgraden bietet, wenn es unabhängige Teilsysteme von Pfaden gibt (vergleiche Beispiel 3.1.2), beispielsweise kann ein Scheduler entscheiden, dass beide Mengen auf einer Ressource ausgeführt werden müssen, und er kann sogar die Reihenfolge festlegen, in der sie darauf ausgeführt werden. Auf Schedulingstrategien, die solche Optimierungen vornehmen, wird in Kapitel 5 eingegangen.

Durch Algorithmus 2 wird eine Totalordnung auf den Teilsystemen festgelegt und damit eine Reihenfolge angegeben, in der die Pfade ausgeführt werden können. Trotzdem kann es vorkommen, dass eine Ressource mehrere Pfade des Teilsystems ausführen soll. Sie kann sich dabei strikt an die Totalordnung halten, aber es ist möglich aus $<$ Informationen über die Parallelisierbarkeit der Pfade zu gewinnen (die durch das Anordnen zur Totalordnung verloren gehen) und begrenzt von der Totalordnung abzuweichen. Dies ist beispielsweise der Fall, wenn eine Ressource Pfade X und Y ausführen soll für die weder $X < Y$ noch $Y < X$ gilt und die in der Totalordnung direkt aufeinanderfolgen. Dann spielt es keine Rolle, welcher der beiden Pfade zuerst ausgeführt wird. Die genauere Anwendung dieser Eigenschaften wird in Kapitel 5 besprochen. Hier wird darauf eingegangen wie man berechnen kann, welche Pfade X und Y paarweise parallel ausgeführt werden können. Dies ist, wie eben bereits beschrieben, genau dann der Fall wenn Pfad X nicht auf Pfad Y warten muss und umgekehrt. Dies ist äquivalent dazu, dass in der Matrix A gilt $a_{ij} = a_{ji} = 0$. Wenn man also die transponierte Matrix A^T berechnet und diese zu A addiert, so stehen in der resultierenden Matrix A^P genau an den Positionen parallel ausführbarer Pfade eine Null. Der folgende Algorithmus beschreibt diesen Sachverhalt:

Die Korrektheit des Algorithmus ist nach dem eben Gesagten klar, ebenso wie die Terminierung: Für das Transponieren und das Addieren der Matrizen werden jeweils n^2 arithmetische Operationen benötigt. Wenn für jeden Pfad noch bestimmt werden soll, mit welchen Pfaden desselben Teilsystems er ausführbar ist, so kostet das höchstens weitere $2n^2$ Vergleichsoperationen (unter der Voraussetzung, dass in den Matrizen die zu den Pfaden zugehörigen Sequenzen eingetragen sind), und dazu kommen noch maximal n^2 arithmetische Operationen, um die Listen zu schreiben, die die parallel ausführbaren Pfade eines Pfades kennzeichnen. Insgesamt wird für die Berechnung dieser Listen also ein Aufwand von maximal

Algorithmus 3 : Gewinnung der paarweise parallel ausführbaren Pfade eines Systems $\mathcal{S}$

Eingabe : $n \times n$-Matrix A, die die Relation $<$ zwischen den Pfaden eines Systems $\mathcal{S}$ beschreibt

Ausgabe : $n \times n$-Matrix A^P, deren Nullstellen genau die paarweise parallel ausführbaren Pfade beschreiben

Beginn

Berechne die Transponierte A^T von A.
$A^P = A + A^T$

Ende

- $3n^2$ arithmetischen Operationen und
- $2n^2$ Vergleichsoperationen benötigt.

Bemerkung 3.1.1. Die Matrix A^P ist konstruktionsbedingt symmetrisch, denn es gilt $a^P_{ij} = a_{ij} + a^T_{ij} = a_{ij} + a_{ji} = a^T_{ji} + a_{ji} = a^P_{ji}$ $(1 \leq i \leq n, 1 \leq j \leq n)$. Dies ist auch anschaulich klar, denn die Eigenschaft der parallelen Ausführbarkeit ist symmetrisch: Wenn Pfad X parallel zu Pfad Y ausgeführt werden kann, dann gilt das auch umgekehrt. Offensichtlich gilt aber nicht die Transitiviät der parallelen Ausführbarkeit der Pfade.

Weiterhin überträgt sich die Eigenschaft der parallelen Ausführbarkeit von Pfaden aufgrund der Definition von $<$ auf die darin enthaltenen Aufträge.

In diesem Unterkapitel wurde davon ausgegangen, dass $<$ irreflexiv auf $\mathcal{S}$ ist. Unter Ausnutzung der Transitivität wurde ein Algorithmus entwickelt, durch den eine Sequentialisierung der Pfade einfach ermöglicht wird. Der andere Fall, nämlich dass $<$ nicht irreflexiv ist, stellt sich als schwerer dar und wird im nächsten Unterkapitel besprochen.

3.2. Ausführungspläne auf Systemen mit nicht irreflexiver Pfad-Relation

Die wesentliche Konsequenz, die sich daraus ergibt, wenn die Relation $<$ transitiv und nicht irreflexiv auf den Pfaden ist, ist, dass nun nicht mehr alle Pfade auf einer Ressource ausgeführt werden können; siehe dazu nochmals Beispiel 2.3.6. Dann existiert nämlich ein Zyklus von Pfaden bezüglich $<$.

Die einfachste Möglichkeit, Pfade eines Zyklus zu verteilen, ist jedem Pfad eine eigene Ressource zuzuweisen. Dies ist aber nicht immer möglich, da eventuell nicht genug Ressourcen vorhanden sind oder nicht alle für eine Ausführung geeignet sind.

Um in den weiteren Kapiteln die Formulierung zu vereinfachen, werden zunächst die folgenden Definitionen getroffen:

Definition 3.2.1 (Einschränkung bezüglich Aufträgen)**.** *Das Tupel* $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ *sei gegegeben. Dann heißt eine Teilmenge* $\mathcal{S}^E_{\mathfrak{A}}$ *von* $\mathcal{S}_{\mathfrak{A}}$ *mit den auf die Aufträge der Teilmenge beschränkten Relationen* $<_{\mathfrak{A}}$ ***Einschränkung*** *bezüglich* $<_{\mathfrak{A}}$ *von* $\mathcal{S}_{\mathfrak{A}}$ *auf* $\mathcal{S}^E_{\mathfrak{A}}$*. Für zwei Aufträge* x *und* y *aus* $\mathcal{S}^E_{\mathfrak{A}}$ *gilt also:*

$$x <_{\mathfrak{A}} y \text{ in } \mathcal{S}_{\mathfrak{A}} \Leftrightarrow x <_{\mathfrak{A}} y \text{ in } \mathcal{S}^E_{\mathfrak{A}}$$

Definition 3.2.2 (Einschränkung bezüglich Pfaden)**.** *Das Tupel* $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ *sei gegegeben. Dann heißt eine Teilmenge* $\mathcal{S}^E$ *von* $\mathcal{S}$ *mit den auf die Pfade von* $\mathcal{S}^E$ *beschränkten Relationen* $<$ ***Einschränkung*** *bezüglich* $<$ *von* $\mathcal{S}$ *auf* $\mathcal{S}^E$*. Für zwei Pfade* X *und* Y *aus* $\mathcal{S}^E$ *gilt also:*

$$X < Y \text{ in } \mathcal{S} \Leftrightarrow X < Y \text{ in } \mathcal{S}^E$$

Bemerkung 3.2.1. Es ist klar, dass die Relation $<_{\mathfrak{A}}$ auf einer Einschränkung $\mathcal{S}^E_{\mathfrak{A}}$ von $\mathcal{S}_{\mathfrak{A}}$ irreflexiv und transitiv ist, wenn sie diese Eigenschaften bereits auf $\mathcal{S}_{\mathfrak{A}}$ besitzt. Dies gilt auch für die Relation $<$. Weiter gilt aufgrund der Definitionen, dass eine Einschränkung der Relationen $<_{\mathfrak{A}}$ beziehungsweise $<$ eindeutig ist. Weitere Eigenschaften zu Einschränkungen werden im Anghang A.5 diskutiert.

Definition 3.2.3 (Einschränkung auf ein Teilsystem)**.**
Das Tupel $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ *sei gegegeben. Sei* $\mathcal{S}^E$ *eine Teilmenge von* $\mathcal{S}$*, und* $\mathcal{S}^E_{\mathfrak{A}} \subseteq \mathcal{S}_{\mathfrak{A}}$ *enthalte genau die Aufträge, die in* $\mathcal{S}^E$ *enthalten sind. Dann ist das Tupel* $(\mathcal{S}^E, \mathcal{S}^E_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ *mit den auf* $\mathcal{S}^E_{\mathfrak{A}}$ *und* $\mathcal{S}^E$ *eingeschränkten Relationen* $<_{\mathfrak{A}}$ *und* $<$ *eine* ***Einschränkung auf ein Teilsystem****.*

Natürlich treffen die Aussagen aus Bemerkung 3.2.1 auch auf diese Definition zu.

Definition 3.2.4. *Das Tupel* $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ *sei gegegeben. Dann heißen diejenigen Pfade* X *von* $\mathcal{S}$ ***frei****, für die nicht gilt* $X < X$*.*

Bemerkung 3.2.2. Aus der Definition folgt sofort, dass für einen nicht freien Pfad X gilt: $X < X$.

Definition 3.2.5. *Sei* X *ein nicht freier Pfad eines Systems* $\mathcal{S}$*. Dann ist* $\zeta(X)$*, der zu* X *gehörende Zyklus, wie folgt definiert:*

$$\zeta(X) := \{Y \in \mathcal{S} : \ X < Y \text{ und } Y < X\}$$

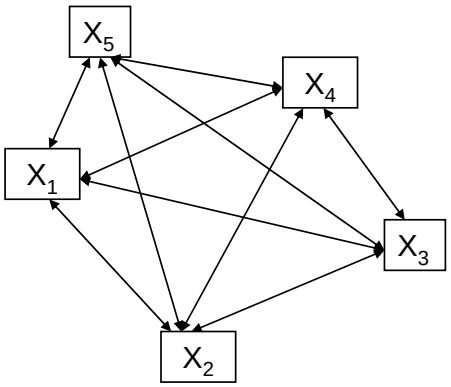


Abbildung 3.2.: Beispiel eines Zyklus von Pfaden bezüglich $<$

Beispiel 3.2.1. *Abbildung 3.2 zeigt eine Beispiel eines Zyklus von fünf Pfaden bezüglich der Relation $<$. Für jeden Pfad X_i und X_j daraus gilt $X_i < X_j$ ($i, j \in \{1, 2, 3, 4, 5\}$).*
Weiterhin gilt $\zeta(X_1) = \zeta(X_2) = \zeta(X_3) = \zeta(X_4) = \zeta(X_5) = \{X_1, X_2, X_3, X_4, X_5\}$.

Aus der Definition folgt unmittelbar, dass $\zeta(X)$ nicht leer ist, also mindestens einen Pfad enthält, und dass jeder nicht freie Pfad in einem solchen Zyklus liegt. Im Folgenden soll dies weiter untersucht und strukturiert werden. Es gelten die Aussagen im folgenden Satz:

Satz 3.2.1.

1. *Jeder Zyklus besteht aus mindestens zwei verschiedenen Pfaden.*
2. *Jeder Pfad in einem Zyklus ist nicht frei, und jeder nicht freie Pfad liegt in einem Zyklus.*
3. *Die Relation $<$ ist innerhalb eines Zyklus eine Äquivalenzrelation.*
4. *Seien X und Y nicht freie Pfade. Dann sind $\zeta(X)$ und $\zeta(Y)$ entweder gleich oder disjunkt.*
5. *Sei X ein nicht freier und Y ein Pfad mit $Y \notin \zeta(X)$. Dann gilt für alle Pfade $Z \in \zeta(X)$: Entweder $Y < Z$ oder $Z < Y$ oder Y und Z stehen nicht in Relation.*

Beweis.
Zu 1.: Folgt direkt aus der Definition 2.3.4 der Relation $<$, da ein Pfad nur vermöge Transitivität zu sich selber in Relation stehen kann.

Zu 2.: Die Behauptung, dass jeder Pfad in einem Zyklus nicht frei ist, folgt direkt aus der Definition eines Zyklus. Ebenso gilt, dass jeder nicht freie Pfad in einem Zyklus liegt, nämlich in seinem eigenen, siehe dazu den Beweis zu 1.

Zu 3.: Die Reflexivität der Pfade eines Zyklus folgt direkt aus der Definition. Auch die Symmetrie folgt aus dieser Definition. Die Transitivität gilt aufgrund der Transitivität der Relation $<$.

Weiter ist klar, dass ein Zyklus eine komplette Äquivalenzklasse darstellt, denn alle Pfade des Zyklus stehen zueinander in Relation. Seien W und Z zwei Pfade aus $\zeta(X)$. Dann gilt $W < X$ und $X < Z$ und somit aufgrund der Transitivität von $<$: $W < Z$. Damit gilt für einen beliebigen Pfad $W \in \zeta(X)$ offensichtlich $\zeta(X) = \zeta(W)$. Dies zeigt die *Repräsentantenunabhängigkeit* eines Zyklus.

Zu 4.: Falls $\zeta(X) \cap \zeta(Y) \neq \emptyset$ gibt es mindestens einen Pfad W, welcher in beiden Zyklen liegt. Aufgrund der Transitivität von $<$ gilt o.B.d.A. $Y \in \zeta(X)$ und somit nach Punkt 3: $\zeta(X) = \zeta(Y)$.

Zu 5.: Annahme, es existieren Z_1, $Z_2 \in \zeta(X)$ mit $Z_1 < Y$ und $Y < Z_2$. Dann gilt $X < Z_1 < Y < Z_2 < X$ und somit wäre Y ein Pfad des Zyklus, Widerspruch. Alternativ gelte o.B.d.A. $Z_1 < Y$ und nicht $Z_2 < Y$ oder $Y < Z_2$. Aber wegen $Z_2 < Z_1$ gilt auch $Z_2 < Y$, Widerspruch. □

3.2.1. Finden von Zyklen

Nach Definition 2.3.4 wird erläutert, wie man die Relation $<$ gewinnen kann, indem man eine Matrix A aufstellt, in der die einzelnen Relationen zwischen den Pfaden gekennzeichnet sind. Die Matrix ist dann notwendigerweise quadratisch (entsprechend der Anzahl n der Pfade im System $\mathcal{S}$) und es gelte für den Eintrag $a_{ij} = 1$ wenn der Pfad X_j nach dem Pfad X_i ausgeführt werden muss; sonst gelte $a_{ij} = 0$. Dabei gilt für i und j: $0 \leq_j^i \leq n$. Aus solch einer Matrix lassen sich vorhandene Zyklen sehr einfach identifizieren, da die wesentliche Eigenschaft eines Zyklus ist, dass alle Pfade darin in Relation $<$ stehen, und zwar in beide Richtungen. Das bedeutet, dass man nur die symmetrischen Anteile der Matrix finden muss, und daran die Zyklen direkt ablesen kann. Der folgende Algorithmus 4 leistet das Gewünschte, indem er aus der Matrix A eine Matrix Ψ gewinnt, die genau die symmetrischen Anteile von A enthält, und diese Matrix dann weiter untersucht.

Zunächst werden einige Eigenschaften der Matrix Ψ betrachtet. Die Matrix wird im Algorithmus mittels der Matrix A und ihrer Transponierten A^T durch komponentenweise Multiplikation der Einträge berechnet: $\psi_{ij} := a_{ij} * a_{ij}^T$. Es ist nun äquivalent:

1. Ein Pfad X des Systems $\mathcal{S}$ ist frei.

Algorithmus 4 : Finden von Zyklen in einem System $\mathcal{S}$ von Pfaden

Eingabe : $n \times n$-Matrix A, die die Relation $<$ zwischen den einzelnen Pfaden eines Systems $\mathcal{S}$ beschreibt

Ausgabe : Menge $\mathcal{S}_\zeta$ aller Zyklen

Beginn

- $\mathcal{S}_\zeta := \emptyset$;
- Berechne die untere Halbmatrix der zu A transponierten Matrix A^T;
- Berechne die untere Halbmatrix der Matrix Ψ mit Einträgen $\psi_{ij} := a_{ij} * (a^T)_{ij}$;
- Betrachte die untere Halbmatrix von Ψ:
- **für** $(i = 1;\ i \leq n - 1;\ i++)$ **tue**
  - **wenn** *der zu i. Spalte gehörige Pfad X nicht als genommen gekennzeichnet ist* ***und*** $\psi_{ii} == 1$ **dann**
    - sei $\zeta(X) := \emptyset$. Alle Pfade dieser Spalte, die mit einer 1 gekennzeichnet sind, werden zu $\zeta(X)$ hinzugefügt und in Ψ als bereits genommen gekennzeichnet. Füge $\zeta(X)$ zu $\mathcal{S}_\zeta$ hinzu.
  - **sonst**
    - tue nichts.
  - **Ende**
- **Ende**

Ende

2. Die zum Pfad X gehörige Zeile und Spalte in der Matrix Ψ ist jeweils komplett mit Nullen besetzt.

3. Der zum Pfad X gehörige Eintrag in der Hauptdiagonalen von Ψ ist Null.

Beweis. Beweis durch Ringschluss.
„1. $\Rightarrow$ 2.“: Wenn ein Pfad X frei ist und man annimmt, dass die zugehörige Zeile oder Spalte in Ψ auch Einser enthält, so bedeutet dies aufgrund Konstruktion von Ψ, dass X nicht frei ist, Widerspruch.
„2. $\Rightarrow$ 3.“: Klar.
„3. $\Rightarrow$ 1.“: Wenn X nicht frei wäre, so würde gelten $X < X$. Dann wäre aber in A in der zu X gehörigen Spalte in der Hauptdiagonalen eine Eins. Dies gilt auch für A^T und somit für Ψ, Widerspruch. □

Nun wird gezeigt, dass der Algorithmus korrekt funktioniert und terminiert.

Beweis.
Korrektheit: Das Berechnen der Matrix Ψ dient dem Finden der symmetrischen Anteile der Matrix A. Ψ ist dann wie A und A^T eine $n \times n$-Matrix und ist aufgrund Konstruktion symmetrisch, siehe auch Anhang A.3. Insbesondere stehen in Ψ aufgrund der Symmetrie Einsen genau an der Stelle von nicht freien Pfaden. Ebenfalls wegen der Symmetrie von Ψ reicht es, im Algorithmus die untere Halbmatrix zu betrachten. Dies wird im folgenden Induktionsbeweis erläutert.

Der Algorithmus sucht nun spaltenweise nach Zyklen.
Induktionsanfang: Gehöre die erste Spalte zum Pfad X_1. Wenn das Hauptdiagonalelement Null ist (also wenn $\psi_{11} = 0$), dann ist der Pfad frei und die Spalte muss nicht weiter untersucht werden. Sonst (also wenn $\psi_{11} = 1$) ist X_1 nicht frei, und alle Pfade in dieser Spalte, die mit einer Eins gekennzeichnet sind, werden zum Zyklus (und somit zur Äquivalenzklasse) $\zeta(X_1)$ hinzugefügt und in Ψ als genommen markiert. Der Zyklus ist vollständig, d.h. bei der Untersuchung der späteren Spalten können keine neuen Pfade mehr hinzukommen: Denn wenn das so wäre, dann gäbe es einen weiteren Pfad Y mit $X_1 < Y$ und dieser wäre nicht in der ersten Spalte aufgeführt. Das ist aber nicht möglich, da dann auch $Y < X_1$ gilt und Y somit in der ersten Spalte aufgeführt sein muss.
Induktionsvoraussetzung: Seien die ersten k Spalten der Matrix Ψ nach Zyklen durchsucht worden. Die dabei gefunden Zyklen sind vollständig und die darin enthaltenen Pfade seien in Ψ als genommen markiert.
Induktionsschluss: Gehöre die $(k+1)$. Spalte zum Pfad X_{k+1}. Wenn dieser Pfad als genommen markiert wurde, so muss X_{k+1} nicht weiter untersucht werden, da er bereits zu einem Zyklus gehört und dieser nach Induktionsvoraussetzung vollständig erfasst ist. Wenn er in keinem der Zyklen vorkommt, dann ist entweder das Hauptdiagonalelement der $(k+1)$. Spalte Null und der Pfad ist frei, oder er liegt in einem Zyklus $\zeta(X_{k+1})$, und alle Pfade in dieser Spalte der unteren Halbmatrix

von Ψ, die mit einer Eins gekennzeichnet sind, werden zum Zyklus (und somit zur Äquivalenzklasse) $\zeta(X_{k+1})$ hinzugefügt und in Ψ als genommen markiert. Wie im Induktionsanfang argumentiert man, dass $\zeta(X_{k+1})$ vollständig ist. Man beachte, dass es reicht den Teil der Spalte der unteren Halbmatrix von Ψ zu untersuchen, da alle Einträge der oberen Halbmatrix an dieser Spalte Null sein müssen. Denn wäre einer dieser Einträge Eins, so wäre X_{k+1} bereits in einem Zyklus enthalten, Widerspruch.

An diesem Beweis sieht man auch, dass die letzte Spalte und somit der letzte Pfad X_n nicht mehr untersucht werden muss: Wenn X_n frei ist, dann ist die komplette Spalte mit Nullen gefüllt. Wenn X_n nicht frei ist, dann liegt er in einem Zyklus, zu dem mindestens ein anderer Pfad X_j (mit $0 \leq j \leq n-1$) gehört, und X_n wurde bereits hier erfasst.

Schließlich ist aufgrund Konstruktion von Ψ klar, dass durch den Algorithmus alle Zyklen des Systems $\mathcal{S}$ erfasst werden. Somit stellt die Menge $\mathcal{S}_\zeta$ eine disjunkte Überdeckung der nicht freien Pfade dar.
Terminierung: Klar aufgrund Induktion und da die Anzahl n der Pfade endlich ist. □

Nun ist noch interessant, welchen Aufwand der Algorithmus verursacht. Dies ist aber einfach auszurechnen:

- Für das Berechnen der unteren Halbmatrix von A^T werden $\frac{n^2-n}{2}$ Vertauschungsschritte benötigt, beachte: Die Hauptdiagonale muss nicht beachtet werden .

- Für das Berechnen der unteren Halbmatrix von Ψ werden $\frac{n^2+n}{2}$ Multiplikationen benötigt.

- Die Schleife wird $n-1$ Mal durchlaufen, und dabei werden jeweils *zwei* Bedingungen geprüft. Der Aufwand dafür beträgt somit $2(n-1)$.

  Da jeder Zyklus aus mindestens zwei Pfaden besteht, existieren höchstens $\left\lfloor \frac{n}{2} \right\rfloor$ Zyklen im System $\mathcal{S}$. Daher müssen maximal $\left\lfloor \frac{n}{2} \right\rfloor$ Spalten der unteren Halbmatrix von Ψ untersucht werden. Im schlimmsten Fall müssen die Einträge der ersten $\left\lfloor \frac{n}{2} \right\rfloor$ Spalten der unteren Halbmatrix betrachtet werden:

$$\begin{aligned}\sum_{i=n-\left\lfloor \frac{n}{2} \right\rfloor+1}^{n} i &= \sum_{i=1}^{n} i - \sum_{i=1}^{n-\left\lfloor \frac{n}{2} \right\rfloor} i \\ &= \frac{n(n+1)}{2} - \frac{(n-\left\lfloor \frac{n}{2} \right\rfloor)(n-\left\lfloor \frac{n}{2} \right\rfloor+1)}{2} \\ &= n\left\lfloor \frac{n}{2} \right\rfloor + \frac{\left\lfloor \frac{n}{2} \right\rfloor}{2} - \frac{\left\lfloor \frac{n}{2} \right\rfloor^2}{2}\end{aligned} \tag{3.2}$$

Dies gilt für $n \geq 2$, da sonst kein Zyklus auftreten kann.

- Es werden m Pfade, die Zyklen zugeordnet wurden, als genommen markiert. Dabei gilt $m \leq n$.
- Weiter werden alle nicht freien Pfade ihrem Zyklus zugeordnet. Das bedeutet, dass m' Pfade in Listen eingeordnet werden müssen. Dabei gilt $m \leq m' \leq n$.

Beispiel 3.2.2. *Zunächst betrachte man ein System mit $n = 3$ Pfaden.*

- *Es werden $\frac{3^2-3}{2} = 3$ Vertauschungsschritte zur Berechnung der unteren Halbmatrix von A^T benötigt.*
- *Weitere $\frac{3^2+3}{2} = 6$ Multiplikationen werden zur Berechnung der unteren Halbmatrix von Ψ benötigt.*
- *In der Schleife müssen schließlich $2 * (3 - 1) = 4$ Bedingungen geprüft werden.*

  Dann müssen maximal 3 Einträge der Matrix Ψ geprüft werden. Dies liegt daran, dass im schlimmsten Fall die erste Spalte untersucht werden muss mit 3 Einträgen. Dann liegt der zugehörige Pfad nämlich in einem Zyklus. Nach der Untersuchung ist klar, welche der beiden anderen Pfade auch noch im Zyklus liegen.
- *Während dieser Prüfung werden die Pfade in Zyklen markiert; dies sind maximal 3.*
- *Es werden maximal 3 Pfade als nicht frei identifiziert und in Listen bezüglich ihrer Zyklen eingetragen.*

Wenn man ein System mit $n = 20$ Pfaden betrachtet, so ergibt sich folgender maximaler Aufwand für den Algorithmus:

- 190 *Vertauschungsschritte zur Berechnung der unteren Halbmatrix von A^T.*
- 210 *Multiplikationen zur Berechnung der unteren Halbmatrix von Ψ.*
- *In der Schleife müssen $2 * (20 - 1) = 38$ Bedingungen geprüft werden.*

  Dann müssen maximal 155 Einträge der Matrix Ψ geprüft werden.
- *Während dieser Prüfung werden die Pfade in Zyklen markiert; dies sind maximal 20.*
- *Es werden maximal 20 Pfade als nicht frei identifiziert und in Listen bezüglich ihrer Zyklen eingetragen.*

3.2.2. Sequentialisierung trotz Zyklen

Bis jetzt wurden in diesem Kapitel der Begriff der Zyklen innerhalb von Pfaden eingeführt, und wie man diese finden kann. Es ist aber noch nicht klar, wie man damit umgehen soll. Dies wird in nun beantwortet. Aus dem Beginn des Kapitels ist nun bekannt, dass man die Pfade eines Systems $\mathcal{S}$ in zwei Kategorien einteilen kann, nämlich in freie Pfade und nicht freie Pfade. Dabei ist jeder nicht freie Pfad Teil eines Zyklus, der aus mindestens zwei Pfaden besteht. Für jeden Zyklus gilt sogar, dass die Relation $<$ auf dem Zyklus eine Äquivalenzrelation darstellt.

Nun liegt die Idee nahe, die Zyklen zunächst nicht komplett mit allen ihren Pfaden zu betrachten, sondern als Einheit. Dies macht auch daher Sinn, da jeder Zyklus schon eine volle Äquivalenzklasse darstellt. Dazu betrachte man nochmals die Menge $\mathcal{S}_\zeta$ aus Algorithmus 4. Sie enthält alle Zyklen (und damit alle nicht freien Pfade), die im System $\mathcal{S}$ vorkommen.

Definition 3.2.6. *Das Tupel $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ sei gegegeben. Dann ist das System $\mathcal{S}_\sim$ wie folgt definiert:*

$$S_\sim := \{X \in \mathcal{S} : X \textit{ ist frei}\} \cup \bigcup_{\zeta(Y) \in \mathcal{S}_\zeta} Y \tag{3.3}$$

Das System $\mathcal{S}_\sim$, welches in dieser Definition beschrieben wird, enthält also alle freien Pfade des Systems $\mathcal{S}$, und weiterhin genau einen „Repräsentanten"-Pfad aus jedem Zyklus: Die Pfade eines Zyklus $\zeta(Y)$ werden durch einen Repräsentanten Y in der Menge $\mathcal{S}_\sim$ ersetzt. In der Begründung zur Wohldefiniertheit der Relation $<_\sim$ auf der Menge $\mathcal{S}_\sim$ im nächsten Absatz wird klar, dass es unerheblich ist, welcher Pfad aus einem Zyklus als dessen Repräsentant gewählt wird, da alle Pfade eines Zyklus bezüglich der Relation $<$ dieselben Eigenschaften haben.

Nun soll auf dem System $\mathcal{S}_\sim$ eine Relation, vergleichbar zur Relation $<$, definiert werden.

Definition 3.2.7. *Die Relation $<_\sim$ auf dem System $\mathcal{S}_\sim$ ist die kleinste Relation mit den folgenden Eigenschaften:*

- *Wenn X und Z freie Pfade in $\mathcal{S}$ sind mit $X < Z$, so gilt $X <_\sim Z$ in $\mathcal{S}_\sim$.*
- *Wenn X ein freier Pfad in $\mathcal{S}$ ist und Y ein nicht freier Pfad und Repräsentant seiner Äquivalenzklasse ist mit $X < Y$, so gilt $X <_\sim Y$ in $\mathcal{S}_\sim$. Dies gilt analog im umgekehrten Fall.*
- *Wenn X und Y nicht freie Pfade und Repräsentanten ihrer ungleichen Äquivalenzklassen sind mit $X < Y$, so gilt so gilt $X <_\sim Y$ in $\mathcal{S}_\sim$.*

Bemerkung 3.2.3. $<_\sim$ ist *wohldefiniert*: Dies ist klar auf den freien Pfaden. In der Kombination freier Pfad und Äquivalenzklasse ist das auch klar, da ein freier Pfad zu jedem Pfad der Äquivalenzklasse in derselben Relation steht. Dasselbe gilt auch für die Kombination von zwei Äquivalenzklassen, siehe Satz 3.2.1 Punkt 5.

Die *Eindeutigkeit* von $<_\sim$ folgt unmittelbar aus der Definition, da die Relation zwischen zwei Elementen der Menge $\mathcal{S}_\sim$ genau einer der drei gegebenen Eigenschaften genügen muss.

Die Relation $<_\sim$ ist auch *transitiv*: Seien X, Y und Z drei Pfade aus $\mathcal{S}_\sim$ mit $X <_\sim Y$ und $Y <_\sim Z$. Dann gilt aufgrund der Definition von $<_\sim$ und der Transitivität von $<$: $X <_\sim Z$. Die Transitivität wurde in dieser Definition nicht gefordert, sie ist vielmehr eine Eigenschaft, die sich aus der Relation $<$ auf die Relation $<_\sim$ durchdrückt.

Eine weitere wichtige Eigenschaft ist die *Irreflexivität* von $<_\sim$: Anschaulich ist das klar, denn alle Zyklen in $\mathcal{S}$ werden in $\mathcal{S}_\sim$ auf jeweils einen einzigen Repräsentantenpfad reduziert und die Relation $<$ auf dieses System adaptiert. Formal kann man sich den Sachverhalt so klar machen: Annahme, es gäbe ein $X \in \mathcal{S}_\sim$ mit $X <_\sim X$. Wenn X ein freier Pfad aus $\mathcal{S}$ wäre, so müsste bereits dort $X < X$ gelten, was ein Widerspruch zur Freiheit von X wäre. Wäre X ein Repräsentant einer Äquivalenzklasse, so käme $X <_\sim X$ aufgrund Definition überhaupt nicht in Frage.

Die *Gewinnung* von $<_\sim$ ist einfach: Man betrachte die Matrix A, die die Relation $<$ beschreibt. Darin streiche man die zu den nicht freien Pfaden gehörigen Zeilen und Spalten mit Ausnahme der Zeilen und Spalten der Repräsentantenpfade von Zyklen. Das Resultat ist offensichtlich eine $n_\sim \times n_\sim$-Matrix $A_\sim$ ($n_\sim \leq n$, $n_\sim \in \mathbb{N}$), die in den Zeilen und Spalten genau die freien Pfade und die Repräsentanten der Zyklen enthält und somit genau die Elemente des Systems $\mathcal{S}_\sim$ enthält. Weiter werde die komplette Hauptdiagonale von $A_\sim$ mit Nullen besetzt. Dann wird durch $A_\sim$ genau die Relation $<_\sim$ beschrieben: Sei X ein freier Pfad in $\mathcal{S}$. Dann ist X auch ein Element von $\mathcal{S}_\sim$. Man betrachte die zu X zugehörige Zeile in der Matrix A. Darin sind genau alle Pfade mit einer 1 gekennzeichnet, die auf X warten. In $A_\sim$ ist das genauso, nur dass die Nicht-Repräsentantenpfade fehlen. Gemäß Definition 3.2.7 entsprechen die in dieser Zeile der Matrix $A_\sim$ gekennzeichneten Relationen aber denen von $<_\sim$. Sei Y ein Repräsentantenpfad und somit ein nicht freier Pfad aus $\mathcal{S}$. Dann beschreibt die zugehörige Zeile in $A_\sim$ mit derselben Argumentation wie beim freien Pfad X die auf Y wartenden Elemente aus $\mathcal{S}_\sim$ bezüglich $<_\sim$. Wichtig ist dabei insbesondere, dass die Hauptdiagonale in $A_\sim$ mit Nullen besetzt wurde, wodurch die Irreflexivität von $<_\sim$ in $A_\sim$ beschrieben wird. Weiter beschreibt $A_\sim$ die Relation $<_\sim$ vollständig: Wenn für zwei Pfade X und Y aus $\mathcal{S}_\sim$ gilt $X <_\sim Y$, dann existiert in $A_\sim$ eine zu X gehörige Zeile. Diese ist dann entsprechend der obigen Argumentation an der Stelle von Y mit einer 1 gekennzeichnet, was die Vollständigkeit von $A_\sim$ zeigt.

Der folgende Algorithmus 5 zeigt die Vorgehensweise zur Gewinnung von $<_\sim$ nochmals.

Die Korrektheit des Algorithmus ist nach oben Gesagtem klar, ebenso wie die

Algorithmus 5 : Gewinnung der die Relation $<_\sim$ beschreibenden Matrix $A_\sim$

Eingabe : $n \times n$-Matrix A, die die Relation $<$ zwischen den einzelnen Pfaden eines Systems $\mathcal{S}$ beschreibt

Ausgabe : $n_\sim \times n_\sim$-Matrix $A_\sim$, die die Relation $<_\sim$ innerhalb des zugehörigen Systems $\mathcal{S}$ beschreibt

Beginn
  für alle *nicht freien Pfade y, die keine Repräsentanten sind* **tue**
    Lösche die zu y gehörige Zeile in A;
    Lösche die zu y gehörige Spalte in A;
  Ende
  für alle *Hauptdiagonalelemente* ${}_\sim a_{ii}$ **tue**
    ${}_\sim a_{ii} = 0$;
  Ende
Ende

Terminierung. Nun muss man sich nur noch den Aufwand für seine Ausführung klar machen: Man kann davon ausgehen, dass m nicht freie Pfade ($m \in \mathbb{N}$), wie in Algorithmus 4, weggestrichen werden, da sie keine Repräsentantenpfade sind.

- Um die Zeilen von A zu finden, die gelöscht werden sollen, müssen nur die Listen der Zyklen durchlaufen werden. Diese enthalten insgesamt m' Pfade ($m' \in \mathbb{N}$), wie in Algorithmus 4, so dass ein Aufwand von m' Vergleichen hinzukommt. Hierbei wird der jeweils erste Pfad einer Liste als Repräsentantenpfad gewählt und nicht aus der Matrix gelöscht.

- In den Zeilen und Spalten der Matrix werden dann $mn + m(n-m)$ Einträge gestrichen.

- Dann wird noch in der Hauptdiagonalen jeder Eintrag zu 0 gesetzt, was maximal $n_\sim = n - m$ Änderungen bedeutet.

Man beachte, dass man durch diesen Algorithmus implizit das System $\mathcal{S}_\sim$ gewinnt: Die in der Matrix $A_\sim$ verbleibenden Einträge kennzeichnen genau die Elemente des Systems.

Beispiel 3.2.3. *Die Gewinnung von $<_\sim$ soll an einem Beispiel verdeutlicht werden. Dazu betrachte man das Pfadsystem $\mathcal{S}$ aus Abbildung 3.3 (a). Darin sind nur die Relationen innerhalb des Zyklus schwarz gezeichnet, die Relationen vom freien Pfad X_6 wurden der Übersichtlichkeit halber grau gezeichnet. Die zu $<$ gehörende*

Adjazenzmatrix sieht dann wie folgt aus:

$<$	X_1	X_2	X_3	X_4	X_5	X_6
X_1	*1*	*1*	*1*	*1*	*1*	*0*
X_2	*1*	*1*	*1*	*1*	*1*	*0*
X_3	*1*	*1*	*1*	*1*	*1*	*0*
X_4	*1*	*1*	*1*	*1*	*1*	*0*
X_5	*1*	*1*	*1*	*1*	*1*	*0*
X_6	*1*	*1*	*1*	*1*	*1*	*0*

Nun ist Pfad X_6 frei im System $\mathcal{S}$ und Pfad X_1 wird als Repräsentantenpfad ausgesucht. Gemäß Algorithmus 5 ergibt sich folgende zur Relation $<_\sim$ gehörige Adjazenzmatrix $A_\sim$:

$<_\sim$	X_1	X_6
X_1	*0*	*0*
X_6	*1*	*0*

Diese Matrix gibt nun genau die Beziehungen zwischen den Elementen des zugehörigen Systems $\mathcal{S}_\sim$ wieder, welches in Abbildung 3.3 (b) gezeigt wird.

Die Interpretation dieser Ergebnisse wird in den folgenden Absätzen und Beispielen erläutert.

Wenn man diese Ergebnisse nun interpretiert, so stellt man Folgendes fest: In das System $\mathcal{S}_\sim$ wurden alle freien Pfade und genau ein Repräsentant eines jeden Zyklus des Systems $\mathcal{S}$ übernommen. Dann wurde die Relation $<$ auf $\mathcal{S}_\sim$ übertragen, indem die Relationen zwischen freien Pfaden beibehalten wurden und die Relationen zwischen freien Pfaden und Zyklen bzw. Zyklen untereinander auf die Repräsentantenpfade reduziert wurden. Die Bedeutung der Relation $<$ hat sich dabei auf die Relation $<_\sim$ übertragen. Wenn zwei Pfade in dieser Relation stehen, dann heißt das, dass sie hintereinander ausgeführt werden müssen. Wenn ein Pfad und ein Repräsentantenpfad in Relation stehen, dann heißt das, dass alle Pfade der Äquivalenzklasse des Repräsentantenpfades vor bzw. nach dem Pfad ausgeführt werden müssen. Dieser Sachverhalt wird in Beispiel 3.2.4 nochmals verdeutlicht.

Beispiel 3.2.4. *Hier sollen das Zusammenfassen von Zyklen und die Relation $<_\sim$ verdeutlicht werden. Im Teil (a) der Abbildung 3.3 liegen sechs Pfade eines Systems $\mathcal{S}$ vor, von denen sich fünf in einem Zyklus bezüglich der Relation $<$ befinden. Pfad* 2*, und somit durch Transitivität alle Pfade des Zyklus (gekennzeichnet durch die grauen Pfade), warten auf die Ausführung von Pfad* 6*. Teil (b) zeigt das transfomierte System $\mathcal{S}_\sim$, in dem die fünf Pfade des Zyklus exemplarisch durch Pfad* 1 *repräsentiert werden.*

Auf der Basis dieser Eigenschaften kann man nun folgenden wichtigen Satz formulieren.

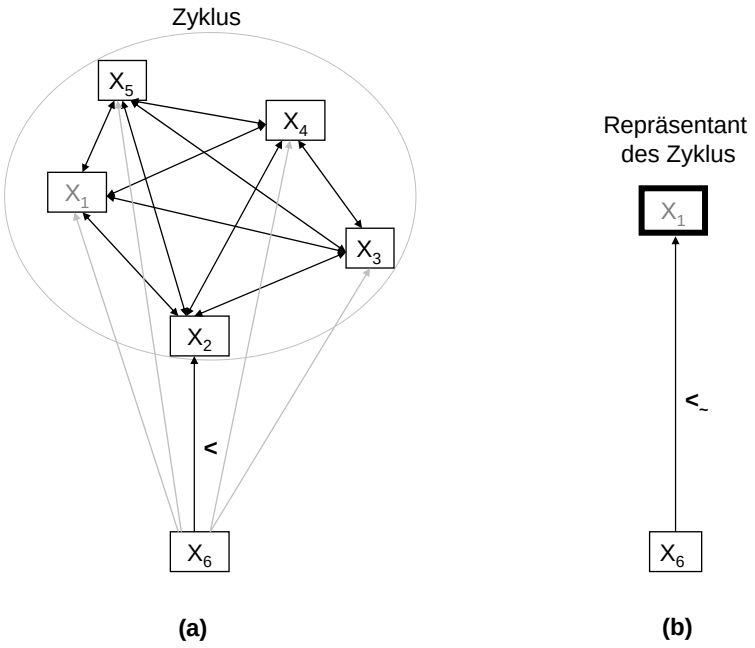


Abbildung 3.3.: (a) System von Pfaden mit Zyklus; (b) zeigt das transformierte System $\mathcal{S}_\sim$

Satz 3.2.2 (Sequentialisieren von $\mathcal{S}_\sim$)**.** *Das Tupel* $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ *sei gegegeben, und* $\mathcal{S}_\sim$ *sei das daraus resultierende System von Pfaden und Repräsentantenpfaden von Äquivalenzklassen gemäß Definition 3.2.6 mit der Relation* $<_\sim$*. Dann ist* $\mathcal{S}_\sim$ *gemäß Algorithmus 2 sequentialisierbar.*

Beweis. Klar, da $<_\sim$ irreflexiv ist. □

Bemerkung 3.2.4. Dieser Satz ist natürlich auch dann richtig, wenn keine Zyklen im System $\mathcal{S}$ existieren. Dann ist $\mathcal{S} = \mathcal{S}_\sim$ und die Relationen $<$ und $<_\sim$ sind identisch.

Was bedeutet aber die Anwendung von Algorithmus 2 auf das System $\mathcal{S}_\sim$? Durch den Algorithmus wird mindestens eine Kette (aus mindestens einem Teilsystem) von Pfaden und Repräsentanten erzeugt. Diese Kette(n) ist/sind total geordnet und genügt/genügen der Relation $<_\sim$. Das bedeutet also, dass zwischen jeweils zwei Elementen einer Kette die Ausführungsreihenfolge festgelegt ist, und man somit eine widerspruchsfreie Ausführungsreihenfolge auf der gesamten Kette erhält, die insbesondere die Relationen $<_{\mathfrak{A}}$ und $<$ beachtet.

Diese Vorgehensweise dient auch wieder der Minimierung der benötigten Anzahl an Ressourcen. Nichtsdestotrotz können alle Pfade, die vom Algorithmus als *sequentiell auszuführen* vorgeschlagen werden, auch *parallel* ausgeführt werden, unter der Voraussetzung, dass genügend Ressourcen vorhanden sind.

Beispiel 3.2.5. *Wenn man Algorithmus 2 auf die Pfade in Teil (b) der Abbildung 3.3 anwendet, so ist das offensichtliche Ergebnis, dass Pfad X_6 vor dem Repräsentantenpfad X_1 ausgeführt werden muss. Wenn also das System $\mathcal{S}$ von Pfaden ausführbar ist, dann sieht man an Teil (a) der Abbildung 3.3, dass Pfad X_2, oder ein Teil davon, erst nach der Abarbeitung von Pfad X_6 ausgeführt werden kann. Also kann, wie durch den Algorithmus 2 vorgeschlagen, auch der Pfad X_6 zuerst komplett abgearbeitet werden, und dann können die Pfade des Zyklus unabhängig von diesem Pfad ausgeführt werden.*

Durch die Relation $<_\sim$ und den Algorithmus 2 wird also nur die Anordnungsstruktur der Pfade offengelegt und gegebenenfalls verstärkt, um eine Totalordnung zu erzeugen. Das einzige, was nun nicht festgelegt wird ist, wie die Ketten innerhalb eines Zyklus ausgeführt werden können. Hier ist es im Extremfall so, dass man für jeden Pfad eines Zyklus eine eigene Ressource benötigt. Daraus ergibt sich ein weiteres wichtiges Resultat, welches durch den folgenden Satz formuliert wird.

Satz 3.2.3 (Minimalanzahl benötigter Ressourcen)**.** *Das Tupel $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ sei gegegeben. Dann entspricht die minimale Anzahl benötigter Ressourcen $R_{\min}$ zur Ausführung der Pfade der Kardinalität des größten Zyklus in $\mathcal{S}$.*

$$R_{\min} = \max_{\zeta(Y) \in \mathcal{S}_\zeta} \|\zeta(Y)\|$$

Beweis. Klar. □

Bemerkung 3.2.5. Satz 3.2.3 ist natürlich nur dann richtig, wenn alle Ressourcen gleich sind und alle diesselben Aufträge bzw. Pfade ausführen können. Ein extremes Gegenbeispiel wären Ressourcen, die auf einen bestimmten Pfad spezialisiert sind. Dann bräuchte man im schlimmsten Fall $\|\mathcal{S}\|$ verschiedene Ressourcen zur Ausführung des Systems $\mathcal{S}$.

3.2.3. Optimierung der Ressourcenanzahl

Bis zu diesem Unterkapitel wurde die Struktur der Pfade eines Systems $\mathcal{S}$ analysiert und Vorschläge gemacht, wie mögliche Ausführungsreihenfolgen berechnet werden können. Diese Untersuchungsmethoden sind rein analytisch und verändern nichts an der Relation $<_{\mathfrak{A}}$. Dies ist auch nicht notwendig, wenn die Pfade sequentiell ausgeführt werden können. Wenn sie aber in einem Zyklus liegen, so muss nach dem bisherigen Stand jeder Pfad auf einer eigenen Ressource ausgeführt werden. Es wurden bis jetzt keine Untersuchungen durchgeführt, wie man hier eine Verringerung beziehungsweise Optimierung der benötigten Ressourcen anstreben kann. In diesem Unterkapitel werden nun Methoden vorgestellt, durch die es ermöglicht werden

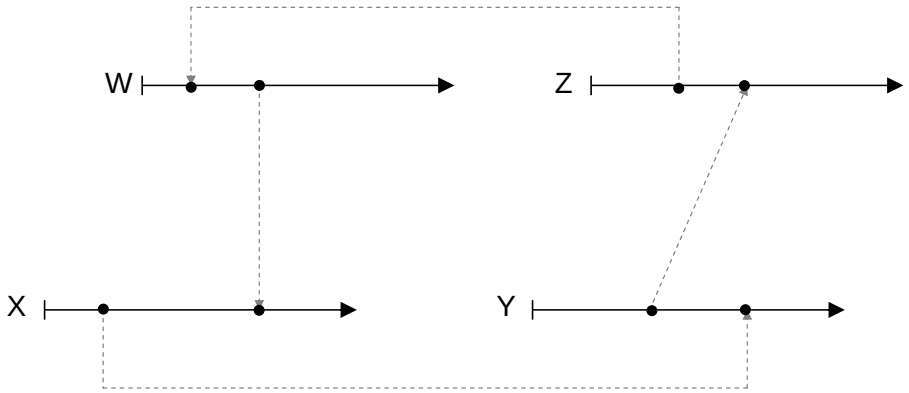


Abbildung 3.4.: Einfacher Zyklus, der aufgelöst werden kann

kann, die zur Ausführung der Pfade eines Systems benötigte Ressourcenanzahl zu optimieren.

Man kann an einem Beispiel leicht sehen, dass es möglich ist, in einem ausführbaren System Pfade von Zyklen (bezüglich der Relation $<$) auszuführen ohne soviel Ressourcen zu benötigen wie Pfade im Zyklus vorhanden sind.

Beispiel 3.2.6. *Abbildung 3.4 zeigt ein Beispiel mit vier Pfaden X, Y, Z und W. Die grau gestrichelten Pfeile deuten die Relation $<_{\mathfrak{A}}$ zwischen Aufträgen auf den Pfaden an, die durch schwarze Punkte skzziert werden. Des einfacheren Verständnisses halber werden die transitiven Relationen zwischen den Aufträgen nicht angegeben. Diese vier Pfade sind ausführbar, denn wenn man die Relation $<_{\mathfrak{A}}$ vervollständigt, dann sieht man, dass sie hier irreflexiv ist. Für die Relation $<$ gilt aber $W < X < Y < Z < W$; die Pfade liegen also in einem Zyklus bezüglich dieser Relation.*

Die Vorgehensweise bis jetzt wäre, dass man jedem Pfad eine eigene Ressource zuordnet. Dies ist allerdings nicht notwendig: Es würde reichen, die Pfade Z, W und X in dieser Reihenfolge auf einer Ressource und den Pfad Y auf einer anderen Ressource auszuführen. Auf der einen Ressource würde Z und auf der anderen Ressource würde Y mit der Ausführung beginnen. Y kann soweit ausgeführt werden, dass Z nun komplett abgearbeitet werden kann. Daraufhin können W und X komplett auf dieser Ressource abgearbeitet werden. Wenn dies geschehen ist, dann kann auch der Pfad Y, dessen Ausführung unterbrochen wurde, beendet werden.

Formal betrachtet ist die vorgeschlagene Ausführungsreihenfolge eine Änderung der Relation $<_{\mathfrak{A}}$. Wenn mit z_E der letzte Auftrag des Pfades Z, mit w_a und w_E der erste und der letzte Auftrag des Pfades W und mit x_A und x_E der erste und der letzte Auftrag des Pfades X bezeichnet wird, dann bedeutet die oben vorgeschlagene Ausführungsreihenfolge die folgenden Verstärkungen[1] der Relation $<_{\mathfrak{A}}$:

[1]Verstärkung heißt: Es werden Relationen zwischen Aufträgen eingeführt, die vorher noch nicht

- $z_E <_{\mathfrak{A}} w_A$, *woraus aufgrund der Transitivität der Relation folgt, dass jeder Auftrag des Pfades* W *nach* z_E *ausgeführt werden muss. Natürlich ist darin die bereits vorhandene Relation zwischen den beiden Pfaden nun schon enthalten.*

- $w_E <_{\mathfrak{A}} x_A$, *woraus aufgrund der Transitivität der Relation folgt, dass jeder Auftrag des Pfades* X *nach* w_E *ausgeführt werden muss. Auch hier wird die bereits vorhandene Relation zwischen den beiden Pfaden durch die neu eingeführten Relationen überdeckt. Interessant ist aber, dass nun auch jeder Auftrag des Pfades* X *nach* z_E *und somit nach allen Aufträgen des Pfades* X *ausgeführt werden muss.*

Man sieht also, dass es durch die vorgeschlagenen „Anordnungen" der Pfade in dem Zyklus möglich geworden ist, ein neues gültiges Schedule zu finden und zusätzlich die Anzahl der benötigten Ressourcen von vier auf zwei zu senken.

Die Frage, die sich aus diesem Beispiel ergibt ist, ob man diese Anordnung von Pfaden zur Minimierung der benötigten Anzahl von Ressourcen in einem Zyklus verallgemeinern kann und wenn ja, wie sich dies dann auf die Relationen $<_{\mathfrak{A}}$, $<$ und $<_{\sim}$ auswirken würde?

Definition 3.2.8 (2-Anordnung von Pfaden)**.** *Das Tupel* $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ *sei gegeben und* X *und* Y *seien zwei Pfade aus* $\mathcal{S}$*. Sei* x_E *der letzte Auftrag von* X *und* y_A *der erste Auftrag von* Y *(jeweils im Sinne von* $<_{\mathfrak{A}}$*). Dann ist* $<_{\mathfrak{A}^*}$ *die Relation auf* $\mathcal{S}_{\mathfrak{A}}$ *mit den Eigenschaften von Definition 2.2.2, die die Relation* $<_{\mathfrak{A}}$ *enthält und für die gilt* $x_E <_{\mathfrak{A}^*} y_A$*.*

Das Tupel $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}^*}, <^*)$ *ist dann das resultierende System gemäß Definition 2.3.5.*

Bemerkung 3.2.6. Die 2-Anordnung von Pfaden bedeutet also, dass zusätzlich zur vorhandenen Relation $<_{\mathfrak{A}}$ eine Abhängigkeit zwischen den Aufträgen x_E und y_A definiert wird. In der resultierenden Relation $<_{\mathfrak{A}^*}$ ist dann genau die alte Relation zusammen der Relation zwischen x_E und y_A und den sich daraus ergebenden transitiven Relationen enthalten.

Bemerkung 3.2.7. Die Existenz und Eindeutigkeit der Relation $<_{\mathfrak{A}^*}$ auf $\mathcal{S}_{\mathfrak{A}}$ sind klar. Die Relation $<_{\mathfrak{A}^*}$ ist offensichtlich transitiv; die Irreflexivität ist allerdings keinesfalls sichergestellt, vergleiche Beispiel 3.2.7. Falls bereits $x_E <_{\mathfrak{A}} y_A$ galt, so wird durch solch eine Anordnung nichts bewirkt.

Wenn man $<_{\mathfrak{A}}$ und $<_{\mathfrak{A}^*}$ als Mengen von Tupeln jeweils in Relation zueinander stehender Aufträge auffasst, so gilt für ein Tupel $(u,\ w) \in <_{\mathfrak{A}^*} \setminus <_{\mathfrak{A}}$, welches bezüglich $<_{\mathfrak{A}^*}$, aber nicht bezüglich $<_{\mathfrak{A}}$ in Relation steht: Entweder $u = x_E$ und $w = y_A$ oder es gibt Folgen von Aufträgen $z_1,\ ...,\ z_n$ und $v_1,\ ...,\ v_m$ (mit $\binom{m}{n} \in \mathbb{N}_0$) mit der

existierten.

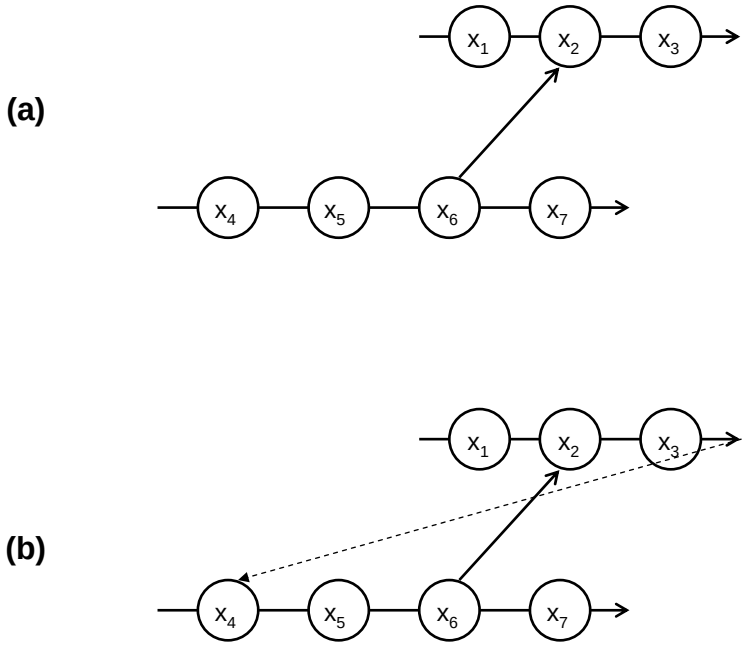


Abbildung 3.5.: Teil (a) zeigt zwei Pfade, die ausführbar sind; Teil (b) zeigt dieselben Pfade, die wegen der Anordnung nicht mehr ausführbar sind

Eigenschaft $u <_{\mathfrak{A}} z_1 <_{\mathfrak{A}} ... <_{\mathfrak{A}} z_n <_{\mathfrak{A}} x_E <_{\mathfrak{A}^*} y_A <_{\mathfrak{A}} v_1 <_{\mathfrak{A}} ... <_{\mathfrak{A}} v_m <_{\mathfrak{A}} w$. Mit anderen Worten: Jedes Tupel aus $<_{\mathfrak{A}^*} \setminus <_{\mathfrak{A}}$ lässt sich genau, aber nicht unbedingt eindeutig als Folge von Tupeln aus $<_{\mathfrak{A}}$ und diesem einen neuen Tupel $(x_E,\ y_A)$ darstellen. Dieser wichtige Sachverhalt liegt an der Forderung der Minimalität der Relation $<_{\mathfrak{A}^*}$ und wird später genutzt.

Beispiel 3.2.7. *Teil (a) der Abbildung 3.5 zeigt zwei Pfade, die offensichtlich ausführbar sind (die Relationen zwischen Aufträgen aufgrund Transitivität wurden der besseren Übersicht halber weggelassen). Die beiden Pfade können auf einer Ressource ausgeführt werden; dann muss der Pfad mit dem Auftrag x_4 zuerst bearbeitet werden, während der andere Pfad als zweites bearbeitet wird. Die beiden Pfade können auch auf zwei verschiedenen Ressourcen ausgeführt werden.*

Teil (b) zeigt dieselben Pfade, nur wurde die Anordnung $x_3 <_{\mathfrak{A}^*} x_4$ vorgenommen. Diese Pfade sind offenbar nicht mehr ausführbar.

Definition 3.2.9 (Anordnung von Pfaden)**.** *Das Tupel $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ sei gegeben und seien X^1, ..., X^k k Pfade aus $\mathcal{S}$ ($k \geq 2$, k gerade). Weiter seien x^i_A und x^i_E der erste bzw. der letzte Auftrag des Pfades X^i ($1 \leq i \leq k$). Dann ist $<_{\mathfrak{A}^*}$ die kleinste Relation auf $\mathcal{S}_{\mathfrak{A}}$ mit den Eigenschaften von Definition 2.2.2, die die Relation $<_{\mathfrak{A}}$ enthält und für die gilt $x_E^{2i-1} <_{\mathfrak{A}^*} x_A^{2i}$ ($1 \leq i \leq \frac{k}{2}$).*

Das Tupel $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}^*}, <^*)$ ist dann das resultierende System gemäß Definition

2.3.5.

Bemerkung 3.2.8. Die Anordnung von Pfaden bedeutet also, dass zusätzlich zur vorhandenen Relation $<_{\mathfrak{A}}$ paarweise Abhängigkeiten zwischen den Aufträgen x_E^{2i-1} und x_A^{2i} definiert werden. In der resultierenden Relation $<_{\mathfrak{A}^*}$ ist dann genau die alte Relation zusammen der Relation zwischen x_E^{2i-1} und x_A^{2i} und den sich daraus ergebenden transitiven Relationen enthalten.

Als Notation für solch eine Anordnung wird im Folgenden immer $A^{x^1\,x^3\ldots x^{k-1}}_{x^2\,x^4\ldots x^k}$ benutzt. Dabei ist der obere Auftrag immer derjenige Auftrag, auf den der direkt darunter stehende Auftrag wartet. Hier wartet also x_2 auf x_1 und so weiter.

Bemerkung 3.2.9. Auch bei einer Anordnung von Pfaden ist die *Existenz* und *Eindeutigkeit* der Relation $<_{\mathfrak{A}^*}$ klar. Genauso gilt definitionsgemäß die *Transitivität*. Da die 2-Anordnung ein Spezialfall der Anordnung ist, kann die Irreflexivität im Allgemeinen *nicht* angenommen werden.

Wenn man $<_{\mathfrak{A}}$ und $<_{\mathfrak{A}^*}$ wieder als Mengen von Tupeln jeweils in Relation zueinander stehender Aufträge auffasst, so gilt genau wie bei der 2-Anordnung von Pfaden, dass sich jedes Tupel aus $<_{\mathfrak{A}^*} \setminus <_{\mathfrak{A}}$ genau als Folge von Tupeln aus $<_{\mathfrak{A}}$ und den neuen Tupeln $x_E^{2i-1} <_{\mathfrak{A}^*} x_A^{2i}$ $(1 \leq i \leq \frac{k}{2})$ darstellen lässt.

Durch die Forderung, dass k gerade sein muss, wird keine Einschränkung der Anordnungen erzeugt. Man kann sich einfach klar machen, das eine Anordnung einer beliebigen Anzahl von Pfaden mit dieser Definition möglich ist, da sie nicht fordert, dass die k Pfade verschieden sein sollen.

Die *Gewinnung* von $<_{\mathfrak{A}^*}$ auf den Aufträgen eines Systems $\mathcal{S}$ ist leicht. Man nehme die $k \times k$-Matrix $\mathfrak{A}$, die die Relation $<_{\mathfrak{A}}$ auf den Aufträgen beschreibt, und füge an den Stellen eine Eins ein, an denen eine neue Abhängigkeit durch die Anordnung $A^{x^1\,x^3\ldots x^{k-1}}_{x^2\,x^4\ldots x^k}$ gefordert wird. Auf der so modifizierten Matrix berechne man die transitive Hülle und erhält eine Matrix $\mathfrak{A}^*$, die genau die Relation $<_{\mathfrak{A}^*}$ beschreibt: Es ist klar, dass je zwei Aufträge, die in $\mathfrak{A}^*$ mit einer Eins gekennzeichnet sind, durch $<_{\mathfrak{A}^*}$ in Beziehung stehen aufgrund Konstruktion von $\mathfrak{A}^*$. Wenn umgekehrt zwei Aufträge mittels $<_{\mathfrak{A}^*}$ in Beziehung stehen, so standen sie bereits in $<_{\mathfrak{A}}$ in Beziehung und tun dies aufgrund Konstruktion auch in $\mathfrak{A}^*$. Ansonsten wurde dies durch die Anordnung von zwei oder mehr Pfaden von $\mathcal{S}$ und der Transitivität von $<_{\mathfrak{A}^*}$ verursacht. Dies gilt wegen der Forderung nach der Minimalität der Relation $<_{\mathfrak{A}^*}$. Daher ist auch diese Beziehung aufgrund Konstruktion in $\mathfrak{A}^*$ enthalten. Der folgende Algorithmus 6 beschreibt die Vorgehensweise noch einmal:
Die *Korrektheit* des Algorithmus ist nach dem bereits Gesagten klar. Die *Terminierung* folgt wegen der Terminierung der einzelnen Operationen im Algorithmus. Nun ist nur noch der *Aufwand* für den Algorithmus interessant:

- Wenn p Anordnungen durchgeführt werden, müssen p Stellen in $\mathfrak{A}$ modifiziert werden.

Algorithmus 6 : Berechnung der Relation $<_{\mathfrak{A}^*}$ auf Basis einer Anordnung

Eingabe : $k \times k$-Matrix $\mathfrak{A}$, welche die Relation $<_{\mathfrak{A}}$ beschreibt
Ausgabe : $k \times k$-Matrix $\mathfrak{A}^*$, welche die Relation $<_{\mathfrak{A}^*}$ beschreibt
Beginn
  Setze Einsen in $\mathfrak{A}$ an den Stellen, die gemäß der Anordnung $A^{x^1\,x^3\ldots x^{k-1}}_{x^2\,x^4\ldots x^k}$ in Beziehung stehen sollen;
  $\mathfrak{A}^*$ = Transitive Hülle der modifizierten Matrix;
Ende

- Das Berechnen der transitiven Hülle kostet maximal k^2 Vergleichoperationen und k^3 arithmetische Operationen.

Bemerkung 3.2.10. Um die Notation nicht unnötig kompliziert zu gestalten, wird nach einer beliebigen Anordnung von Pfaden weiterhin die Notation $<_{\mathfrak{A}}$ beziehungsweise $<$ verwendet, wenn eine Unterscheidung nicht unbedingt notwendig ist.

Satz 3.2.4 (Kommutativität von Anordnungen)**.** *Seien* $A^{x^1\,x^3\ldots x^{k_1-1}}_{x^2\,x^4\ldots x^{k_1}}$ *und* $A^{y^1\,y^3\ldots y^{k_2-1}}_{y^2\,y^4\ldots y^{k_2}}$ *zwei Anordnungen wie in Definition 3.2.9. Dann kommutieren die Anordnungen, das heißt es gilt:*

$$A^{x^1\,x^3\ldots x^{k_1-1}}_{x^2\,x^4\ldots x^{k_1}} \circ A^{y^1\,y^3\ldots y^{k_2-1}}_{y^2\,y^4\ldots y^{k_2}} = A^{y^1\,y^3\ldots y^{k_2-1}}_{y^2\,y^4\ldots y^{k_2}} \circ A^{x^1\,x^3\ldots x^{k_1-1}}_{x^2\,x^4\ldots x^{k_1}}$$

Bemerkung 3.2.11. Mit $\circ$ ist die Hintereinanderanwendung der Anordnungen auf das System $\mathcal{S}$ gemeint.

Beweis. Um die Gleichheit zu zeigen, werden die Relationen wieder als Mengen von Tupeln aufgefasst und die Gleichheit dieser Mengen gezeigt.
„$\subseteq$“: Sei $(x, y) \in A^{x^1\,x^3\ldots x^{k_1-1}}_{x^2\,x^4\ldots x^{k_1}} \circ A^{y^1\,y^3\ldots y^{k_2-1}}_{y^2\,y^4\ldots y^{k_2}}$. Dieses Tupel kann man aufgrund der Transitivität der Relation auf den Aufträgen als Folge von Tupeln der ursprünglichen Relation $<_{\mathfrak{A}}$ und $x_E^{2i-1} <_{\mathfrak{A}^*} x_A^{2i}$ $(1 \leq i \leq \frac{k_1}{2})$ und $y_E^{2j-1} <_{\mathfrak{A}^*} y_A^{2j}$ $(1 \leq j \leq \frac{k_2}{2})$ darstellen. Dies liegt wieder an der Minimalitätsforderung in der Definition der Relation $<_{\mathfrak{A}}$. Damit gilt aber auch $(x, y) \in A^{y^1\,y^3\ldots y^{k_2-1}}_{y^2\,y^4\ldots y^{k_2}} \circ A^{x^1\,x^3\ldots x^{k_1-1}}_{x^2\,x^4\ldots x^{k_1}}$.
„$\supseteq$“: Analog zu „$\subseteq$“. □

Bemerkung 3.2.12. Das Resultat dieses Satzes ist, dass man Anordnungen beliebig umsortieren kann, die dabei entstehende Relation ist gleich.

Man mache sich auch klar, dass die Pfade, die angeordnet werden sollen, auch gleich sein können; es ist nicht gefordert, dass sie paarweise verschieden sind.

Beispiel 3.2.8. *Gegeben seien drei Pfade* X, Y *und* Z *bestehend aus den Aufträgen* x_1, x_2, x_3 *beziehungsweise* y_1, y_2, y_3 *und* z_1, z_2, z_3. *Diese stehen in einer zeitlichen*

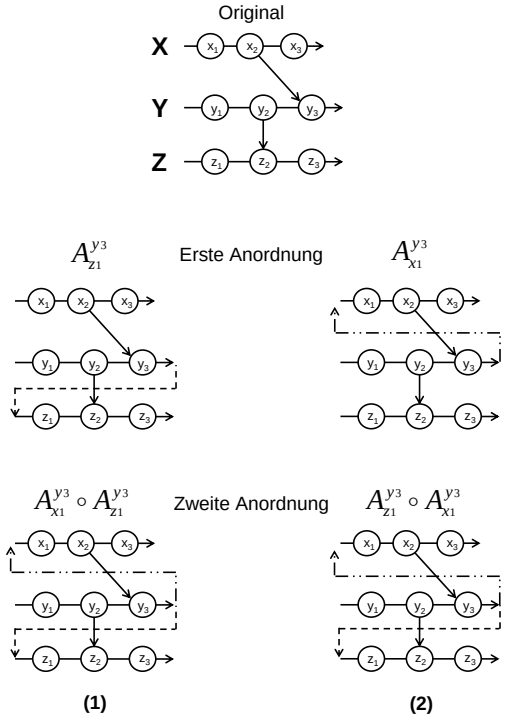


Abbildung 3.6.: Kommutativität der Anordnung

Abhängigkeit wie in Abbildung 3.6 gezeigt. Der Übersichtlichkeit halber wurde in dieser Abbildung auf die Abhängigkeiten, die sich durch Transitivität ergeben, verzichtet; es wird also nicht die komplette Relation $<_{\mathfrak{A}}$ dargestellt. Die folgende Adjazenzmatrix zeigt nun aber die vollständige Relation $<_{\mathfrak{A}}$ auf dem Pfadsystem $\{X, Y, Z\}$.

	x_1	x_2	x_3	y_1	y_2	y_3	z_1	z_2	z_3
x_1	0	1	1	0	0	1	0	0	0
x_2	0	0	1	0	0	1	0	0	0
x_3	0	0	0	0	0	0	0	0	0
y_1	0	0	0	0	1	1	0	1	1
y_2	0	0	0	0	0	1	0	1	1
y_3	0	0	0	0	0	0	0	0	0
z_1	0	0	0	0	0	0	0	1	1
z_2	0	0	0	0	0	0	0	0	1
z_3	0	0	0	0	0	0	0	0	0

Dann wird die Verknüpfung der Anordnungen $A^{y_3}_{z_1}$ und $A^{y_3}_{x_1}$ betrachtet. Offensichtlich wird durch die Anordnung $A^{y_3}_{x_1}$ ein Zyklus auf den Aufträgen eingeführt; das System von Pfaden ist dann zyklusbehaftet und nicht ausführbar. Dies ist in diesem Beispiel beabsichtigt und soll exemplarisch zeigen, dass die Kommutativität der Anordnung auch in solch einem Fall gilt.

$\mathbf{A^{y_3}_{x_1} \circ A^{y_3}_{z_1}}$*: Im Zweig* (1) *der Abbildung wird zunächst die Anordnung* $A^{y_3}_{z_1}$ *betrachtet, und es ergibt sich folgende Adjazenzmatrix der Relation* $<_{\mathfrak{A}}$ *gemäß Definition 3.2.8 beziehungsweise Definition 3.2.9 unter dieser Anordnung.*

$A^{y_3}_{z_1}$	x_1	x_2	x_3	y_1	y_2	y_3	z_1	z_2	z_3
x_1	*0*	*1*	*1*	*0*	*0*	*1*	*1*	*1*	*1*
x_2	*0*	*0*	*1*	*0*	*0*	*1*	*1*	*1*	*1*
x_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
y_1	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*
y_2	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*
y_3	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*
z_1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*
z_2	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*
z_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Man kann leicht nachrechnen, dass durch die Adjazenzmatrix die kleinste Relation beschrieben wird, die die „alte" Relation $<_{\mathfrak{A}}$ *und das Tupel* $y_3 <_{\mathfrak{A}} z_1$ *enthält.*

Nun wird auf dieses neue System von Pfaden die Anordnung $A^{y_3}_{x_1}$ *angewendet. Dies entspricht der Anwendung der Verkettung* $A^{y_3}_{x_1} \circ A^{y_3}_{z_1}$ *auf das originale System von Pfaden. Wie oben ergibt sich folgende Adjazenzmatrix der Relation* $<_{\mathfrak{A}}$ *gemäß Definition 3.2.8 beziehungsweise Definition 3.2.9 unter dieser Verkettung der Anordnungen.*

$A^{y_3}_{x_1} \circ A^{y_3}_{z_1}$	x_1	x_2	x_3	y_1	y_2	y_3	z_1	z_2	z_3
x_1	*1*	*1*	*1*	*0*	*0*	*1*	*1*	*1*	*1*
x_2	*1*	*1*	*1*	*0*	*0*	*1*	*1*	*1*	*1*
x_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
y_1	*1*	*1*	*1*	*0*	*1*	*1*	*1*	*1*	*1*
y_2	*1*	*1*	*1*	*0*	*0*	*1*	*1*	*1*	*1*
y_3	*1*	*1*	*1*	*0*	*0*	*1*	*1*	*1*	*1*
z_1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*
z_2	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*
z_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

An den Einsen in der Hauptdiagonalen dieser Matrix sieht man, dass das System der Pfade durch die Anordnung $A^{y_3}_{x_1}$ *nun zyklusbehaftet ist. Auch hier kann man leicht nachrechnen, dass durch die Adjazenzmatrix die kleinste Relation beschrieben wird, die die „alte" Relation* $<_{\mathfrak{A}}$ *und das Tupel* $y_3 <_{\mathfrak{A}} x_1$ *enthält.*

$\mathbf{A^{y_3}_{z_1} \circ A^{y_3}_{x_1}}$*: In Zweig* (2) *der Abbildung 3.6 wird die umgekehrte Reihenfolge der Anordnungen untersucht. Zunächst wird die Anordnung* $A^{y_3}_{x_1}$ *auf das originale System*

von Pfaden angewendet. Dabei ergibt sich folgende Adjazenzmatrix der Relation $<_{\mathfrak{A}}$.

$A_{x_1}^{y_3}$	x_1	x_2	x_3	y_1	y_2	y_3	z_1	z_2	z_3
x_1	1	1	1	0	0	1	0	0	0
x_2	1	1	1	0	0	1	0	0	0
x_3	0	0	0	0	0	0	0	0	0
y_1	1	1	1	0	1	1	0	1	1
y_2	1	1	1	0	0	1	0	1	1
y_3	1	1	1	0	0	1	0	0	0
z_1	0	0	0	0	0	0	0	1	1
z_2	0	0	0	0	0	0	0	0	1
z_3	0	0	0	0	0	0	0	0	0

Auch hier sieht man an den Einsen in der Hautdiagonalen der Matrix, dass das so angeordnete System von Pfaden zyklusbehaftet ist.

Eine Anwendung der Anordnung $A_{z_1}^{y_3}$ *auf dieses neue System von Pfaden entspricht der Hintereinanderausführung der Anordnungen* $A_{z_1}^{y_3} \circ A_{x_1}^{y_3}$ *auf das originale System von Pfaden. Dabei ergibt sich folgende Adjazenzmatrix der Relation* $<_{\mathfrak{A}}$.

$A_{z_1}^{y_3} \circ A_{x_1}^{y_3}$	x_1	x_2	x_3	y_1	y_2	y_3	z_1	z_2	z_3
x_1	1	1	1	0	0	1	1	1	1
x_2	1	1	1	0	0	1	1	1	1
x_3	0	0	0	0	0	0	0	0	0
y_1	1	1	1	0	1	1	1	1	1
y_2	1	1	1	0	0	1	1	1	1
y_3	1	1	1	0	0	1	1	1	1
z_1	0	0	0	0	0	0	0	1	1
z_2	0	0	0	0	0	0	0	0	1
z_3	0	0	0	0	0	0	0	0	0

Nun sieht man, dass die Adjazenzmatrizen zu $A_{x_1}^{y_3} \circ A_{z_1}^{y_3}$ *und* $A_{z_1}^{y_3} \circ A_{x_1}^{y_3}$ *gleich sind, dass heißt es gilt* $A_{x_1}^{y_3} \circ A_{z_1}^{y_3} = A_{z_1}^{y_3} \circ A_{x_1}^{y_3}$*, wie in Satz 3.2.4 vorhergesagt.*

Satz 3.2.5. *Sei* $\mathcal{S}$ *ein System von Pfaden und* X^1, ..., X^k k *Pfade aus* $\mathcal{S}$ *(*$k \geq 2$*,* k *gerade). Seien* $A_{x^2\,x^4\ldots x^j}^{x^1\,x^3\ldots x^{j-1}}$, $A_{x^{j+2}\,x^{j+4}\ldots x^k}^{x^{j+1}\,x^{j+3}\ldots x^{k-1}}$ *und* $A_{x^2\,x^4\ldots x^k}^{x^1\,x^3\ldots x^{k-1}}$ *Anordnungen wie in Definition 3.2.9. Dann gilt für ein gerades* j *mit* $0 \leq j \leq k$*:*

$$A_{x^2\,x^4\ldots x^j}^{x^1\,x^3\ldots x^{j-1}} \circ A_{x^{j+2}\,x^{j+4}\ldots x^k}^{x^{j+1}\,x^{j+3}\ldots x^{k-1}} = A_{x^2\,x^4\ldots x^k}^{x^1\,x^3\ldots x^{k-1}} \tag{3.4}$$

Beweis. Kann analog zu dem Beweis aus Satz 3.2.4 geführt werden. □

Bemerkung 3.2.13. Da die Anordnungen kommutativ sind bezüglich der Reihenfolge, in der sie eingeführt werden, kann auf der linken Seite der Gleichung (3.4) auch

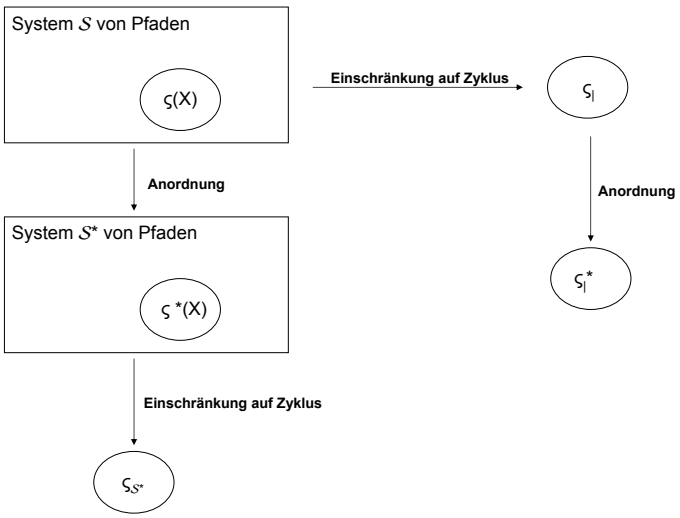


Abbildung 3.7.: Informative Darstellung der Begriffe aus Satz 3.2.6

eine beliebige andere Reihenfolge der Hintereinanderanwendung der Anordnungen gewählt werden.

Dass die „Reihenfolge“ bei der Anordnung beliebig ist, ist klar aufgrund ihrer Definition. Dies wird aber durch diesen Satz und die Kommutativität der Anordnungen nochmals bestätigt.

Ein Beispiel zu Satz 3.2.5 kann man sich ganz ähnlich zu Beispiel 3.2.8 überlegen; da aber im Wesentlichen dieselben Argumentationen und Rechenschritte verwendet werden können, wird hier darauf verzichtet.

Eine Anordnung von Pfaden, wie sie in den Definitionen 3.2.8 bzw. 3.2.9 eingeführt wurde, sollte nicht auf sequentiell ausführbare Pfade angewendet werden, da sie hier Freiheitgrade in der Bearbeitung einschränken würde. Die Anordnung soll vielmehr auf Pfade innerhalb von Zyklen angewendet werden, um eine Verringerung der benötigten Ressourcen zur Ausführung der Zyklen zu erreichen.

Eine wesentliche Hilfe auf dem Weg dorthin ist der folgende Satz 3.2.6. Wegen seiner komplizierten Notation werden die wichtigsten Begriffe daraus in Abbildung 3.7 informativ dargestellt.

Satz 3.2.6. *Sei $\mathcal{S}$ ein System von Pfaden, X ein nicht freier Pfad aus $\mathcal{S}$ und $\zeta(X)$ der zugehörige Zyklus. Weiter sei $\zeta_|$ die Einschränkung auf das System der Pfade dieses Zyklus. Sei $\zeta_|^*$ das aus einer Anordnung von Pfaden aus $\zeta_|$ entstehende System und $\mathcal{S}^*$ das aus einer Anwendung der Anordnung auf $\mathcal{S}$ entstehende System. Dann gilt:*

1. *Die Relation $<$ ist auf $\mathcal{S}$ und $\mathcal{S}^*$ identisch.*

2. *Sei $\zeta_{\mathcal{S}^*}$ die Einschränkung von $\mathcal{S}^*$ auf die Pfade von $\zeta^*_|$. Dann gilt für die Aufträge je zweier Pfade Y und W aus $\zeta_{\mathcal{S}^*}$ bzw. $\zeta^*_|$:*

$$y_i <_{\mathfrak{A}} w_j \text{ in } \zeta_{\mathcal{S}^*} \Leftrightarrow y_i <_{\mathfrak{A}} w_j \text{ in } \zeta^*_|$$

Dabei sind y_i und w_j Aufträge der Pfade Y und W.

3. *Sei R ein nicht freier Pfad aus $\mathcal{S}$ mit $R \notin \zeta(X)$. Weiter sei $\zeta_|(R)$ die Einschränkung von $\mathcal{S}$ auf $\zeta(R)$ und $\zeta^*_|(R)$ die Einschränkung von $\mathcal{S}^*$ auf $\zeta(R)$. Dann gilt für die Aufträge je zweier Pfade Y und W aus $\zeta^*_|(R)$ bzw. $\zeta_|(R)$:*

$$y_i <_{\mathfrak{A}} w_j \text{ in } \zeta^*_|(R) \Leftrightarrow y_i <_{\mathfrak{A}} w_j \text{ in } \zeta_|(R)$$

Beweis.
Zu 1.: Es ist klar, dass für alle Pfade T und U mit $T < U$ in $\mathcal{S}$ gilt, dass auch $T < U$ in $\mathcal{S}^*$ gilt. Seien nun T und U Pfade mit $T < U$ in $\mathcal{S}^*$ und wieder X^1, ..., X^k die angeordneten Pfade mit $x_E^{2i-1} <_{\mathfrak{A}^*} x_A^{2i}$ $(1 \leq i \leq \frac{k}{2})$. Da bereits vor dem Anordnen der Pfade $X^{2i-1} < X^{2i}$ $(1 \leq i \leq \frac{k}{2})$ galt kann dadurch keine neue Relation bezüglich $<$ auf $\mathcal{S}^*$ erzeugt werden: Durch die Relation $<_{\mathfrak{A}}$ und Transitivität darauf kann keine neue Pfad-Relation $<$ erzeugt werden, da sie bereits in $\mathcal{S}$ durch $X^{2i-1} < X^{2i}$ $(1 \leq i \leq \frac{k}{2})$ und der Transitivität von $<$ erfasst ist. Somit kann durch Transitivität auf $<$ auch hier keine neuen Relationen erzeugt werden.

Zu 2.:
„⇒“: Da sich nach Punkt 1 die Relation $<$ nicht geändert hat enthalten $\zeta(X)$, $\zeta^*_|$ und $\zeta_{\mathcal{S}^*}$ genau dieselben Pfade. Annahme $y_i <_{\mathfrak{A}} w_j$ gilt nicht in $\zeta^*_|$. Dann ist $y_i <_{\mathfrak{A}} w_j$ in $\mathcal{S}^*$ durch Transitivität mit einem Auftrag t_m eines Pfades T entstanden, welcher kein Pfad des Zyklus ist. In $\mathcal{S}^*$ gelte also o.B.d.A. $y_i <_{\mathfrak{A}} t_m <_{\mathfrak{A}} w_j$. Dann gilt für die Pfade in $\mathcal{S}^*$ aber $Y < T$ und $T < W$. Nach Punkt 1 dieses Satzes ist die Relation $<$ aber auf $\mathcal{S}$ und $\mathcal{S}^*$ identisch. Dann kann $Y < T$ und $T < W$ nicht gelten, da sonst aufgrund der Definition eines Zyklus $T \in \zeta_{\mathcal{S}^*}$ gelten würde im Widerspruch zur Voraussetzung. Also ist die Annahme falsch.
„⇐“: Klar.

Zu 3.:
„⇒“: Fast wörtlich wie der Beweis zu 2.: Da sich nach Punkt 1 die Relation $<$ nicht geändert hat enthalten $\zeta(R)$, $\zeta_|(R)$ und $\zeta^*_|(R)$ genau dieselben Pfade. Annahme $y_i <_{\mathfrak{A}} w_j$ gilt nicht in $\zeta^*_|(R)$. Dann ist $y_i <_{\mathfrak{A}} w_j$ in $\mathcal{S}^*$ durch Transitivität mit einem Auftrag t_m eines Pfades T entstanden, welcher kein Pfad des Zyklus ist. In $\mathcal{S}^*$ gelte also o.B.d.A. $y_i <_{\mathfrak{A}} t_m <_{\mathfrak{A}} w_j$. Dann gilt für die Pfade in $\mathcal{S}^*$ aber $Y < T$ und $T < W$. Nach Punkt 1 dieses Satzes ist die Relation $<$ aber auf $\mathcal{S}$ und $\mathcal{S}^*$ identisch. Dann kann $Y < T$ und $T < W$ nicht gelten, da sonst aufgrund der Definition eines Zyklus

$T \in \zeta_{\mathcal{S}^*}$ gelten würde im Widerspruch zur Voraussetzung. Also ist die Annahme falsch.
„⇐“: Klar. □

Bemerkung 3.2.14. Zunächst gilt für alle Aussagen dieses Satzes, dass sie unabhängig davon sind, ob das System $\mathcal{S}$ ausführbar ist oder nicht. Dies liegt daran, dass die Anordnung innerhalb eines Zyklus vorgenommen wird.

Punkt 1 dieses Satzes zeigt, dass sich durch das Anordnen von Pfaden im Zyklus die Relation $<$ nicht ändert, was für einen späteren Satz eine wesentliche Voraussetzung sein wird. Insbesondere folgt aus Punkt 1: Zwei Pfade sind in $\mathcal{S}$ parallel ausführbar genau dann wenn sie in $\mathcal{S}^*$ parallel ausführbar sind. Das bedeutet, dass man Algorithmus 3 (Gewinnung der parallel ausführbaren Pfade) nicht bei jeder solchen Anordnung neu berechnen muss, sondern es reicht, ihn ein einziges Mal durchzuführen.

Punkt 2 zeigt, dass das Anordnen in einem Zyklus zwar die Relation $<_{\mathfrak{A}}$ in diesem ändert, aber dass die Aufträge außerhalb des Zyklus keinen Einfluss auf die Beziehungen innerhalb des Zyklus haben. Man kann Punkt 2 auch so interpretieren: Es macht keinen Unterschied bezüglich der Relation $<_{\mathfrak{A}}$ zwischen Aufträgen eines Zyklus, ob man zunächst seine Einschränkung betrachtet und diese anordnet oder ob man zunächst das gesamte System $\mathcal{S}$ anordnet und dann die Einschränkung auf den Zyklus betrachtet. Dieser Sachverhalt wird durch das folgende kommutierende Diagramm verdeutlicht:

$$\begin{array}{ccc} \text{System } \mathcal{S} \text{ mit } <_{\mathfrak{A}} & \xrightarrow{\text{Anordnung von Pfaden (in } \zeta(X))} & \mathcal{S}^* \\ \Big\downarrow \text{Einschränkung auf } \zeta(X) & & \Big\downarrow \text{Einschränkung auf } \zeta(X) \\ \zeta_| & \xrightarrow[\text{Anordnung von Pfaden}]{} & \zeta_|^* \end{array}$$

Punkt 3 zeigt schließlich, dass eine Anordnung auf einem Zyklus keinen Einfluss auf die Relation $<_{\mathfrak{A}}$ innerhalb eines anderen Zyklus hat. Das Anordnen innerhalb eines Zyklus verstärkt somit nur die *lineare Struktur* der Relation $<_{\mathfrak{A}}$. Die Verstärkung wird am Ende dieses Unterkapitels genauer charakterisiert.

Satz 3.2.7 (Ausführbare Anordnung von Pfaden innerhalb eines Zyklus)**.** *Sei $\mathcal{S}$ ein ausführbares System von Pfaden, X ein nicht freier Pfad aus $\mathcal{S}$ und $\zeta(X)$ der zugehörige Zyklus. Weiter sei $\zeta_|$ die Einschränkung auf die Pfade dieses Zyklus.*

Sei $\zeta_|^*$ das aus einer Anordnung von Pfaden aus $\zeta_|$ entstehende ausführbare *System. Dann gilt: Das aus der Anwendung der Anordnung auf $\mathcal{S}$ resultierende System $\mathcal{S}^*$ ist ausführbar.*

Beweis.
Nach Satz 3.2.6 Punkt 1 ist die Relation $<$ auf $\mathcal{S}$ und $\mathcal{S}^*$ gleich. Sei $\zeta_{\mathcal{S}^*}$ die Einschränkung von $\mathcal{S}^*$ auf die Pfade von $\zeta_|^*$. Das bedeutet, dass $\zeta(X)$, $\zeta_|^*$ und $\zeta_{\mathcal{S}^*}$ Zyklen

sind und genau diesselben Pfade enthalten. Annahme, dass $\mathcal{S}^*$ nicht ausführbar wäre. Wegen der Minimalitätsforderung von $<_{\mathfrak{A}}$ kann man diesen Zyklus als Folge von Aufträgen $x <_{\mathfrak{A}} y$ in $\mathcal{S}$ und $z_1^E <_{\mathfrak{A}^*} z_2^A$, ... durch die Anordnung beschreiben. Der Zyklus muss daher mindestens einen Auftrag enthalten, der nicht in den Pfaden von $\zeta_|^*$ bzw. $\zeta_{\mathcal{S}*}$ liegt (nach Satz 3.2.6 Punkt 2), denn sonst wäre bereits $\zeta_|^*$ nicht ausführbar. Sei R der Pfad aus $\mathcal{S}^*$, der diesen Auftrag enthält. Somit liegt R in $\zeta_{\mathcal{S}^*}$, was ein Widerspruch zu Punkt 1 von Satz 3.2.6 ist. □

Bemerkung 3.2.15. Dieser Satz zeigt nun, dass man einen Zyklus alleine betrachten kann mit seinen Abhängigkeiten. Wenn man hier eine ausführbare Anordnung findet, so ist das entsprechend angeordnete System $\mathcal{S}^*$ auch ausführbar. Man kann durch diese Eigenschaft also Rechenleistung einsparen, weil man nur die Pfade des Zyklus betrachten muss.

Wie bereits in der letzten Bemerkung erläutert wurde, erlaubt es der Satz 3.2.7, einen Zyklus alleine zu betrachten (genauer: die Einschränkung auf den Zyklus). Wenn man hierauf eine ausführbare Anordnung findet, so ist das entsprechend angeordnete System $\mathcal{S}^*$ auch ausführbar. Interessanter ist aber die Frage, ob man diese Aussage verallgemeinern kann, d.h. ob eine ausführbare Anordnung in der Einschränkung auf mehrere Zyklen auch wieder ein ausführbares System $\mathcal{S}^*$ erzeugt? Diese Frage wird durch den folgenden Satz beantwortet.

Satz 3.2.8 (Ausführbare Anordnungen von Pfaden innerhalb von Zyklen)**.** *Sei $\mathcal{S}$ ein ausführbares System von Pfaden und ζ_1, ..., ζ_q paarweise verschiedene Zyklen bezüglich der Relation $<$ in diesem System. Weiter seien ${}_|\zeta_1$, ..., ${}_|\zeta_q$ die Einschränkungen auf diese Zyklen. Seien A_i jeweils zu ${}_|\zeta_i$ passende Anordnungen ($1 \leq i \leq q$). Die Anwendung der Anordnung A_i auf ${}_|\zeta_i$ erzeuge die ausführbaren Systeme ${}_|\zeta_1^*$, ..., ${}_|\zeta_q^*$. Dann gilt: Das System $\mathcal{S}^*$, dass aus Anwendung der Anordnungen A_1, ..., A_q auf das System $\mathcal{S}$ entsteht, ist ausführbar.*

Beweis. Man betrachte zunächst das System $\mathcal{S}$, ${}_|\zeta_1^*$ und die Anordnung A_1. Diese erfüllen die Voraussetzungen von Satz 3.2.7. Daher erzeugt die Anwendung von A_1 auf $\mathcal{S}$ das ausführbare System $\mathcal{S}^1$.

Nun betrachte man das neue System $\mathcal{S}^1$, welches wegen Satz 3.2.6 Punkt 1 diesselbe Struktur wie $\mathcal{S}$ bezüglich der Relation $<$ besitzt. Dies bedeutet, dass die restlichen Zyklen aus $\mathcal{S}$ auch im System $\mathcal{S}^1$ unverändert existieren. Wegen Punkt 3 von Satz 3.2.6 wurde die innere Struktur der Relation $<_{\mathfrak{A}}$ dieser Zyklen durch die Anordnung A_1 nicht beeinflusst. Dies bedeutet, dass man Satz 3.2.7 sukzessive auf diese Zyklen anwenden kann.

Satz 3.2.4 impliziert, dass die Reihenfolge der Anordnungen bei dem eben beschriebenen Verfahren beliebig gewählt werden kann. Satz 3.2.5 gewährleistet schließlich,

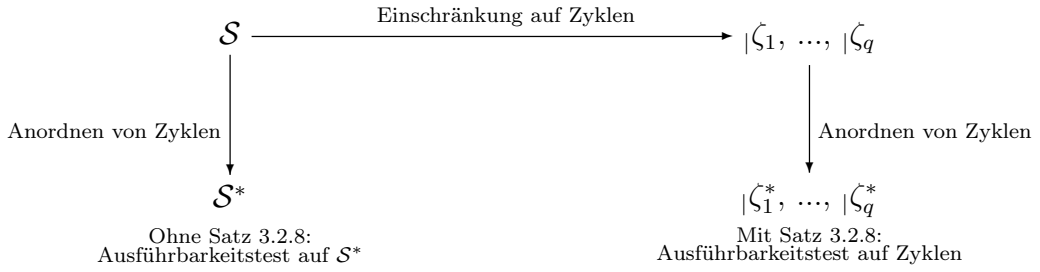


Abbildung 3.8.: Vorteil durch die Optimierung durch Satz 3.2.8

dass das ausführbare System $\mathcal{S}'$, welches durch eine vollständige Anwendung des Verfahrens entsteht, mit dem im Satz definierten System $\mathcal{S}^*$ identisch ist. □

Satz 3.2.8 erlaubt es nun, nur die Pfade der Zyklen eines ausführbaren Systems $\mathcal{S}$ zu betrachten. Wenn man in der Lage ist, auf ihnen günstige ausführbare Anordnungen zu finden, durch die man Ressourcen einsparen kann, so ist das entsprechend angeordnete System $\mathcal{S}^*$ ebenfalls ausführbar.

Der Vorteil des Resultats dieses Satzes liegt darin, dass man auf der Suche nach einer Optimierung (im Sinne einer Minimierung der zur Ausführung benötigten Ressourcen) nicht alle Pfade von $\mathcal{S}$ betrachten muss; es reicht vielmehr, dass man die Pfade der Zyklen und ihre Abhängigkeiten betrachtet. Wenn nun ein günstiges Schedule (möglicherweise auch nach mehreren Iterationen) auf den Zyklen berechnet ist, dann wird das System $\mathcal{S}$ entsprechend angeordnet. Dies wird durch das Diagramm in Abbildung 3.8 verdeutlicht. Wenn man Satz 3.2.8 nicht hätte, dann müsste man die Pfade eines Zyklus anordnen und gleich die Auswirkungen auf das *ganze System* $\mathcal{S}$ berechnen, was durch den linken Teil des Diagramms in Abbildung 3.8 gezeigt wird. Da das gesamte System gegebenenfalls mehr freie Pfade als nicht freie Pfade enthält, wäre der Aufwand für diese Vorgehensweise wesentlich größer als die Tests nur auf den Zyklen durchzuführen: Wenn das Resultat der Anordnung auf ganz $\mathcal{S}$ nicht ausführbar wäre, müsste man eine andere Anordnung untersuchen. Wenn das Resultat ausführbar wäre, so würde man versuchen, das entstandene System weiter zu optimieren.

Man beachte, dass nun für die minimal benötigte Anzahl von Ressourcen $R^*_{\min}$ zur Ausführung des Systems $\mathcal{S}^*$ eine Abwandlung von Satz 3.2.3 gilt.

$$R^*_{\min} = \max_{\zeta(Y) \in \mathcal{S}^*_\zeta} \Gamma_Y \tag{3.5}$$

In Gleichung (3.5) ist mit $\mathcal{S}^*_\zeta$ die Menge der Zyklen bezüglich $<$ gemeint, siehe Algorithmus 4. Γ_Y bezeichne die zur Ausführung der Pfade des angeordneten Zyklus $\zeta(Y)$ benötigte Zahl an Ressourcen. Der Beweis von Gleichung (3.5) ist klar. Weiter

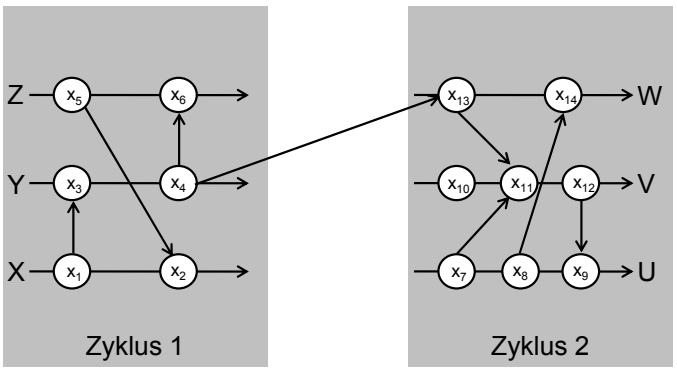


Abbildung 3.9.: Ausführbares System mit zwei Zyklen

ist offensichtlich, dass die Anzahl von Ressourcen $R^*_{\min}$ zur Ausführung des angeordneten Systems kleiner oder gleich der Anzahl von Ressourcen $R_{\min}$ ist, die für die Ausführung des ursprünglichen Systems benötigt werden:

$$R^*_{\min} \leq R_{\min}$$

Wie in den Bemerkungen zu Satz 3.2.3 gilt auch hier, dass diese Abschätzungen nur dann richtig sind, wenn alle Ressourcen gleich sind und alle diesselben Aufträge beziehungsweise Pfade ausführen können.

Beispiel 3.2.9. *Abbildung 3.9 zeigt ein ausführbares System mit zwei Zyklen. Die zu den Aufträgen des Systems gehörige Adjazenzmatrix sieht wie folgt aus:*

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
x_1	0	1	1	1	0	1	0	0	1	0	1	1	1	1
x_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_3	0	0	0	1	0	1	0	0	1	0	1	1	1	1
x_4	0	0	0	0	0	1	0	0	1	0	1	1	1	1
x_5	0	1	0	0	0	1	0	0	0	0	0	0	0	0
x_6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_7	0	0	0	0	0	0	0	1	1	0	1	1	0	1
x_8	0	0	0	0	0	0	0	0	1	0	0	0	0	1
x_9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_{10}	0	0	0	0	0	0	0	0	1	0	1	1	0	0
x_{11}	0	0	0	0	0	0	0	0	1	0	0	1	0	0
x_{12}	0	0	0	0	0	0	0	0	1	0	0	0	0	0
x_{13}	0	0	0	0	0	0	0	0	1	0	1	1	0	1
x_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Man rechnet leicht nach, dass sich durch die Anordnung $A^{x_2}_{x_3}$ der Pfade X und Y auf die Einschränkung auf Zyklus ζ_1 die folgenden Relation $<_{\mathfrak{A}}$ auf ihren Aufträgen

ergibt:

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	*0*	*1*	*1*	*1*	*0*	*1*
x_2	*0*	*0*	*1*	*1*	*0*	*1*
x_3	*0*	*0*	*0*	*1*	*0*	*1*
x_4	*0*	*0*	*0*	*0*	*0*	*1*
x_5	*0*	*1*	*1*	*1*	*0*	*1*
x_6	*0*	*0*	*0*	*0*	*0*	*0*

Man sieht also, dass der angeordnete Zyklus ebenfalls ausführbar ist, da $<_{\mathfrak{A}}$ auf seinen Aufträgen irreflexiv ist.

Ebenso leicht rechnet man nach, dass sich durch die Anordnung $A^{x_{14}}_{x_{10}}$ der Pfade W und V im Zyklus Z_2 die folgende Relation $<_{\mathfrak{A}}$ auf ihren Aufträgen ergibt:

	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
x_7	*0*	*1*	*1*	*1*	*1*	*1*	*0*	*1*
x_8	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*1*
x_9	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
x_{10}	*0*	*0*	*1*	*0*	*1*	*1*	*0*	*0*
x_{11}	*0*	*0*	*1*	*0*	*0*	*1*	*0*	*0*
x_{12}	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*
x_{13}	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*1*
x_{14}	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*0*

Auch dieser angeordnete Zyklus ist ausführbar.

Wenn man die beiden Anordnungen jetzt auf das Gesamtsystem anwendet, so ergibt sich für die Relation $<_{\mathfrak{A}}$ auf den Aufträgen:

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
x_1	*0*	*1*	*1*	*1*	*0*	*1*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*
x_2	*0*	*0*	*1*	*1*	*0*	*1*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*
x_3	*0*	*0*	*0*	*1*	*0*	*1*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*
x_4	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*
x_5	*0*	*1*	*1*	*1*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
x_6	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
x_7	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*0*	*1*
x_8	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*1*
x_9	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
x_{10}	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*1*	*1*	*0*	*0*
x_{11}	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*1*	*0*	*0*
x_{12}	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*
x_{13}	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*1*
x_{14}	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*0*

An dieser Adjazenzmatrix liest man leicht ab, dass das angeordnete Gesamtsystem ausführbar ist und dass die Einschränkungen auf Zyklus 1 beziehungsweise Zyklus 2 dieselbe Relation $<_{\mathfrak{A}}$ liefern wie die Anwendung der Anordnung $A_{x_3}^{x_2}$ beziehungsweise $A_{x_{10}}^{x_{14}}$ auf die Einschränkung des Originalsystems auf das System der Pfade von Zyklus 1 beziehungsweise Zyklus 2. Dabei sind die Aufträge aus den jeweiligen Zyklen grau markiert.

Dieses Beispiel verdeutlicht also die Aussagen aus Satz 3.2.6 Punkte 2 und 3 und Satz 3.2.8.

3.2.3.1. Eigenschaften der Anordnungen und Laufzeitoptimierung

Wie in Abbildung 3.8 dargestellt, wird $<_{\mathfrak{A}}$ nur auf den zu optimierenden Zyklen neu berechnet. Daraus ergibt sich die Frage, ob es notwendig ist, $<_{\mathfrak{A}}$ auf allen Aufträgen des Systems $\mathcal{S}$ neu zu berechnen, also ob die Kenntnis von $<_{\mathfrak{A}}$ auf dem System $\mathcal{S}^*$ wirklich benötigt wird? Die folgenden Sätze geben darauf eine Antwort.

Definition 3.2.10 (Vorgängermenge)**.** *Sei $\mathcal{S}$ ein System von Pfaden. Sei $x \in \mathcal{S}_{\mathfrak{A}}$ ein Auftrag eines Pfades. Dann ist*

$$\mathcal{V}(x) := \{y \in \mathcal{S}_{\mathfrak{A}} \mid y <_{\mathfrak{A}} x\}$$

die Vorgängermenge von x.

Bemerkung 3.2.16. Anschaulich gesprochen sind in der Menge $\mathcal{V}(x)$ alle Aufträge enthalten, auf die x warten muss, bevor er selber ausgeführt werden darf.

Falls $<_{\mathfrak{A}}$ irreflexiv auf $\mathcal{S}_{\mathfrak{A}}$ ist gilt offensichtlich für alle Aufträge $x \in \mathcal{S}_{\mathfrak{A}}$: $x \notin \mathcal{V}(x)$. Weiter muss es dann mindestens einen Auftrag $x_1 \in \mathcal{S}_{\mathfrak{A}}$ geben mit $\mathcal{V}(x_1) = \emptyset$. Falls es nämlich kein solches x_1 geben würde, so könnte kein einziger Auftrag ausgeführt werden, und das System $\mathcal{S}$ von Pfaden wäre somit nicht ausführbar. Nach Satz 2.3.2 wäre $<_{\mathfrak{A}}$ nun aber aber nicht irreflexiv im Widerspruch zur Voraussetzung.

Für jeden Auftrag x gilt wegen der Endlichkeit der Aufträge $\|\mathcal{V}(x)\| < \infty$.

Bemerkung 3.2.17. Mit der Definition der Vorgängermengen kann man auch die Ausführbarkeit eines Auftrages äquivalent zur Definition 2.3.1 formulieren: Ein Auftrag x in einem System ist ausführbar genau dann wenn entweder $\mathcal{V}(x) = \emptyset$ gilt oder alle Aufträge aus $\mathcal{V}(x)$ ausgeführt sind.

Nun sind aber speziell diejenigen Aufträge interessant, die einem Auftrag x direkt vorausgehen, das heißt die Aufträge, die unmittelbar vor einem Auftrag x ausgeführt werden müssen. Wenn diese Aufträge ausgeführt worden sind, so kann x auch ausgeführt werden.

Definition 3.2.11 (Direkte Vorgängermenge)**.** *Sei $\mathcal{S}$ ein System von Pfaden. Sei x ein Auftrag eines Pfades. Dann ist*

$$\max \mathcal{V}(x) := \{y \in \mathcal{V}(x) \mid \textit{Es existiert kein } z \in \mathcal{V}(x) \textit{ mit } y <_{\mathfrak{A}} z\}$$

die direkte Vorgängermenge von x.

Bemerkung 3.2.18. Zunächst ist klar, dass für alle $x \in \mathcal{S}_{\mathfrak{A}}$ gilt $\max \mathcal{V}(x) \subseteq \mathcal{V}(x)$. Damit ist auch die Kardinalität der direkten Vorgängermenge eines Auftrages x beschränkt und es gilt $\| \max \mathcal{V}(x)\| \leq \|\mathcal{V}(x)\| < \infty$.

Diese Definition ist nur hilfreich auf einem System von Pfaden, auf welchem $<_{\mathfrak{A}}$ irreflexiv ist. Andernfalls ist die direkte Vorgängermenge für bezüglich $<_{\mathfrak{A}}$ reflexive Aufträge unvollständig und es gibt gemäß Satz 2.3.2 Aufträge, die nicht ausgeführt werden können.

Anschaulich enthält die direkte Vorgängermenge eines Auftrages x in ausführbaren Systemen diejenigen Aufträge, nach deren Ausführung x garantiert ausgeführt werden darf. Diese Aussage ist in späteren Betrachtungen wichtig, siehe Satz 3.2.14.

Ein Algorithmus zur Berechnung der direkten Vorgängermenge von Aufträgen wird in Anhang A.4 vorgestellt.

Satz 3.2.9. *Sei $\mathcal{S}$ ein ausführbares System von Pfaden. Dann gilt für jeden Auftrag $x \in \mathcal{S}_{\mathfrak{A}}$:*

$$\mathcal{V}(x) \neq \emptyset \Longleftrightarrow \max \mathcal{V}(x) \neq \emptyset$$

Beweis.
„$\Rightarrow$“: Sei $y \in \mathcal{V}(x)$. Dann gilt entweder $y \in \max \mathcal{V}(x)$ und die Behauptung gilt, oder es existiert ein $y_1 \in \mathcal{V}(x)$ mit $y <_{\mathfrak{A}} y_1 <_{\mathfrak{A}} x$, und es gilt $y_1 \neq y$ da $<_{\mathfrak{A}}$ nach Voraussetzung irreflexiv auf $\mathcal{S}_{\mathfrak{A}}$ ist. Da die Menge $\mathcal{V}(x)$ endlich ist, kann nach höchstens $\|\mathcal{V}(x)\| - 1$ Schritten dieser Argumentation kein weiterer Auftrag eingeschoben werden und es muss einen Auftrag $y_F \in \mathcal{V}(x)$ geben mit $y_F \in \max \mathcal{V}(x)$.
„$\Leftarrow$“: Klar, gilt für beliebige Systeme, wegen $\max \mathcal{V}(x) \subseteq \mathcal{V}(x)$ für $x \in \mathcal{S}_{\mathfrak{A}}$. □

Bemerkung 3.2.19. Dieser Satz konkretisiert die Aussagen in der letzten Bemerkung: Wenn ein System ausführbar ist, und ein Auftrag x in diesem System eine nicht leere Vorgängermenge besitzt, dann muss auch seine direkte Vorgängermenge Aufträge enthalten. Aus dem Beweis geht hervor, dass dann für jeden Auftrag $y \in \mathcal{V}(x) \setminus \max \mathcal{V}(x)$ mindestens ein $y_F \in \max \mathcal{V}(x)$ existiert mit $y <_{\mathfrak{A}} y_F <_{\mathfrak{A}} x$.

Bemerkung 3.2.20. Eine äquivalente Formulierung des Satzes lautet: Sei $\mathcal{S}$ ein ausführbares System von Pfaden. Dann gilt für jeden Auftrag $x \in \mathcal{S}_{\mathfrak{A}}$:

$$\mathcal{V}(x) = \emptyset \Longleftrightarrow \max \mathcal{V}(x) = \emptyset$$

Wenn ein System von Pfaden ausführbar ist, kann man die Definition der Ausführbarkeit eines Auftrages schärfer fassen.

Satz 3.2.10. *Sei $\mathcal{S}$ ein ausführbares System von Pfaden. Dann gilt: Ein Auftrag $x \in \mathcal{S}_{\mathfrak{A}}$ ist ausführbar genau dann wenn entweder $\max \mathcal{V}(x) = \emptyset$ gilt oder alle Aufträge aus $\max \mathcal{V}(x)$ ausgeführt wurden.*

Beweis.
„⇒“: Sei $x \in \mathcal{S}_{\mathfrak{A}}$ ausführbar und gelte $\mathcal{V}(x) = \emptyset$. Dies ist nach Bemerkung 3.2.20 äquivalent zu $\max \mathcal{V}(x) = \emptyset$. Sonst sei $\mathcal{V}(x) \neq \emptyset$. Dies ist wiederum nach Satz 3.2.9 äquivalent zu $\max \mathcal{V}(x) \neq \emptyset$. Da aber nach Voraussetzung alle Aufträge aus $\mathcal{V}(x)$ ausgeführt sind, sind auch alle Aufträge aus $\max \mathcal{V}(x)$ ausgeführt.
„⇐“: Sei $x \in \mathcal{S}_{\mathfrak{A}}$ ausführbar und gelte $\max \mathcal{V}(x) = \emptyset$. Dies ist nach Bemerkung 3.2.20 äquivalent zu $\mathcal{V}(x) = \emptyset$. Sonst sei $\max \mathcal{V}(x) \neq \emptyset$. Dies ist wiederum nach Satz 3.2.9 äquivalent zu $\mathcal{V}(x) \neq \emptyset$. Falls nun $\mathcal{V}(x) = \max \mathcal{V}(x)$ gilt ist die Behauptung klar. Sonst sei $y \in \mathcal{V}(x) \setminus \max \mathcal{V}(x)$. Wegen Bemerkung 3.2.19 existiert ein $y_F \in \max \mathcal{V}(x)$ mit $y <_{\mathfrak{A}} y_F <_{\mathfrak{A}} x$. Da y_F aber bereits ausgeführt wurde, muss y nach Definition der Ausführbarkeit auch ausgeführt worden sein. □

Bemerkung 3.2.21. Nach Voraussetzung dieses Satzes muss jeder Auftrag aus $\mathcal{S}_{\mathfrak{A}}$ sowieso ausführbar sein. Er zeigt allerdings, dass es für eine Ressource reicht zu prüfen, ob die Aufträge der direkten Vorgängermenge eines Auftrages ausgeführt wurden, um seine Ausführbarkeit zu verifizieren.

Nun ist es interessant, sich die Zusammenhänge von Vorgängermengen eines Auftrages vor und nach einer Anordnung anzuschauen.

Lemma 3.2.1. *Sei x ein Auftrag in einem System $\mathcal{S}$ von Pfaden. Werden darauf Anordnungen durchgeführt, die zu einem System $\mathcal{S}^*$ führen. Sei $\mathcal{V}(x)$ die Vorgängermenge von x in $\mathcal{S}$ und $\mathcal{V}^*(x)$ die Vorgängermenge von x in $\mathcal{S}^*$. Dann gilt:*

$$\mathcal{V}(x) \subseteq \mathcal{V}^*(x)$$

Beweis. Klar, da alle Relationen $<_{\mathfrak{A}}$ aus $\mathcal{S}$ erhalten bleiben. □

Bemerkung 3.2.22. Man beachte, dass es bei diesem Satz unwesentlich ist, ob das System $\mathcal{S}$ und das aus Anordnungen resultierende System $\mathcal{S}^*$ ausführbar sind.

Des Weiteren sind die Zusammenhänge der direkten Vorgängermenge eines Auftrages vor und nach einer Anordnung wichtig für die weiteren Betrachtungen. Dabei ist es wichtig zwischen Aufträgen, welche auf freien Pfaden und Aufträgen, welche auf nicht freien Pfaden liegen, zu unterscheiden.

Der folgende Satz 3.2.11 betrachtet einen Auftrag auf einem freien Pfad. Es zeigt sich, dass seine direkte Vorgängermenge durch eine Anordnung nur verkleinert werden kann.

Satz 3.2.11. *Sei $\mathcal{S}$ ein System von Pfaden. Sei $\mathcal{S}^*$ das aus einer Anwendung von Anordnungen auf Pfade von Zyklen des Systems $\mathcal{S}$ resultierende System. Weiter sei x ein Auftrag eines freien Pfades, $\mathcal{V}(x)$ beziehungsweise $\max \mathcal{V}(x)$ seine Vorgängermenge beziehungsweise seine direkte Vorgängermenge in $\mathcal{S}$ und $\mathcal{V}^*(x)$ beziehungsweise $\max \mathcal{V}^*(x)$ die entsprechenden Mengen in $\mathcal{S}^*$. Dann gilt:*

$$\max \mathcal{V}^*(x) \subseteq \max \mathcal{V}(x)$$

Beweis. Wenn keine Zyklen in $\mathcal{S}$ existieren werden keine Anordnungen durchgeführt und die Behauptung gilt. Falls $\max \mathcal{V}^*(x) = \emptyset$ ist die Behauptung wahr. Sei also $y \in \max \mathcal{V}^*(x)$. Dann gilt $y <_{\mathfrak{A}} x$ im angeordneten System und es existiert kein $z \in \mathcal{V}^*(x)$ mit $y <_{\mathfrak{A}} z$. Es muss aber bereits $y <_{\mathfrak{A}} x$ im ursprünglichen System gewesen sein, denn sonst wäre diese Relation nur durch eine Anordnung zustandegekommen. Da x aber nicht auf einem Pfad eines Zyklus liegt, kann diese Relation nicht unmittelbar durch eine Anordnung zustandekommen, sondern durch Transitivität. Das wiederum ist ein Widerspruch zur Maximalität von y relativ zu x im angeordneten System. Also gilt $y \in \mathcal{V}(x)$. Annahme, es existiert ein $z \in \mathcal{V}(x)$ mit $y <_{\mathfrak{A}} z$, dann gilt dies auch im angeordneten System im Widerspruch zur Voraussetzung. □

Bemerkung 3.2.23. Man kann in der Formulierung des Satzes von einem Aufrag auf einem freien Pfad unabhängig vom zugehörigen System $\mathcal{S}$ oder $\mathcal{S}^*$ sprechen, da die Struktur von $<$ unter einer Anordnung von Zyklen wie in Satz 3.2.6 invariant ist.

Anschaulich ist die Aussage des Satzes, dass durch Anordnungen auf Zyklen die direkte Vorgängermenge eines Auftrages x auf einem freien Pfad höchstens verkleinert wird; es kommen insbesondere keine Aufträge zu ihr hinzu.

Eine ähnlich Aussage kann man nun auch für Aufträge auf nicht freien Pfaden formulieren.

Satz 3.2.12. *Sei $\mathcal{S}$ ein zyklenbehaftetes System von Pfaden. Werden Anordnungen auf Zyklen dieses Systems durchgeführt, die zu einem System $\mathcal{S}^*$ führen. Es sei x ein Auftrag eines nicht freien Pfades, $\mathcal{V}(x)$ beziehungsweise $\max \mathcal{V}(x)$ seine Vorgängermenge beziehungsweise seine direkte Vorgängermenge in $\mathcal{S}$ und $\mathcal{V}^*(x)$ beziehungsweise $\max \mathcal{V}^*(x)$ die entsprechenden Mengen in $\mathcal{S}^*$. Weiter seien $\mathcal{V}^*_{|x}(x)$ die Vorgängermenge von x in der Einschränkung auf den Zyklus des zu x gehörigen Pfades und $\max \mathcal{V}^*_{|x}(x)$ die zugehörige direkte Vorgängermenge. Dann gilt:*

$$\max \mathcal{V}^*(x) \subseteq \max \mathcal{V}(x) \cup \max \mathcal{V}^*_{|x}(x)$$

Beweis. Für $\max \mathcal{V}^*(x) = \emptyset$ is die Behauptung klar. Sei also $y \in \max \mathcal{V}^*(x)$. Zunächst wird gezeigt, dass y in $\mathcal{V}(x)$ oder in $\mathcal{V}^*_{|x}(x)$ liegt.

$y \notin \mathcal{V}(x)$: Da aber $y \in \mathcal{V}^*(x)$ gilt, muss dies durch die Anordnungen geschehen sein. Annahme, y liegt nicht auf einem Pfad, der zum Zyklus des Pfades von x gehört. Dann muss $y <_{\mathfrak{A}} x$ im angeordneten System $\mathcal{S}^*_{\mathfrak{A}}$ durch Transitivität vermittelt worden sein, da nur innerhalb von Zyklen angeordnet wird. Das stellt einen Widerspruch zur Maximalität von y zu x (in diesem System) dar. Also liegt y auf einem solchen Pfad und es gilt $y \in \mathcal{V}^*_{|x}(x)$ nach Satz 3.2.6 Punkt 2.

$y \notin \mathcal{V}^*_{|x}(x)$: Es gilt also $y <_{\mathfrak{A}} x$ im angeordneten System $\mathcal{S}^*_{\mathfrak{A}}$. Annahme, y liegt auf einem Pfad der zum Zyklus des Pfades von x gehört. Wenn bereits vor den Anordnungen $y <_{\mathfrak{A}} x$ galt, so ist das in $\mathcal{V}^*_{|x}(x)$ auch so im Widerspruch zur Voraussetzung. Wenn diese Beziehung durch die Anordnungen zustandegekommen ist, so gilt $y \in \mathcal{V}^*_{|x}(x)$ wegen Satz 3.2.6 Punkte 2 und 3 ebenfalls im Widerspruch zur Voraussetzung. Also liegt y auf einem Pfad, der nicht im Zyklus des Pfades von x liegt. Wegen der Maximalität von y zu x im angeordneten System $\mathcal{S}^*$ muss nun $y \in \mathcal{V}(x)$ gelten.

Es bleibt zu zeigen, dass y auch die zweite Eigenschaft eines Auftrages aus der direkten Vorgängermenge von x erfüllt. Es gilt $y <_{\mathfrak{A}} x$ in $\mathcal{S}^*$ und es existiert kein $z \in \mathcal{V}^*(x)$ mit $y <_{\mathfrak{A}} z$ in $\mathcal{S}^*$. Dann kann es auch in $\mathcal{S}$ und in der Einschränkung auf den zu x gehörenden Zyklus keinen Auftrag z geben mit $y <_{\mathfrak{A}} z$, denn sonst würde dies auch in $\mathcal{S}^*$ gelten im Widerspruch zur Voraussetzung. □

Bemerkung 3.2.24. Bei der Berechnung von $\mathcal{V}^*_{|x}(x)$ und $\max \mathcal{V}^*_{|x}(x)$ ist es wegen Satz 3.2.6 unwesentlich, ob zuerst die Anordnung auf $\mathcal{S}$ berechnet und dann die Einschränkung auf den zu x gehörigen Zyklus berechnet wird oder umgekehrt.

Wenn auf dem zum Pfad von x gehörigen Zyklus keine Anordnung durchgeführt wurde, so gilt $\max \mathcal{V}^*(x) \subseteq \max \mathcal{V}(x)$, da $\max \mathcal{V}^*_{|x}(x) \subseteq \max \mathcal{V}(x)$ wegen Satz 3.2.6 Punkte 1 und 3 gilt.

Satz 3.2.13. *Sei $\mathcal{S}$ ein System von Pfaden. Werden Anordnungen auf Zyklen dieses Systems durchgeführt, die zu einem System $\mathcal{S}^*$ führen. Es sei x ein Auftrag eines Pfades, $\mathcal{V}(x)$ beziehungsweise $\max \mathcal{V}(x)$ seine Vorgängermenge beziehungsweise seine direkte Vorgängermenge in $\mathcal{S}$ und $\mathcal{V}^*(x)$ beziehungsweise $\max \mathcal{V}^*(x)$ die entsprechenden Mengen in $\mathcal{S}^*$.*

- *Wenn x ein Auftrag auf einem freien Pfad ist, so gilt:*

$$\max \mathcal{V}^*(x) \subseteq \max \mathcal{V}(x) \subseteq \mathcal{V}(x) \subseteq \mathcal{V}^*(x) \tag{3.6}$$

- *Wenn x ein Auftrag eines nicht freien Pfades ist, dann seien $\mathcal{V}^*_{|x}(x)$ die Vorgängermenge von x in der Einschränkung auf den Zyklus des zu x gehörigen Pfades und $\max \mathcal{V}^*_{|x}(x)$ die zugehörige direkte Vorgängermenge. Es gilt nun:*

$$\max \mathcal{V}^*(x) \subseteq \max \mathcal{V}(x) \cup \max \mathcal{V}^*_{|x}(x) \subseteq \mathcal{V}(x) \cup \mathcal{V}^*_{|x}(x) \subseteq \mathcal{V}^*(x) \tag{3.7}$$

Beweis. Wenn x ein Auftrag auf einem freien Pfad ist, so ist die Behauptung klar nach Satz 3.2.11, Bemerkung 3.2.18 und Lemma 3.2.1. Die Behauptung ist nach Satz 3.2.12, Bemerkung 3.2.18 und Lemma 3.2.1 auch für einen Auftrag x auf einem nicht freien Pfad klar. Hier muss man sich zusätzlich verdeutlichen, dass $\mathcal{V}^*_{|x}(x) \subseteq \mathcal{V}^*(x)$ gilt. □

Satz 3.2.14. *Sei $\mathcal{S}$ ein ausführbares System von Pfaden. Werden darauf Anordnungen wie in Satz 3.2.8 durchgeführt. Dann reicht es, für die Ausführbarkeit eines Auftrages x auf einem*

- *freien Pfad die Beendigung der Ausführung jedes Auftrages der Menge $\mathcal{V}(x)$ zu prüfen.*
- *nicht freien Pfad die Beendigung der Ausführung jedes Auftrages der Menge $\mathcal{V}(x) \cup \mathcal{V}^*_{|x}(x)$ zu prüfen.*

Beweis. Sei x ein Auftrag auf einem freien Pfad. Dann gelten die Mengeninklusionen nach (3.6). Falls $\mathcal{V}(x) = \emptyset$ ist, so gilt das auch für $\max \mathcal{V}^*(x)$ und die Behauptung folgt aus Satz 3.2.10. Wenn nun alle Aufträge aus $\mathcal{V}(x)$ ausgeführt worden sind, dann bedeutet dies, dass auch alle Aufträge aus $\max \mathcal{V}(x)$ und $\max \mathcal{V}^*(x)$ ausgeführt worden sind, sofern darin Aufträge existieren. Die Behauptung folgt nun ebenfalls mit Satz 3.2.10.

Diesselbe Argumentation beweist die Aussage des Satzes für einen Auftrag auf einem nicht freien Pfad. □

Bemerkung 3.2.25. Dieser Satz stellt also eine Optimierungsmöglichkeit dar, durch die das System $\mathcal{S}^*$ nicht berechnet werden muss. Die Kenntnis der Relation $<_{\mathfrak{A}}$ auf $\mathcal{S}$ ist ausreichend, um entsprechende Rückschlüsse auf das System $\mathcal{S}^*$ zu ziehen.

Bemerkung 3.2.26. Im Gegensatz zu den vorangegangenen drei Sätzen wird hier die Ausführbarkeit des Systems vorausgesetzt. Dies ist notwendig, da nur genau dann auf die Ausführbarkeit von x gemäß Satz 3.2.10 geschlossen werden kann.

Beispiel 3.2.10. *In diesem Beispiel sollen die vorangegangenen Sätze demonstriert werden. Gegeben sei ein System $\mathcal{S}$ bestehend aus fünf Pfaden X, Y, Z, U und W wie in Abbildung 3.10. Offensichtlich liegen die Pfade Y, Z und U in einem Zyklus*

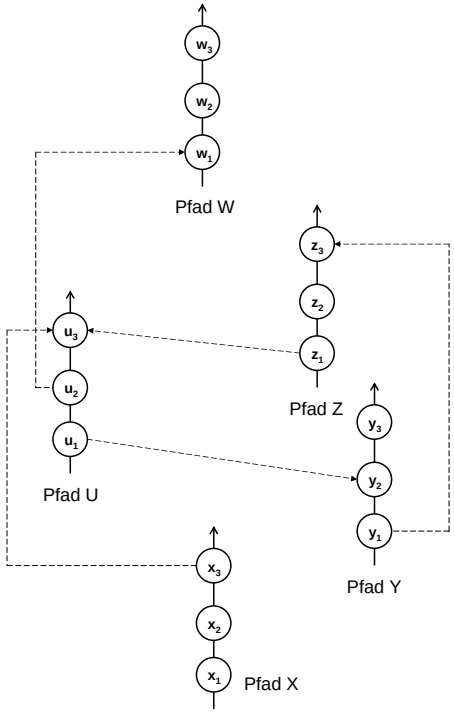


Abbildung 3.10.: Ausführbares Pfadsystem

bezüglich $<$ und bezüglich $<_{\mathfrak{A}}$ gelten die folgenden Abhängigkeiten:

$<_{\mathfrak{A}}$ *in* $\mathcal{S}$	x_1	x_2	x_3	y_1	y_2	y_3	z_1	z_2	z_3	u_1	u_2	u_3	w_1	w_2	w_3
x_1	*0*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*
x_2	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*
x_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*
y_1	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*
y_2	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
y_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
z_1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*0*	*0*	*0*
z_2	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*
z_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
u_1	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*
u_2	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*
u_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
w_1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*
w_2	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*
w_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Durch die Ausführung der Anordnung $A_{u_1}^{z_3}$ wird gezeigt, dass der Zyklus optimiert werden kann indem der Pfad Z vor dem Pfad U auf einer einzigen Ressource ausge-

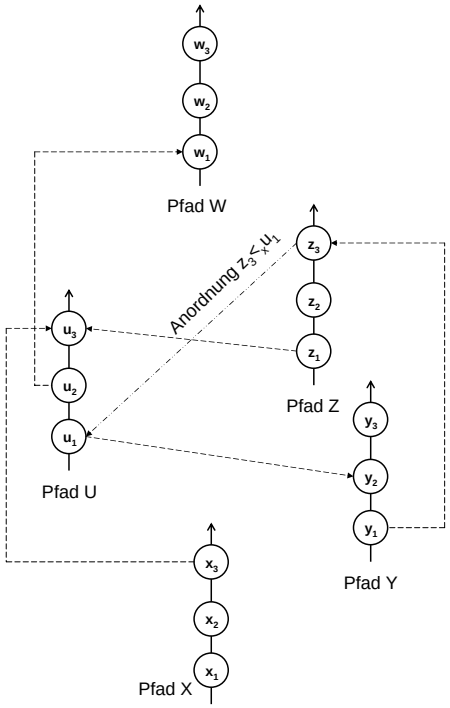


Abbildung 3.11.: Ausführbares angeordnetes Pfadsystem

führt wird, siehe Abbildung 3.11. Dabei sind die transitiven Abhängigkeiten, die durch die Anordnung erzeugt werden, nicht eingezeichnet. Die vollständige Adjazenzmatrix von $\mathcal{S}^*$ sieht nun wie folgt aus:

$<_{\mathfrak{A}}$ *in* $\mathcal{S}^*$	x_1	x_2	x_3	y_1	y_2	y_3	z_1	z_2	z_3	u_1	u_2	u_3	w_1	w_2	w_3
x_1	*0*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*
x_2	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*
x_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*
y_1	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
y_2	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
y_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
z_1	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
z_2	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
z_3	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*
u_1	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*
u_2	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*
u_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
w_1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*
w_2	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*
w_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Die Adjazenzmatrix zur Einschränkung auf die Pfade Y, Z und U des Zyklus sieht wie folgt aus:

$<_{\mathfrak{A}}$ *in Zyklus*	y_1	y_2	y_3	z_1	z_2	z_3	u_1	u_2	u_3
y_1	*0*	*1*	*1*	*0*	*0*	*1*	*1*	*1*	*1*
y_2	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*
y_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
z_1	*0*	*1*	*1*	*0*	*1*	*1*	*1*	*1*	*1*
z_2	*0*	*1*	*1*	*0*	*0*	*1*	*1*	*1*	*1*
z_3	*0*	*1*	*1*	*0*	*0*	*0*	*1*	*1*	*1*
u_1	*0*	*1*	*1*	*0*	*0*	*0*	*0*	*1*	*1*
u_2	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*
u_3	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Die Vorgängermenge eines Auftrages kann man einfach in diesen Tabellen ablesen, indem man die zugehörige Spalte betrachtet. Man sieht nun leicht, dass für jeden Auftrag x der Systeme $\mathcal{S}$ und $\mathcal{S}^*$ die Aussage von Lemma 3.2.1 gilt: $\mathcal{V}(x) \subseteq \mathcal{V}^*(x)$.

Für Aufträge auf freien Pfaden gilt auch die Aussage aus Satz 3.2.13, etwa für den Auftrag w_1: Man rechnet leicht nach, dass gilt $\mathcal{V}(w_1) = \{u_1, u_2\}$, $\max \mathcal{V}(w_1) = \{u_2\}$, $\mathcal{V}^*(w_1) = \{y_1, z_1, z_2, z_3, u_1, u_2\}$ und $\max \mathcal{V}^*(w_1) = \{u_2\}$.

Ebenso kann man die Aussage von Satz 3.2.13 für Aufträge auf nicht freien Pfaden bestätigen. So gilt beispielsweise für den Auftrag u_1: $\mathcal{V}(u_1) = \max \mathcal{V}(u_1) = \emptyset$, $\max \mathcal{V}^*_{|_{u_1}}(u_1) = \{z_3\}$, $\mathcal{V}^*(u_1) = \{y_1, z_1, z_2, z_3\}$ und $\max \mathcal{V}^*(u_1) = \{z_3\}$.

Für den Auftrag u_3 gilt weiter: $\mathcal{V}(u_3) = \{z_1, u_1, u_2\}$, $\max \mathcal{V}(u_3) = \{z_1, u_2\}$, $\max \mathcal{V}^*_{|_{u_3}}(u_3) = \{u_2\}$, $\mathcal{V}^*(u_3) = \{y_1, z_1, z_2, z_3, u_1, u_2\}$ und $\max \mathcal{V}^*(u_3) = \{u_2\}$.

Die Aussagen aus Satz 3.2.14 lassen sich an diesen Aufträgen mit den eben angegebenen Mengen leicht nachprüfen.

Analog zu den Definitionen von Vorgängermengen und direkten Vorgängermengen kann man *Nachfolgermengen* und *direkte Nachfolgermengen* zu einem beliebigen Auftrag x definieren, etwa $\mathcal{N}(x) := \{y \in \mathfrak{A} \mid x <_{\mathfrak{A}} y\}$ und $\min \mathcal{N}(x) := \{y \in \mathcal{N}(x) \mid \text{Es existiert kein } z \in \mathcal{N}(x) \text{ mit } z <_{\mathfrak{A}} y\}$. Die Sätze und das Lemma dieses Unterkapitels gelten dann genauso für diese Mengen; die Beweise können analog übernommen werden.

Nun kann man die Verstärkung der Relation $<_{\mathfrak{A}}$ durch eine Anordnung in einem Zyklus genauer verstehen: Die Vorgänger- und Nachfolgermenge eines Auftrages vergrößert sich zwar durch eine Anordnung. Für einen Auftrag auf einem freien Pfad gilt allerdings, dass durch eine solche Anordnung keine neuen Aufträge zu seiner direkten Vorgängermenge und seiner direkten Nachfolgermenge hinzukommen; es kann sogar passieren, dass ihre Anzahl abnimmt. Für Aufträge auf einem nicht freien Pfad kommen dagegen Aufträge in seiner direkten Vorgängermenge und seiner direkten Nachfolgermenge höchstens aus ihrem eigenen Zyklus hinzu.

Satz 3.2.14 erlaubt es schließlich, dass $<_{\mathfrak{A}}$ nach Anordnungen wie in Satz 3.2.8 nicht neu berechnet werden muss, und stellt somit eine Optimierung dar, da das Berechnen der transitiven Hülle einer Relation mit Warshalls Algorithmus kubischen Aufwand benötigt.

3.2.4. Strategien zur Anordnung von Pfaden in Zyklen

Satz 3.2.8 setzt die Existenz von ausführbaren Anordnungen von Zyklen voraus. Dabei bleibt aber die Frage offen, wie man eine solche Anordnung finden kann (wenn überhaupt eine existiert), um die benötigte Ressourcenzahl zur Ausführung der Pfade eines Systems zu optimieren? Da die analytischen Werkzeuge mittels der Relation $<$ hier nicht greifen, ergeben sich zwei Möglichkeiten:

1. Die Erstellung einer Bibliothek von einfachen beziehungsweise typischen Zyklen und Informationen darüber, wie diese zur Ressourcenminimierung behandelt werden sollen. Wenn dann ein Zyklus erkannt wird, der in dieser Sammlung vorliegt, dann kann er mittels der vorab vorhandenen Informationen aus der Sammlung möglichst ressourcenminimierend angeordnet werden.

2. Die Berechnung einer zufälligen Anordnung von Pfaden aus einem oder mehrerer Zyklen innerhalb einer gegebenen Zeit. Dieses Verfahren sollte dann genutzt werden, wenn in der Sammlung aus Punkt 1 keine Informationen über die Behandlung der Zyklen vorliegt. Wenn sich bei der zufälligen Anordnung eine ausführbare Reihenfolge ergibt, so muss entschieden werden, ob die Ressourceneinsparung an dieser Stelle ausreichend ist. Wenn dem so ist, kann Satz 3.2.8 direkt angewendet werden; sonst müssen weitere zufällige Anordnungen getestet werden.

Im Folgenden wird wieder davon ausgegangen, dass alle Ressourcen gleich sind, das heißt, dass jede Ressource jeden Auftrag beziehungsweise jeden Pfad ausführen kann.

Zunächst soll aber gezeigt werden, dass es keineswegs immer möglich ist, einen Zyklus derart aufzulösen, dass für seine Ausführung weniger Ressourcen als Pfade benötigt werden. Ein einfaches Beispiel dafür ist durch die drei Pfade und den Relationen zwischen ihnen (beziehungsweise ihren Aufträgen; die Relationen $<_{\mathfrak{A}}$ sind wieder durch die „Hintereinander“-Relation dargestellt) in Abbildung 3.12 gegeben. Man sieht unschwer, dass die drei Pfade ausführbar sind: Wenn man jedem Pfad eine eigene Ressource zuordnet, können alle drei Pfade vollständig ausgeführt werden. Weiterhin liegen sie bezüglich der Relation $<$ in einem Zyklus. Durch einfaches Nachrechnen stellt man nun fest, dass es keine mögliche Anordnung von Pfaden gibt, durch die die Anzahl der zur Ausführung benötigten Ressourcen auf weniger als drei reduziert werden kann.

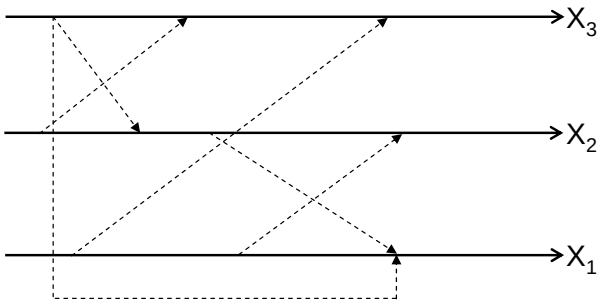


Abbildung 3.12.: Beispiel für drei Pfade in einem Zyklus, die auf drei verschiedenen Ressourcen ausgeführt werden müssen

Anordnen von Pfaden in typischen Zyklen Da innerhalb von Zyklen die Relation $<$ zwischen jedem Pfadpaar besteht, hilft ihre Betrachtung nichts mehr. Daher ist es nötig, die Pfade nur mit der Relation $<_{\mathfrak{A}}$ zu betrachten. Um hier eine Übersicht zu erhalten ist es sinnvoll die Relation $<_{\mathfrak{A}}$ nochmals auf Pfade zu übertragen, allerdings in einer schärferen Form als dies durch die Relation $<$ geschieht. Diese Relation $<_D$ auf den Pfaden wird durch folgende Definition beschrieben.

Definition 3.2.12. *Sei $\mathcal{S}$ ein System von Pfaden. Die (Pfad-)Relation $<_D$ auf der Menge $\mathcal{S}$ der Pfade ist die kleinste Relation, für die gilt: Wenn für zwei Pfade X und Y $(X \neq Y)$ gilt, dass es einen Auftrag x von X und y von Y gibt mit $x <_{\mathfrak{A}} y$, dann gilt $X <_D Y$.*

Bemerkung 3.2.27. Diese Definition stellt eine Vorstufe zur Definition 2.3.4 der Relation $<$ dar. Nach dem dort Gesagten sind Existenz und Eindeutigkeit von $<_D$ klar. Durch Algorithmus 1 wird auch gleich noch die zu $<_D$ gehörende quadratische $n \times n$-Adjazenzmatrix D ohne weiteren Berechnungsaufwand mitgeliefert.

Bemerkung 3.2.28. An den Pfaden X und W in Abbildung 3.10 kann man den Unterschied zwischen den Pfadrelationen $<$ und $<_D$ gut erkennen. Es gilt offensichtlich $X < W$ aufgrund Transitivität, aber $X <_D W$ gilt wegen der fehlenden Transitivität in der Definition von $<_D$ nicht.

Bemerkung 3.2.29. Der Algorithmus 3 zur Gewinnung paralleler Pfade bezüglich der Relation $<$ gilt auch für die schwächere Relation $<_D$. Als Resultat erhält man damit im besten Fall mehr parallelisierbare Pfade als durch Betrachtung von $<$. Diese Eigenschaft wird daher ohne weiteres in Kapitel 6 verwendet.

Definition 3.2.13 (Einschränkung bezüglich $<_D$)**.** *Sei $\mathcal{S}$ ein System von Pfaden zusammen mit den durch Definition 3.2.12 gegebenen Relationen $<_D$. Dann heißt eine Teilmenge $\mathcal{S}^E$ von $\mathcal{S}$ mit den auf die Pfade von $\mathcal{S}^E$ beschränkten Relationen $<_D$ **Einschränkung** bezüglich $<_D$ von $\mathcal{S}$ auf $\mathcal{S}^E$. Für zwei Pfade X und Y aus $\mathcal{S}^E$*

gilt also:

$$X <_D Y \text{ in } \mathcal{S} \Leftrightarrow X <_D Y \text{ in } \mathcal{S}^E$$

Bemerkung 3.2.30. Die zu einer Einschränkung gehörende quadratische Adjazenzmatrix D^E kann leicht aus D gewonnen werden, indem man in D nur die Pfade der Einschränkung betrachtet. Ein Vergleich zwischen der Relation $<$ und $<_D$ findet sich in Anhang A.5.

Im Folgenden werden fünf Beispiele für typische Zyklen vorgestellt und besprochen, wie sie behandelt werden sollen. Wie bereits oben gesagt wurde sind dies nur einige Beispiele aus der (unendlich) großen Zahl von möglichen Zyklen. Weiter gilt, dass in den Abbildungen die Relation $<_D$ beziehungsweise ihre Einschränkung auf die Zyklen dargestellt wird.

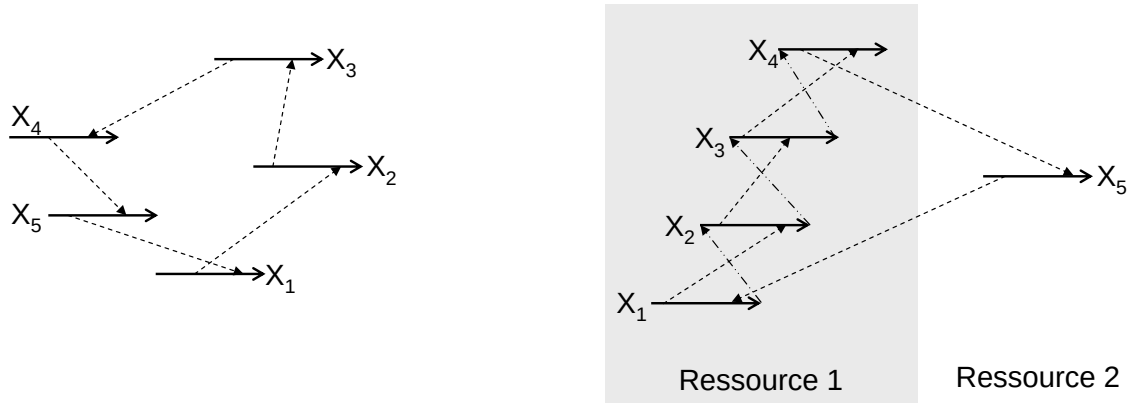
Erster Zyklustyp: Im einfachsten Fall ist ein Zyklus so aufgebaut wie die fünf Pfade in Abbildung 3.13 (a). Die wichtigste Eigenschaft des Zyklus besteht darin, dass durch die Relation $<_{\mathfrak{A}}$ ($<_D$), die zwischen je zwei Pfaden besteht, keine weitere Relation zu einem dritten Pfad durch Transitivität aufgebaut wird, wie dies etwa bei der Relation $<$ der Fall wäre. Die zugehörige Adjazenzmatrix D^*, die $<_D$ zwischen den Pfaden beschreibt, sieht wie folgt aus:

$$D^E = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Es ist offensichtlich, dass man Pfade, die auf einem solchen Zyklus liegen, auf nur zwei Ressourcen ausführen kann. Dies ist in Abbildung 3.13 (b) dargestellt: Man kann einen beliebigen der fünf Pfade herausgreifen, und diesen einer Ressource zuordnen. Die restlichen vier Pfade können auf einer weiteren Ressource in der bereits durch $<_{\mathfrak{A}}$ ($<_D$) vorgegebenen Reihenfolge *sequentiell* ausgeführt werden.

Insgesamt hat man bei diesem einfachen Zyklus eine Einsparung von $5-2=3$ Ressourcen. Wenn man dieses Ergebnis nun auf l Pfade ($l \in \mathbb{N}$, $l \geq 2$) überträgt, die bezüglich $<_{\mathfrak{A}}$ ($<_D$) die im ersten Absatz dieses Punktes beschriebene Eigenschaft erfüllen, so hat man eine Ersparnis von $l-2$ Ressourcen. Unter der Annahme, dass ein Zyklus bestehend aus 50 Pfaden vorliegt, der in das vorliegende Schema passt, so kann man sich durch eine solche Anordnung 48 Ressourcen gegenüber der nicht angeordneten Variante sparen.

Nun bleibt die Frage, wie man solch einen Zyklus erkennen kann. Dazu stellt es sich als hilfreich heraus, die zum Zyklus zugehörige Relation $<_D$ zu betrachten.



(a) Zyklus **(b) Angeordneter Zyklus**

Abbildung 3.13.: Auflösen eines einfachen Zyklus

Lemma 3.2.2. *Sei X ein nicht freier Pfad aus einem System $\mathcal{S}$, $\zeta(X)$ der zugehörige Zyklus mit $\|\zeta(X)\| = l$ und D^E die zur Einschränkung von $<_D$ auf $\zeta(X)$ zugehörige Adjazenzmatrix. Dann gilt:*

$$\textit{Transitive Hülle}(D^E) = \begin{pmatrix} 1 & 1 & ... & ... & 1 & 1 \\ 1 & 1 & ... & ... & 1 & 1 \\ ... & ... & 1 & ... & ... & ... \\ ... & ... & ... & 1 & ... & ... \\ 1 & 1 & ... & ... & 1 & 1 \\ 1 & 1 & ... & ... & 1 & 1 \end{pmatrix} = (1)_{i \times i}$$

Beweis. Zunächst werde $\overline{D^E} :=$ Transitive Hülle(D^E) definiert. Annahme, es existiert ein Null-Eintrag in $\overline{D^E}$. Dann existieren zwei Pfade Y und Z im Zyklus, die nicht in $\overline{D^E}$ in Relation stehen, o.B.d.A. muss der Pfad Z nicht auf den Pfad Y warten (in $<_{\mathfrak{A}}$). Da Y und Z aber Pfade eines Zyklus sind, stehen sie in der Relation $<$ in Beziehung und es gilt $Y < Z < Y$. Da $<$ nach Algorithmus 1 aber aus $<_D$ gewonnen wird, muss es folglich Pfade X_1, X_2, ..., X_i geben, die nicht alle im Zyklus enthalten sind (und ungleich Y und Z sind) und durch die insbesondere $Y < Z$ realisiert wird. Es gelte o.B.d. A. $X_1 \notin \zeta(X)$. Dann gilt aber $Y <_D X_1 <_D X_2 <_D ... <_D X_i <_D Z$, woraus $Y < X_1$ und $X_1 < Z$ folgt. Dies ist aber ein Widerspruch zu Satz 3.2.1 Punkt 5. □

Bemerkung 3.2.31. Dieser Hilfssatz zeigt, dass es reicht, die Einschränkung von $<_D$ auf einem Zyklus zu betrachten, um die Relation $<$ in Einschränkung

auf den Zyklus zu gewinnen. Eine Folgerung dieses Satzes ist, dass in der Matrix D^* in jeder Zeile und jeder Spalte mindestens eine Eins steht, sonst kann daraus durch den Algorithmus von Warshall nicht die voll besetzte Matrix $(1)_{i\times i}$ entstehen.

Satz 3.2.15. *Sei $\mathcal{S}$ ein ausführbares System von Pfaden, X ein nicht freier Pfad aus $\mathcal{S}$ und $\zeta(X)$ der zugehörige Zyklus. Weiter sei D^E die zur Einschränkung von $<_D$ auf den Zyklus gehörige Adjazenzmatrix. Dann sind äquivalent:*

1. *Für jeden Pfad des Zyklus gilt durch $<_D$, dass er auf genau einen anderen Pfad wartet und genau ein anderer Pfad auf ihn wartet. Ausgehend von einem beliebigen Pfad kann (über den Umweg über andere Pfade des Zyklus durch $<_D$) jeder beliebige Pfad des Zyklus erreicht werden.*
2. *D^E enthält in jeder Zeile genau eine* 1 *und in jeder Spalte genau eine* 1.

Beweis.
„1. $\Rightarrow$ 2.“: Wenn jeder Pfad auf genau einen anderen Pfad wartet und andererseits auf jeden Pfad von genau einem anderen Pfad gewartet wird, so ergibt sich eine Matrix D^E, die in jeder Zeile genau eine 1 enthält und in jeder Spalte genau eine 1 enthält.
„2. $\Rightarrow$ 1.“: Wenn D^E in jeder Zeile genau eine 1 und in jeder Spalte genau eine 1 enthält, so bedeutet dies gerade, dass jeder Pfad auf genau einen anderen Pfad wartet und andererseits auf jeden Pfad von genau einem anderen Pfad gewartet wird. Die Eigenschaft, dass man ausgehend von einem beliebigen Pfad über den Umweg durch andere Pfade des Zyklus jeden beliebigen Pfad des Zyklus erreichen kann ist klar wegen Lemma 3.2.2. □

Bemerkung 3.2.32. Es ist nicht notwendig die Charakterisierung der Erreichbarkeit von Pfaden in Punkt 1 mit aufzunehmen; sie wurde aber hinzugefügt, weil dadurch ein Zyklus wie in Abbildung 3.13 (a) einleuchtend beschrieben wird.

Satz 3.2.16. *Sei $\mathcal{S}$ ein ausführbares System von Pfaden, X ein nicht freier Pfad aus $\mathcal{S}$ und $\zeta(X)$ der zugehörige Zyklus. Weiter sei D^E die zur Einschränkung von $<_D$ auf den Zyklus gehörige Adjazenzmatrix.*
Dann gilt: Wenn D^E in jeder Zeile genau eine 1 *und in jeder Spalte genau eine* 1 *enthält, dann kann man einen beliebigen Pfad auf einer Ressource ausführen und alle restlichen Pfade in der Reihenfolge, die durch $<_D$ gegeben ist, auf einer anderen Ressource ausführen.*

Beweis. Nach Satz 3.2.15 wartet nun jeder Pfad des Zyklus auf genau einen anderen Pfad und auf jeden Pfad wartet genau ein anderer Pfad. Wähle nun einen beliebigen Pfad Y und beginne, ihn auf einer Ressource auszuführen.

Suche den Pfad Z, der auf die Ausführung von Y wartet. Solch ein Z muss aufgrund Voraussetzung existieren. Z kann nach einer Weile vollständig auf einer Ressource ausgeführt werden, aber spätestens dann wenn Y in der Ausführung soweit ist, dass er selber auf einen anderen Pfad wartet. Wenn Z nun nicht ausführbar wäre, so gäbe es in $<_{\mathfrak{A}}$ und somit in $<_D$ und $<$ Abhängigkeiten von dem Pfad, auf den Y wartet und Z, was ein Widerspruch zur Voraussetzung ist. Nach der Ausführung von Z kann nun der darauf wartende Pfad vollständig ausgeführt werden und so weiter bis schließlich Y ausgeführt werden kann (dies liegt an der endlichen Anzahl von Pfaden). Zu diesem Zeitpunkt sind alle anderen Pfade des Zyklus komplett ausgeführt: Wenn ein Pfad noch nicht ausgeführt wäre, dann gäbe es mindestens einen weiteren Pfad, zu dem der erste in wechelseitiger Beziehung durch $<_D$ steht. Diese Pfade hätten aber keine Beziehung zu den Pfaden, die zu Y gehören aufgrund Voraussetzung. Dann muss die transitive Hülle auf $<_D$ aber separate Zyklen ergeben im Widerspruch zur Voraussetzung. □

Bemerkung 3.2.33. Dieser Satz liefert nun einen effizienten Test, um einen Zyklus des hier beschriebenen Typs zu erkennen. Wenn der Zyklus aus l Pfaden besteht, so muss die Summe über alle Einträge der Matrix D^E genau l ergeben. Nach der Bemerkung 3.2.16 enthält D^E in jeder Zeile mindestens eine 1 und in jeder Spalte mindestens eine 1. Wenn die Summe über alle Matrixeinträge bei einem Zyklus von l Pfaden also genau l ergibt, so muss D^E in jeder Zeile genau eine 1 enthalten und auch in jeder Spalte genau eine 1 enthalten. Satz 3.2.15 zeigt dann die Charakteristik des Zyklus. Algorithmus 7 verdeutlicht die Vorgehensweise.
Der Aufwand des Algorithmus liegt auf der Hand: Es müssen höchstens

- $l^2 - 1$ arithmetische Operationen (Additionen) und
- ein Vergleich

durchgeführt werden.

Er kann einfach modifiziert werden, um auch einen Ablaufplan aus der Matrix D^* zu erhalten. Da man die freie Wahl hat nimmt man an, dass Pfad 1 auf einer exklusiven Ressource ausgeführt wird. Dann geht man wie im Beweis zu Satz 3.2.16 vor.
Der Aufwand für diesen Algorithmus ist auch wieder leicht abzuschätzen:

- Es wird jede Zeile von D^* untersucht. Da aber in jeder Zeile genau eine 1 und in jeder Spalte genau eine 1 stehen, werden genau $\sum_{i=1}^{l} i = \frac{l(l+1)}{2}$ Vergleiche benötigt.
- Weiter werden alle Pfade in die Liste eingefügt, also sind genau l Listenzuordnungs-Operationen notwendig.

Algorithmus 7 : Erkennung eines charakteristischen Zyklus

Eingabe : Quadratische $l \times l$-Matrix D^E, die $<_D$ auf den Pfaden eines Zyklus beschreibt
Ausgabe : *Wahr*, wenn es sich um einen Zyklus mit Charakteristik wie aus Satz 3.2.15 handelt
Falsch sonst

```
Beginn
    summe=0;
    für {k = 1; k ≤ l; k++} tue
        für {h = 1; h ≤ l; h++} tue
            summe = summe + D*[k][h]
        Ende
    Ende
    wenn {summe== l} dann
        Rückgabe Wahr
    sonst
        Rückgabe Falsch
    Ende
Ende
```

Algorithmus 8 : Auflösen eines charakteristischen Zyklus

Eingabe : Quadratische $l \times l$-Matrix D^*, die $<_D$ auf den Pfaden eines Zyklus mit Charakteristik wie aus Satz 3.2.15 beschreibt
Ausgabe : Ablaufplan

```
Beginn
    Liste=[Pfad 1];
    counter = 1;
    solange {counter ≠ 1} tue
        für {k = 1; k ≤ l; k++} tue
            wenn {D*[counter][k] == 1} dann
                counter = k;
                Liste = Liste ∘ [Pfad k];
                break;
            Ende
        Ende
    Ende
Ende
```

- Der Zähler „counter“ wird dabei jedes Mal neu definiert, also sind dafür genau l arithmetische Operationen (Schreiboperationen) notwendig.

Zweiter Zyklustyp: Nun wird ein Zyklus bestehend aus fünf Pfaden wie in Abbildung 3.14 (a) betrachtet. Für dieses und die restlichen drei Beispiele gilt, dass hier genau Zyklen mit einer festen Anzahl von Pfaden untersucht werden, und auf eine allgemeine Untersuchung wie im ersten Beispiel verzichtet wird, da dies nicht im Fokus dieser Arbeit steht.

Dieser Zyklus ist ähnlich dem aus Abbildung 3.13 (a), nur dass sich Relationen $<_{\mathfrak{A}}$ zwischen verschiedenen Pfaden aufgrund von Transitivität ergeben. Er enthält also eine transitive Kette, die bei Pfad X_1 beginnt und bei Pfad X_4 endet. Seine charakteristische Matrix D^E lautet:

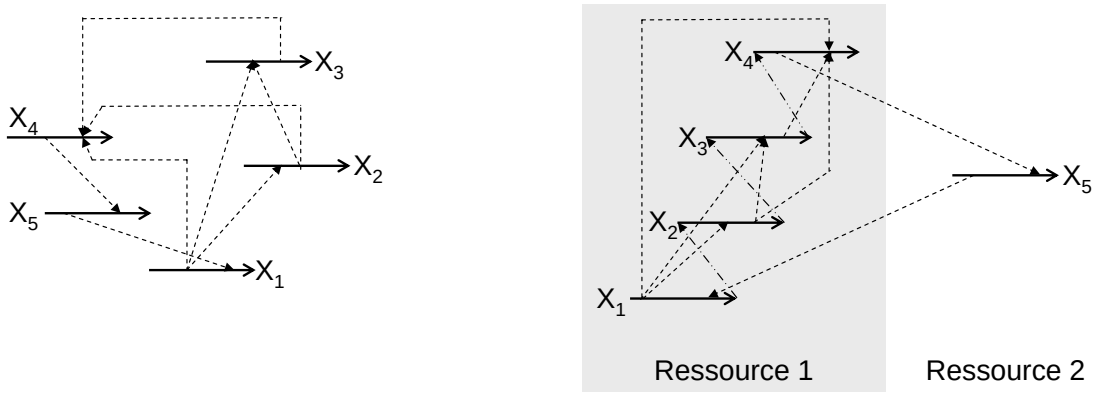
$$D^E = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Es ist klar, dass man auch Pfade, die auf einem Zyklus dieses Typs liegen, auf nur zwei Ressourcen ausführen kann. Dies ist in Abbildung 3.14 (b) dargestellt: Man beginnt mit Pfad X_1, der am Beginn der transitiven Kette steht, und ordnet ihn und die Pfade X_2, X_3 und X_4 in dieser durch $<_{\mathfrak{A}}$ definierten Reihenfolge einer Ressource zu. Pfad X_5 wird auch hier einer eigenen Ressource zugeordnet. Natürlich können die Pfade auch in diesem Beispiel in einem anderen Verhältnis auf die beiden Ressourcen verteilt werden, so können auch auf einer Ressource drei Pfade und auf der anderen Ressource zwei Pfade ausgeführt werden. Die einzige Bedingung dafür ist, dass $<_{\mathfrak{A}}$ eingehalten wird.

Die Ressourceneinsparung bei diesem Typ von Zyklus, der l Pfade enthält ($l \in \mathbb{N}$, $l \geq 2$), liegt wieder bei $l - 2$.

Der Aufwand, einen solchen Zyklus zu erkennen, hängt wieder von der Anzahl l der Pfade ab. Somit kann solch ein Zyklus in l^2 Schritten durch Vergleich mit der charakteristischen Matrix D^E erkannt werden.

Dritter Zyklustyp: Abbildung 3.15 (a) zeigt fünf Pfade, in denen sich zwei verschiedene Ketten ergeben. Die eine Kette beinhaltet die Pfade X_1, X_2 und X_3, während die andere Kette die Pfade X_2 und X_5 enthält. Die charakteris-



(a) Zyklus **(b) Angeordneter Zyklus**

Abbildung 3.14.: Auflösen eines einfachen Zyklus unter Berücksichtigung der Transitivität von $<_{\mathfrak{A}}$

tische Matrix D^E des Zyklus lautet wie folgt:

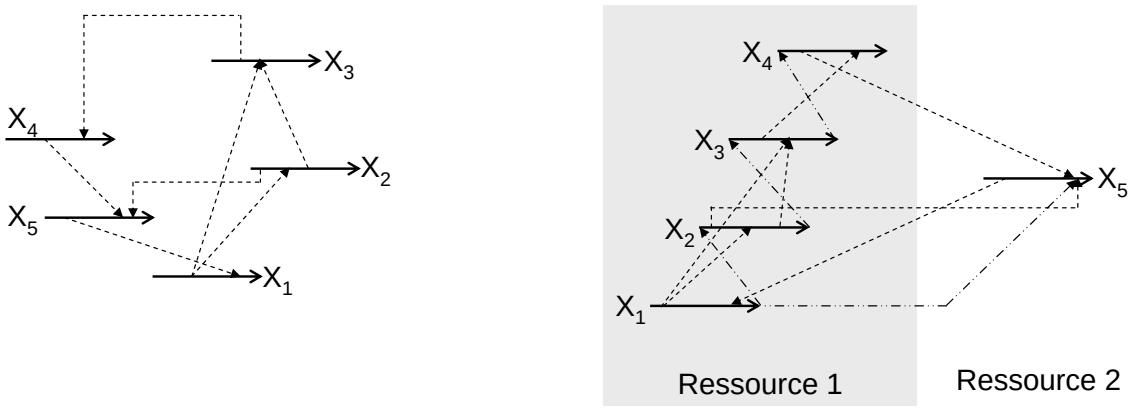
$$D^E = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Auch hier ist es leicht die Pfade durch Anordnen auf nur zwei Ressourcen auszuführen. Dies wird in Abbildung 3.15 (b) gezeigt.

Auch bei diesem Typ von Zyklus, welcher l ($l \in \mathbb{N}, l \geq 2$) Pfade enthält, liegt die Ressourceneinsparung bei $l - 2$. Der Aufwand, einen solchen Zyklus zu identifizieren, hängt wiederum von der Anzahl l der Pfade ab. Somit kann solch ein Zyklus in l^2 Schritten durch Vergleich mit der charakteristischen Matrix D^E erkannt werden.

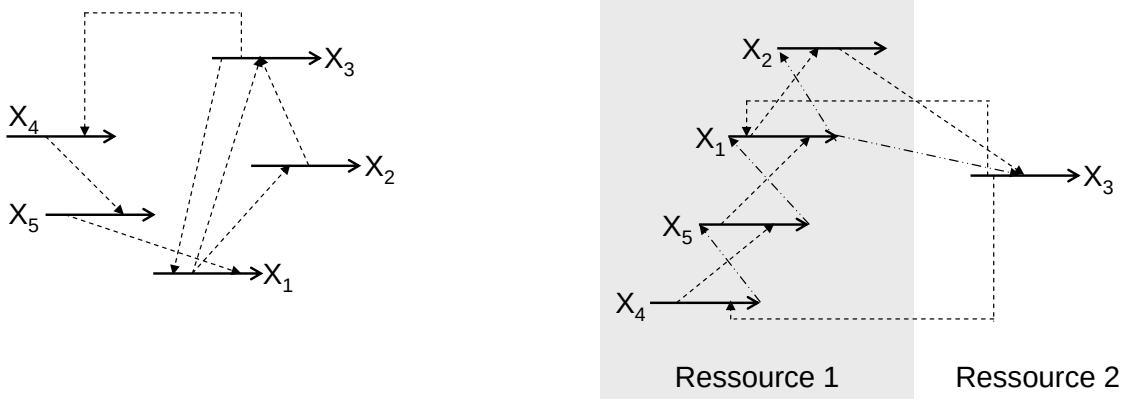
Vierter Zyklustyp: Abbildung 3.16 (a) zeigt nun einen Zyklus bestehend aus fünf Pfaden, der bezüglich $<_{\mathfrak{A}}$ aus zwei Teilzyklen besteht, die ineinander liegen. Der eine Zyklus umfasst alle Pfade, während der andere nur die Pfade X_1, X_2 und X_3 umfasst. Die charakteristische Matrix D^E des Zyklus lautet nun:

$$D^E = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$



(a) Zyklus **(b) Angeordneter Zyklus**

Abbildung 3.15.: Weiterer Zyklus mit Auflösung



(a) Zyklus **(b) Angeordneter Zyklus**

Abbildung 3.16.: Auflösen eines komplexen Zyklus (1)

Abbildung 3.16 (b) zeigt, wie man diesen Zyklus auflösen kann. Man ordnet Pfad X_3 eine eigene Ressource zu. Damit kann die Ausführung von Pfad X_3 begonnen werden. Auf einer anderen Ressource ordnet man die Pfade X_4, X_5, X_1 und X_2 in dieser Reihenfolge zu. Nachdem Pfad X_3 soweit wie möglich ausgeführt wurde, kann nun Pfad X_4 komplett auf der anderen Ressource ausgeführt werden, woraufhin Pfad X_5 vollständig ausgeführt wird. Damit sind alle Abhängigkeiten von Pfad X_1 gelöst und er kann ebenfalls beendet werden, worauf die Ausführung von Pfad X_2 erfolgt. Wenn dies geschehen ist, dann kann Pfad X_3 auf der anderen Ressource beendet werden.

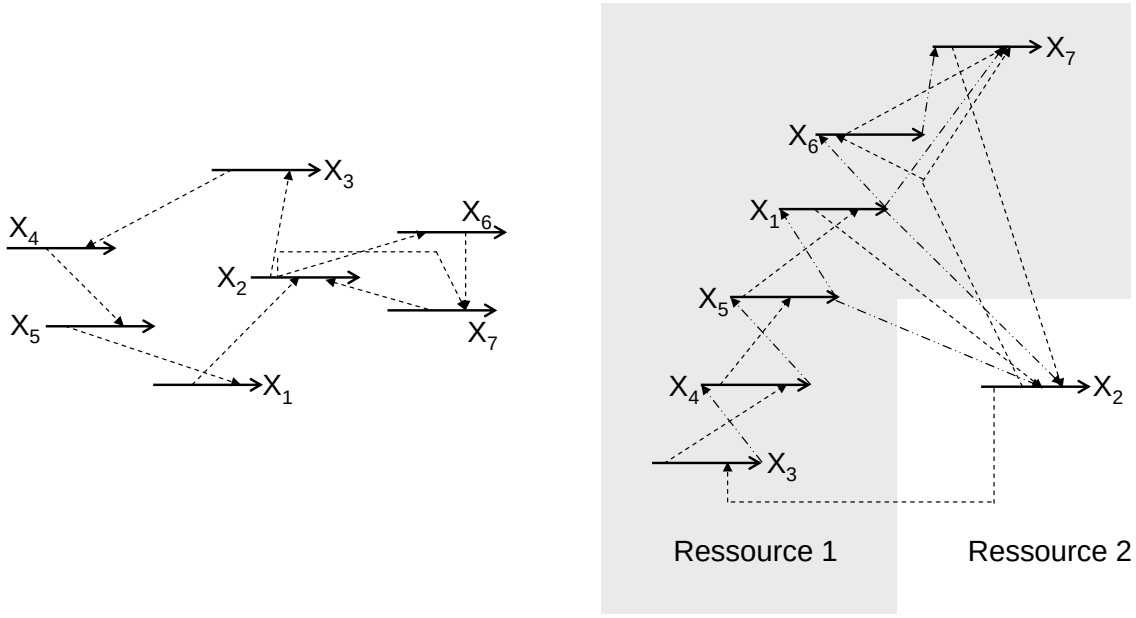
An diesem Beispiel sieht man, dass auch kompliziertere Zyklen aufgelöst werden können. Der Aufwand diesen speziellen Zyklus in der Matrix D^* zu erkennen, beträgt maximal $5^2 = 25$.

Fünfter Zyklustyp: Abbildung 3.17 (a) zeigt schließlich einen weiteren komplexeren Zyklus bestehend aus sieben Pfaden. Dabei besteht er bezüglich $<_D$ aus zwei einzelnen Zyklen, die Pfad X_2 als gemeinsamen Pfad enthalten. Die charakteristische Matrix D^E lautet nun:

$$D^E = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Die Auflösung des Zyklus wird in Abbildung 3.17 (b) gezeigt. Dabei wird Pfad X_2 (dem gemeinsamen Pfad der beiden Teilzyklen bezüglich $<_D$) eine eigene Ressource zugeordnet. Die Pfade X_3, X_4, X_5, X_1, X_6 und X_7 werden in dieser Reihenfolge auf einer anderen Ressource ausgeführt: Nach dem Beginn der Ausführung von Pfad X_2 ist Pfad X_3 vollständig ausführbar, woraufhin die Pfade X_4, X_5 und X_1 auch ausgeführt werden können. Pfad X_6 kann nun auch auf dieser Ressource ausgeführt werden, da seine einzigen Abhängigkeiten zu Pfad X_2 bestehen, die ebenfalls nach Beginn dessen Ausführung gelöst sind. Daraufhin kann Pfad X_7 auch auf dieser Ressource ausgeführt werden, woraufhin alle Abhängigkeiten zu Pfad 2 aufgelöst sind. Nun kann Pfad 2 komplett ausgeführt werden. Tatsächlich kann Pfad X_2 immer weiter teilweise ausgeführt werden, abhängig davon, welche seiner Abhängigkeiten aktuell gelöst wurden.

Der Aufwand um diesen Zyklus zu erkennen, beträgt maximal $7^2 = 49$ Vergleichsschritte in der Matrix.



(a) Zyklus **(b) Angeordneter Zyklus**

Abbildung 3.17.: Auflösen eines komplexen Zyklus (2)

Allen Beispielen ist gemein, dass man möglichst lange lineare Ketten von Pfaden in den Zyklen sucht und dann versucht, diese sequentiell auf einer Ressource auszuführen. Der folgende Satz beschreibt diese Beobachtung.

Satz 3.2.17. *Sei $\mathcal{S}$ ein ausführbares System von Pfaden und Z ein Zyklus (bezüglich $<$) darin. Seien X_1, X_2, ..., X_n Pfade aus Z mit der Eigenschaft $X_i <_D X_{i+1}$ oder X_i steht in keiner Relation zu X_{i+1} ($0 \leq i \leq n-1$). Weiter gebe es keine Relation der Form $X_j <_D X_i$ mit $j > i$ ($i, j \in \{1, ..., n\}$). Seien x_i^A und x_i^E der erste und letzte Auftrag des Pfades X_i ($0 \leq i \leq n$). Dann ist $\mathcal{S}$ mit der Anordnung $A_{n-1} \circ ... \circ A_2 \circ A_1$ ausführbar. Hierbei gilt $A_1 = x_1^E <_{\mathfrak{A}} x_2^A$, $A_2 = x_2^E <_{\mathfrak{A}} x_2^A$, ..., $A_{n-1} = x_{n-1}^E <_{\mathfrak{A}} x_n^A$.*

Beweis. Zunächst ist klar, dass die Reihenfolge der Anwendungen der Anordnungen keine Rolle spielt aufgrund ihrer Kommutativität. Annahme, dass $\mathcal{S}$ nach der Anwendung der Anordnungen nicht mehr ausführbar ist. Das bedeutet, dass nun ein Zyklus bezüglich der Relation $<_{\mathfrak{A}}$ existiert. Da $\mathcal{S}$ ohne die Anordnung ausführbar ist, muss mindestens eine der 2-Anordnungen A_i im Zyklus liegen. Dies sei o.B.d.A. A_1. Somit gilt $x_1^E <_{\mathfrak{A}} x_2^A$ und $x_2^A <_{\mathfrak{A}} x_1^E$. Die letztere Relation existiert in $\mathcal{S}$ nicht, sie muss also durch die Anordnung entstanden sein. Dies bedeutet, dass die Relation durch Transitivität der Form $x_2^A <_{\mathfrak{A}} x_i^E <_{\mathfrak{A}} x_{i+1}^A <_{\mathfrak{A}} ... <_{\mathfrak{A}} x_j^E <_{\mathfrak{A}} x_{j+1}^E <_{\mathfrak{A}} x_1^E$ entsteht (Die Indices stammen aus der Menge $\{3, ..., n\}$). Da die Kette endlich ist, kann man sie von links nach rechts durchgehen und an einer Stelle muss eine Re-

lation der Form $x_j^E <_{\mathfrak{A}} x_i^A$ mit $j > i$ auftreten. An der Kette liest man sofort ab, dass dies nicht durch die Anordnung geschieht. Daher muss diese Relation bereits in $\mathcal{S}$ existiert haben im Widerspruch zur Voraussetzung. Die Argumentation geht genauso, wenn mehrere 2-Anordnungen im Zyklus liegen. □

Bemerkung 3.2.34. Dieser Satz zeigt, dass lineare Ketten von Pfaden (bezüglich $<_D$) selbst innerhalb von Zyklen hintereinander auf einer Ressource ausgeführt werden können.

Bemerkung 3.2.35. Man hätte im Beweis auch zunächst die Einschränkung auf die Pfade des Zyklus betrachten können. Mit Hilfe von Satz 3.2.8 hätte sich dann auch die Ausführbarkeit des angeordneten Systems ergeben. Dies zeigt, dass Satz 3.2.8 in erster Linie nützlich für die Ermittlung von ausführbaren Anordnungen auf Zyklen ist; für Aussagen wie etwa in diesem Satz 3.2.17 benötigt man ihn nicht, da es für den Beweis gleichgültig ist, ob man die Anordnungen in der Einschränkung auf den Zyklus oder aber auf dem ganzen System betrachtet.

Im ersten Beispiel wurde der Typ des Zyklus durch die Matrix D^E charakterisiert; dadurch ist man in der Lage, einen solchen Zyklus unabhängig von der Anzahl der darin enthaltenen Pfade zu erkennen. Der benötigte Aufwand zur Erkennung wurde genau berechnet.

In den anderen Beispielen wurden konkrete Zyklen mit einer festen Anzahl von Pfaden und deren Auflösung betrachtet. In der Bibliothek liegen somit ihre charakteristischen Matrizen D^E vor, sowie deren Auflösungsschemata.

Satz 3.2.18. *Der Aufwand für den Vergleich einer bestimmten Matrix D^E mit einer Matrix gleicher Dimension in der Bibliothek beträgt l^2 Vergleiche bei l Pfaden.*

Aus den Auflösungsschemata der Bibliothek ergeben sich nun eine Reihe von Anordnungen. Wenn man diese auf das System $\mathcal{S}$ anwendet, so erhält man $<_{\mathfrak{A}}$ ($<_D$) unter Berücksichtigung dieser Anordnungen.

Die hier dargestellten Beispiele von Zyklen stellen nur eine kleine Auswahl dar, an der man aber sehen kann, wie es möglich ist, sie aufzulösen. Für eine konkrete Anwendung kann, wie zu Beginn des Unterkapitels bereits beschrieben, eine Bibliothek oft vorkommender Zyklen und deren Auflösung angelegt werden. Dies kann durch Profiling geschehen, also dem Betrachten von Szenarien der Anwendung und dem Auflösen der vorkommenden Zyklen von Hand.

3.2.4.1. Berechnung zufälliger Anordnungen innerhalb von Zyklen

In vielen Fällen wird man aber in der Bibliothek keine Information zur Auflösung eines Zyklus finden. Dann bleibt die Möglichkeit, eine Anordnung von Pfaden zu raten und mittels der Aussage von Satz 2.3.2 zu prüfen, ob diese Anordnung ausführbar ist. Dabei sollten in der Matrix D^E solche Pfade für die Anordnung gewählt werden,

die relativ wenige Abhängigkeiten zu anderen Pfaden besitzen. Die Beispiele des vorherigen Unterkapitels motivieren diese Vorgehensweise. Wenn eine oder mehrere solcher Anordnungen gefunden worden sind, so werden sie auf das gesamte System $\mathcal{S}$ angewandt, so dass $<_{\mathfrak{A}}$ darauf bekannt ist.

Das Ziel beim Finden einer ausführbaren Anordnung soll sein, dadurch eine oder mehrere Ketten von Pfaden innerhalb eines Zyklus zu erzeugen. Die Pfade innerhalb einer Kette können dabei nach Konstruktion sequentiell ausgeführt werden. Daher muss die Anordnung so gewählt werden, dass disjunkte Ketten von Pfaden erzeugt werden.

Algorithmus 9 : Optimieren eines Zyklus in einem System $\mathcal{S}$ von Pfaden

```
Eingabe : Einschränkung E auf alle Pfade eines Zyklus bezüglich < eines
          Pfadsystems S
Ausgabe : Optimierter Zyklus
Beginn
    boolean abbruch;
    wenn Restzeit>0 dann
        abbruch = false;
    sonst
        abbruch=true;
    Ende
    int abbruchZaehler = 100;
    Anordnung A = id;
    solange (abbruch==false ∧ abbruchZaehler>−1 ∧ Restzeit > 0) tue
        Erzeuge zufällige Anordnung A;
        wenn A ∘ A ist ausführbar dann
            A = A ∘ A;
            wenn Ressourceneinsparung ausreichend? dann
                abbruch = true;
            Ende
        Ende
        abbruchZaehler−−;
        Dekrement(Restzeit);
    Ende
Ende
```

Ein Scheduler kann eine solche Anordnungen erzeugen und diese auf ihre Ausführbarkeit hin untersuchen. Wenn diese Anordnung nicht ausführbar ist, so kann er eine andere Anordnung wählen oder das Verfahren nach einer gewissen Anzahl Versuche abbrechen (hier: 100 Versuche). Wenn die Anordnung ausführbar ist, so muss er entscheiden, ob er eine verbesserte Optimierung durch weiteres Anordnen

versucht (dies ist wegen der Aussage von Satz 3.2.4 möglich), oder ob die Optimierung bis zu diesem Punkt reicht und er das Verfahren abbricht. Dieses Verfahren wird in Algorithmus 9 nochmals verdeutlicht. Natürlich wird man in einer konkreten Implementierung des Algorithmus eine Art Tiefen- oder Breitensuche realisieren, wodurch nicht nur ein Zweig von Anordnungen, sondern verschiedene solcher Zweige untersucht werden. Der Algorithmus hier soll nur eine Idee geben, wie die eigentliche Untersuchung dann aussehen kann.

Die wichtigsten Parameter des Algorithmus sind einerseits der Abbruch-Zähler (abbruchZaehler), durch den festgelegt wird, wieviele Anordnungsversuche maximal gemacht werden, und andererseits die Ressourceneinsparung. Diese beiden Parameter sind in jeder Implementierung von Bedeutung.

Die Größe des Abbruch-Zählers hängt natürlich vom Berechnungsaufwand ab, der benötigt wird um zu prüfen, ob eine Anordnung ausführbar ist. Wie bereits nach dem Beweis zu Satz 2.3.2 angegeben wurde, liegt der Berechnungsaufwand der transitiven Hülle von $<_{\mathfrak{A}}$ bei k^3 Vergleichsoperationen und ebenso vielen Matrixmodifikationen, wobei k der Anzahl der Aufträge eines Systems entspricht. Dies bedeutet, dass der Scheduler für *jede Prüfung* einer Anordnung einen Aufwand in dieser Größenordnung benötigt. Daher ist es eine Frage der zu Verfügung stehenden Zeit, wie oft der Scheduler einen Zyklus zu Optimierungszwecken maximal anordnen kann. Wenn die Optimierung nicht zur Laufzeit erfolgt, dann spielt der Aufwand keine Rolle, und es kann eine sehr genaue Untersuchung vorgenommen werden. Wenn dagegen die Optimierung zur Laufzeit erfolgt, dann kann die Analyse entsprechend weniger präzise ausfallen.

Die Ressourceneinsparung kann man schließlich direkt aus den Anordnungen ablesen. Alle Pfade, die durch eine ausführbare Anordnung in einer Kette bezüglich $<_{\mathfrak{A}}$ geordnet wurden, können auf einer Ressource ausgeführt werden. Daher kann der Scheduler direkt aus den Anordnungen ermitteln, wie gross die Ressourceneinsparung ist. Diese Tatsache mache man sich an der verschiedenen Beispielen des letzten Unterkapitels klar. Man kann dies auch in einer Formel ausdrücken. Dabei sei j die Anzahl der Pfade eines Zyklus, und j_i sei die Anzahl der Pfade, die durch Anordnungen in eine sequentielle Ausführungsreihenfolge in einer Kette i gebracht wurden (beachte, dass verschiedene Ketten nach Voraussetzung disjunkt sind). Die Anzahl der dann noch benötigten Ressourcenzahl (wieder unter der Voraussetzung von Ressourcen gleicher Eigenschaften) ergibt sich zu:

$$\text{Anzahl benötigter Ressourcen} = j - \sum_i (j_i - 1) \tag{3.8}$$

Mit Hilfe dieser Formel kann der Scheduler also die nun noch zur Ausführung benötigte Anzahl von Ressourcen berechnen. Wenn in einer konkreten Anwendung die Anzahl insgesamt vorhandener Ressourcen bekannt ist, kann man auf dieser Basis ein Optimierungskriterium für die Anordnung im Algorithmus definieren, beispiels-

weise ein maximales Verhältnis von Anzahl der benötigten Ressourcen und Anzahl der vorhandenen Ressourcen.

Durch diese Präzisierung kann nun die Formel aus (3.5), die die minimal benötigte Anzahl von Ressourcen $R^*_{\min}$ zur Ausführung eines angeordneten Systems $\mathcal{S}^*$ beschreibt, weiter verfeinert werden. Dabei wird durch den Index Y bei den Maßzahlen der zugehörige Zyklus $\zeta(Y)$ gekennzeichnet. Es gilt nun:

$$R^*_{\min} = \max_{\zeta(Y)\in\mathcal{S}^*_\zeta} \left\{ j^{(Y)} - \sum_{i^{(Y)}} (j_i^{(Y)} - 1) \right\} \tag{3.9}$$

3.3. Überblick über den Ablauf des Ausführbarkeitstests

Abbildung 3.18 gibt nun einen Überblick, wie man ein System $\mathcal{S}$ von Pfaden auf seine Ausführbarkeit testen kann. Die einzelnen Schritte des Tests sind dabei exakt aus den Unterkapiteln 2.2 und 2.3 entnommen und die verwendeten Algorithmen werden bei jeder Aktion angegeben.

Im Folgenden wird angenommen, dass es einen *Scheduler* gibt, der diese Tests und Optimierungen durchführt.

1. Zunächst muss der Scheduler das System $\mathcal{S}$ von Pfaden untersuchen. Dazu stellt er die Relation $<_{\mathfrak{A}}$ auf. Er kann dies, wie im Abschnitt vor Satz 2.3.1 beschrieben, tun.

   Dann prüft er $<_{\mathfrak{A}}$ auf Irreflexivität: Wenn die Relation nicht irreflexiv ist, so ist sie nach Satz 2.3.2 nicht ausführbar, und er sendet $\mathcal{S}$ an einen Anwender zurück, damit dieser eventuell ein Re-Design der Pfade vornehmen kann. Falls die Relation irreflexiv ist, so wird sie an die nächste Test-Stufe des Schedulers weitergegeben.

2. Wenn $<_{\mathfrak{A}}$ irreflexiv ist, so wird die Relation $<$ aufgestellt und die paarweise parallel ausführbaren Pfade nach Algorithmus 3 ermittelt. Diese können schon an dieser Stelle berechnet werden, da sich durch eventuellen Anordnungen auf Zyklen nichts an $<$ ändert.

   Nun gibt es auch hier wieder zwei Möglichkeiten: Entweder ist $<$ irreflexiv oder nicht. Wenn $<$ irreflexiv ist, dann bedeutet das, dass die Pfade des Systems $\mathcal{S}$ geordnet werden können, das heißt, dass mittels Algorithmus 2 eine Ordnung bezüglich $<$ auf den Pfaden erzeugt wird, die dann in dieser Reihenfolge ausgeführt werden können. Dabei können auch mehrere Ketten von Pfaden entstehen, wobei die Ausführung von Pfaden verschiedener Teilsysteme völlig unabhängig voneinander erfolgt.

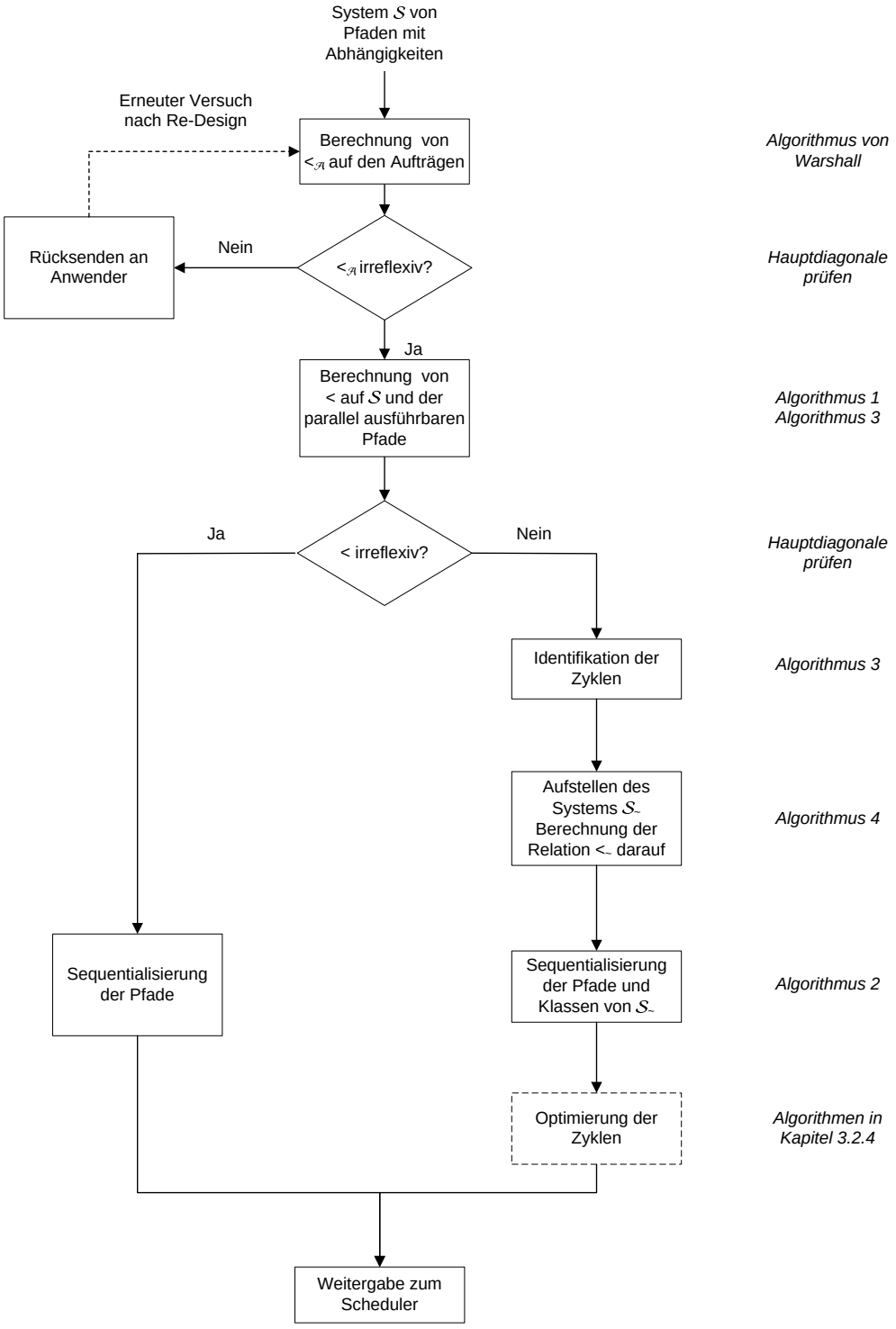


Abbildung 3.18.: Ablauf der Ausführbarkeitsprüfung auf einem System $\mathcal{S}$ von Pfaden

Falls $<$ nicht irreflexiv ist, dann sind eine Reihe weiterer Untersuchungen nötig, die im folgenden Punkt beschrieben werden.

3. $<$ ist also nicht irreflexiv. Das bedeutet, dass das System $\mathcal{S}$ zwar durchaus ausführbar ist, aber bezüglich $<$ existieren Zyklen, die eine sequentielle Ausführung der Pfade auf einer Ressource unmöglich machen. Zunächst muss der Scheduler die Pfade, die zu (möglicherweise) verschiedenen Zyklen gehören, identifizieren. Dann braucht er für den Moment die Pfade eines Zyklus nicht einzeln betrachten, sondern kann einen Repräsentanten daraus auswählen. So erhält er ein System $\mathcal{S}_\sim$, welches aus freien Pfaden und Repräsentanten von Zyklen besteht. Auf diesem System kann er nun die Relation $<_\sim$ aufstellen, die wie $<$ eine Aussage über die Hintereinanderausführung von Pfaden und Repräsentantenpfaden (also den Pfaden in Zyklen) macht. $<_\sim$ ist irreflexiv auf $\mathcal{S}_\sim$ und damit kann wieder Algorithmus 2 auf dieses System angewendet werden. Dadurch wird wieder eine Ordnunug auf den Pfaden und Repräsentantenpfaden von $\mathcal{S}_\sim$ erzeugt, wodurch eine Reihenfolge vorgegeben wird, in der diese ausgeführt werden müssen. Auch hier können Ketten von Pfaden und Repräsentantenpfaden entstehen. Im Gegensatz zu obigem Fall kann man die Pfade nun nicht mehr von nur einer Ressource ausführen lassen; vielmehr hängt die Anzahl der benötigten Ressourcen nun von der Kardinalität der einzelnen Zyklen ab.

   Um dieses Problem zu entschärfen kann der Scheduler nun verschiedene Optimierungsstrategien auf die Zyklen anwenden, was durch Satz 3.2.8 ermöglicht wird: Eine Möglichkeit besteht in einer Bibliothek, die Anweisungen enthält, wie bestimmte Zyklen aufgelöst werden können. Eine andere Möglichkeit ist, eine Brute-Force-Suche nach anderen ausführbaren Anordungen der Zyklen durchzuführen. Ein Ablaufplan kann nun an eine Zuordnungseinheit weitergeben werden, die die Pfade von $\mathcal{S}$ zur Laufzeit bestimmten Ressourcen zuweist.

4. Ein Ablaufplan, welcher die Teilsysteme von Pfaden und Zyklen von Pfaden sowie die originale Relation $<_{\mathfrak{A}}$ und auch $<_{\mathfrak{A}}$ auf den optimierten Zyklen enthält, wird nun an eine Zuordnungseinheit weitergegeben, die die Pfade von $\mathcal{S}$ zur Laufzeit bestimmten Ressourcen zuweist, siehe Kapitel 5.

   Man beachte, dass $<$ trotz eventueller Optimierungen gleich bleibt, siehe Satz 3.2.6 Punkt 1. Somit müssen weder $<$ noch $<_\sim$ neu berechnet werden.

3.4. Abschätzung der Planungs- und Optimierungszeit

Nun ist für einen Anwender noch der maximale Zeitaufwand T_A, die *Worst Case Execution Time* (WCET), zur Berechnung eines Ablaufplans interessant. T_A setzt

sich aus der WCET T_G für die Vorberechnung (Prüfen auf Ausführbarkeit eines System $\mathcal{S}$ und seine Sequentialisierung) und der WCET T_O für die Optimierung (gestrichelte Box in Abbildung 3.18) zusammen; es gilt also $T_A = T_G + T_O$.

Hierbei ist T_G leicht zu ermitteln, wohingegen T_O aufgrund der wählbaren Strategien und der variablen Anzahl der Optimierungsschritte wesentlich schwieriger abzuschätzen ist.

Ein Anwender kann diese Daten nun auf zwei verschiedene Arten nutzen:

1. Falls es eine (systembedingte oder anwendungsbedingte) Deadline T_D gibt, nach der spätestens mit der Ausführung begonnen werden muss, so ist es für einen Anwender leicht zu entscheiden, ob er diese Deadline einhalten kann.

   Satz 3.4.1. *Gegeben sei ein System $\mathcal{S}$ von Pfaden zusammen mit den Abhängigkeiten $<_{\mathfrak{A}}$ und $<$. Ein Scheduler findet garantiert einen Ablaufplan wenn gilt:*

   $$T_G \leq T_D$$

   Beweis. Klar. □

   Bemerkung 3.4.1. Diese Bedingung, die nichts anderes sagt, als dass die Vorberechnungszeit kleiner oder gleich der Deadline sein muss, ist offensichtlich notwendig und hinreichend, um das Finden eines Ablaufplanes zu garantieren. Das schließt gegebenenfalls mit ein, dass kein vollständiger Ablaufplan existiert, falls eine ausführbare Reihenfolge gar nicht gefunden werden kann. Man beachte auch, dass hier noch keinerlei Optimierung enthalten ist; bis zu diesem Zeitpunkt weiß man nur, ob das System ausführbar ist und wenn ja, wie die notwendigen Sequenzen auf den Pfaden beziehungsweise den Zyklen aussehen.

   Wenn nun die Eigenschaft aus Satz 3.4.1 gilt, dann kann die Zeit $T_O := T_D - T_G$ für eine Optimierung genutzt werden. Somit gilt $T_A = T_D$. Dieser Sachverhalt wird in Abbildung 3.19 verdeutlicht.

   Die Deadline T_D ist in diesem Fall eine *harte oder zumindest feste Zeitschranke*, die vom System oder der Anwendung diktiert wird.

2. Ein Anwender kann die Kenntnis der Vorberechnungszeit T_G und der Größenordnung der Zeit T_O einer Optimierungsstrategie nutzen, um den Scheduler sinnvoll zu konfigurieren. Da er die Anwendung kennt, die durch das System von Pfaden ausgeführt werden soll, kann er Abschätzungen treffen, nach welcher Zeit er die Ausführung der Pfade veranlassen will und auf dieser Abschätzung T_O und somit T_A dimensionieren. Auch dies ist in Abbildung 3.19 dargestellt.

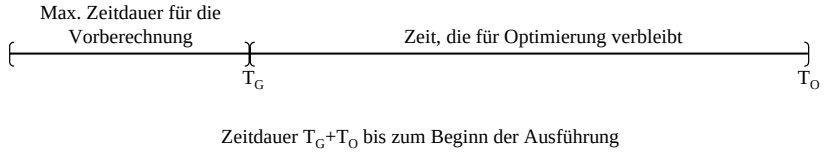


Abbildung 3.19.: Unterteilung der zur Verfügung stehenden Zeit für Vorberechnung und Optimierung

Zunächst müssen für die Berechnung von T_G und T_O einige Annahmen getroffen werden. Im Folgenden wird davon ausgegangen, dass der Zeitaufwand einer beliebigen Operation als reelles Vielfaches einer Basiszeitdauer n_0 angegeben werden kann. Sei nun

- $t_V n_0$ der Zeitaufwand, der für einen Vergleich benötigt wird. Dazu zählen Vergleiche von Matrixeinträgen, das Prüfen von Bedingungen bei der Ausführung eines Algorithmus und Vergleiche bezüglich der Relationen $<_{\mathfrak{A}}$, $<$ und $<_{\sim}$.
- $t_M n_0$ der Zeitaufwand für die Ausführung einer arithmetischen Operation.
- $t_Z n_0$ der Zeitaufwand für das Hinzufügen eines Elementes zu einer Liste.
- $t_L n_0$ der Zeitaufwand zum Löschen eines Elementes aus einer Matrix.

3.4.1. Abschätzung der WCET für die Vorberechnungszeit

Damit wird zunächst die WCET T_G, die der Scheduler für die Vorberechnung höchstens benötigt, abgeschätzt. Dabei werden für die Berechnung der oberen Schranken der Vorberechnungszeiten für die Algorithmen die dort jeweils angegebenen maximal benötigten Operationen angenommen.

Satz 3.4.2. *Sei $\mathcal{S}$ ein System von Pfaden. Sei $||\mathcal{S}||_{\mathfrak{A}} = k$ die Anzahl der darin enthaltenen Aufträge und $||\mathcal{S}|| = n$ die Anzahl der darin enthaltenen Pfade. Die Zeit $T_G^{\max}$ definiert durch*

$$T_G^{\max} := T_V + T_M + T_Z + T_L \tag{3.10}$$

ist eine obere Schranke für die Zeit T_G, das heißt es gilt $T_G \leq T_G^{\max}$. Dabei ist

- $T_V = t_V n_0 \left(2k^2 + k + 3n^2 + 4n - 2 + n \left\lfloor \frac{n}{2} \right\rfloor + \frac{\left\lfloor \frac{n}{2} \right\rfloor}{2} - \frac{\left\lfloor \frac{n}{2} \right\rfloor^2}{2} + 2 \sum_{i=1}^{n-1} i(n-i) \right)$
- $T_M = t_M n_0 (k^3 + n^3 + 5n^2 + 2n)$
- $T_Z = 3 t_Z n_0 n$

- $T_L = t_L n_0 n^2$

Beweis. T_G ist nach Definition die Zeit, die für alle Schritte der Planung außer der Optimierung benötigt wird. Dabei muss beachtet werden, dass abhängig von der Irreflexivität von $<$ zwei verschiedene Wege der Planung gegangen werden: Im irreflexiven Fall wird nur der Aufwand für die Sequentialisierung der n Pfade benötigt. Im nicht irreflexiven Fall geht dieser Aufwand auch ein, aber hier werden nur die $n - m$ Repräsentantenpfade (m ist die Anzahl von Pfaden, die jeweils einem Repräsentantenpfad zugeordnet wurden) sequentialisiert. Für die Aufwandsabschätzung kann man sich daher Abhilfe verschaffen, indem man mit dem nicht irreflexiven Fall rechnet und dabei davon ausgeht, dass n Repräsentantenpfade sequentialisiert werden sollen. Dann ist der Aufwand des irreflexiven Falles in diesem Fall enthalten.

Nun kann man T_G nach oben abschätzen als Summe der Zeit für

- den Algorithmus von Warshall und die Prüfung von $<_{\mathfrak{A}}$ auf Irreflexivität. Dafür wird höchstens die Zeit $W_1 = (k^2 + k)t_V n_0 + k^3 t_M n_0$ benötigt.
- die Berechnung von $<$ auf $\mathcal{S}$, seine Prüfung auf Irreflexivität und das Ermitteln der paarweise parallel ausführbaren Pfade. Dafür wird höchstens die Zeit $W_2 = (k^2 + 3n^2 + n)t_V n_0 + (4n^2 + n^3)t_M n_0$ benötigt.
- das Finden von Zyklen nach Algorithmus 4. Dafür wird höchstens die Zeit

  $$\left(2(n-1) + n\left\lfloor\frac{n}{2}\right\rfloor + \frac{\left\lfloor\frac{n}{2}\right\rfloor}{2} - \frac{\left\lfloor\frac{n}{2}\right\rfloor^2}{2}\right) t_V n_0 \quad + \left(\frac{n^2-n}{2} + \frac{n^2+n}{2} + m\right) t_M n_0 + m' t_Z n_O$$

  benötigt. Hierbei bezeichnet m die Anzahl von Pfaden, die jeweils einem Repräsentantenpfad zugeordnet wurden, und m' bezeichnet die Anzahl von Pfaden in Zyklen, siehe Algorithmus 4. Beide Größen können durch die Gesamtanzahl n von Pfaden im System abgeschätzt werden und es ergibt sich:

  $$W_3 = \left(2(n-1) + n\left\lfloor\frac{n}{2}\right\rfloor + \frac{\left\lfloor\frac{n}{2}\right\rfloor}{2} - \frac{\left\lfloor\frac{n}{2}\right\rfloor^2}{2}\right) t_V n_0 + (n^2+n) t_M n_0 + n t_Z n_0$$

- das Aufstellen des Systems $\mathcal{S}_\sim$ und die Berechnung von $<_\sim$ darauf durch Algorithmus 5. Dafür wird höchstens die Zeit $m' t_V n_0 + (n-m) t_M n_0 + (mn + m(n-m)) t_L n_0$ benötigt. Wenn man m und m' abschätzt, so ergibt sich eine maximale Ausführungszeit für Algorithmus 5 zu $W_4 = n t_V n_0 + n t_M n_0 + n^2 t_L n_0$.
- das Sequentialisieren nach Algorithmus 2. Unter Beachtung des eingangs zu diesem Beweis Gesagten ergibt sich eine obere Zeitschranke für diesen Algorithmus zu $W_5 = 2\left(\sum_{i=1}^{n-1} i(n-i)\right) t_V n_0 + 2n t_Z n_0$.

Für diese Abschätzung gilt:

$$
\begin{aligned}
W_1 + W_2 + W_3 + W_4 + W_5 = \\
& t_V n_0 \left(2k^2 + k + 3n^2 + 4n - 2 + n \left\lfloor \frac{n}{2} \right\rfloor + \frac{\left\lfloor \frac{n}{2} \right\rfloor}{2} - \frac{\left\lfloor \frac{n}{2} \right\rfloor^2}{2} + 2 \sum_{i=1}^{n-1} i(n-i) \right) \\
& + t_M n_0 (k^3 + n^3 + 5n^2 + 2n) \\
& + 3 t_Z n_0 n \\
& + t_L n_0 n^2
\end{aligned}
$$

Diese Summe ist gleich $T_G^{\max}$ und aufgrund der Herleitung ist klar, dass $T_G \leq T_G^{\max}$ gilt. □

Satz 3.4.3. *Gegeben sei ein System $\mathcal{S}$ von Pfaden zusammen mit den Abhängigkeiten $<_{\mathfrak{A}}$ und $<$. Ein Scheduler findet garantiert einen Ablaufplan wenn gilt:*

$$T_G^{\max} \leq T_D$$

Beweis. Wegen Satz 3.4.2 gilt $T_G \leq T_G^{\max} \leq T_D$, und die Behauptung folgt dann aus Satz 3.4.1. □

3.4.2. Abschätzung der WCET für die Optimierungszeit

Nun wird noch eine Aussage über die WCET T_O für die Optimierung benötigt. Dies ist aufgrund der gewählten Optimierungsstrategie und der variablen Anzahl von Optimierungsschritten nicht ohne Weiteres möglich. Im Gegensatz zur feststehenden Vorberechnungszeit kann hier allerdings auf die beiden oben erwähnten Varianten der harten Echtzeit und der Einstellung einer Optimierungszeit für den Scheduler eingegangen werden.

3.4.2.1. Die Optimierungszeit T_O bei harten Echtzeitschranken

Bei harten Echtzeitschranken ist eine Deadline T_D für die Planerstellung gegeben. Um zu prüfen, ob überhaupt Zeit für eine Optimierung bleibt, muss notwendigerweise mindestens $T_G \leq T_D$ nach Satz 3.4.1 gelten. Da man aber T_G a priori nicht kennt, muss man verstärkend voraussetzen, dass $T_G^{\max} \leq T_D$ nach Satz 3.4.3 gilt, um gesicherte Aussagen über die Möglichkeit zur Optimierung zu treffen. Unter dieser Voraussetzung kann man davon ausgehen, dass für eine Optimierung des Planes höchstens die folgende Zeit zur Verfügung steht:

$$T_O := T_D - T_G^{\max} \tag{3.11}$$

Die Zeit T_O kann nun genutzt werden, um zufällige Anordnungen auf dem System $\mathcal{S}$ zu testen. Sobald die Zeit T_O aufgebraucht ist wird das Testen der Anordnungen abgebrochen.

Da man wieder a priori weiter keine Informationen über die Struktur des Systems $\mathcal{S}$ und seine Zyklen besitzt, wird in diesem Unterkapitel vom schlechtest möglichen Fall ausgegangen, nämlich, dass alle Pfade von $\mathcal{S}$ in einem einzigen Zyklus liegen. Im Folgenden wird daher die Zeit berechnet, die man für einen Optimierungsschritt auf einem solchen Zyklus benötigt.

1. Die erste Optimierungsstrategie war die Suche nach bekannten Zyklen und deren Auflösung in einer bereitstehenden Bibliothek. Da man für jedes Beispiel darin eine WCET angeben kann, nach der der jeweilige Zyklus erkannt worden ist, kann man die Zeit $t_B^{\max}(\alpha)$ als Maximum über diese einzelnen Zeiten in Abhängigkeit von der Anzahl α von Pfaden im Zyklus definieren; das bedeutet, dass $t_B^{\max}$ eine Funktion von $\mathbb{N}$ nach $\mathbb{R}$ ist, formal $t_B^{\max} : \mathbb{N} \to \mathbb{R}$. Hier wird angenommen, dass in der Bibliothek für jede darin enthaltene Zyklusgröße α der zugehörige Funktionswert $t_B^{\max}(\alpha)$ vorliegt. Dabei gelte $t_B^{\max}(\alpha) := 0$, falls kein Zyklus der Größe α in der Bibliothek existiert. Das ist sinnvoll, da dann auch keine Vergleiche mit Zyklen vorgenommen werden können.

   Es ist leicht möglich, die Funktion $t_B^{\max}$ für eine vorliegende Bibliothek zu berechnen: Wenn man Unterkapitel 3.2.4 mit den Beispielen und deren Auflösungen betrachtet, so wird im ersten Fall der Aufwand für die Erkennung und Auflösung eines bestimmten Zyklustyps angegeben. In den anderen Fällen muss die charakteristische Matrix untersucht werden, was jeweils l^2 Vergleichsoperationen (wobei l die Größe des Zyklus ist) kostet. Dies kann einfach in eine Vergleichszeit umgerechnet werden. Allgemein kann man also für jedes Muster V in der Bibliothek eine maximale endliche Zeit angeben, nach der dieses Muster erkannt wird (Vergleich der charakteristischen Matrix). Diese Zeit werde durch die Funktion t_B : Muster in Bibliothek $\longrightarrow \mathbb{R}$ beschrieben[2] und wieder mit dem Zeitfaktor n_0 skaliert.

   Die in der vorliegenden Arbeit betrachteten Beispiele zu Zyklen und deren Auflösung, siehe Unterkapitel 3.2.4, stellen typische Beispiele dar; sie reichen jedoch nicht als umfassende Bibliothek von Zyklen aus. Diese muss basierend auf einer Vielzahl von Anwendungsbeispielen erstellt werden, was kein Schwerpunkt dieser Arbeit ist. Aus diesem Grund wird $t_B^{\max}$ in dieser Arbeit nicht berechnet. Für die Zwecke hier reicht es zu sehen, dass $t_B^{\max}$ für jede Zyklusgröße existiert und beschränkt ist; sie existiert also insbesondere auch für $\alpha = n$ Pfade.

[2]Die Funktion $t_B^{\max}$ kann man dann leicht berechnen durch $t_B^{\max}(n) = \max_{\substack{V \in \text{Bibliothek} \\ \|V\|=n}} t_B(V)$.

2. Die zweite Optimierungsstrategie beinhaltete die Generierung einer zufälligen Anordnung der Pfade eines oder mehrerer Zyklen und die Anwendung von Algorithmus 6, um zu testen ob diese Anordnungen ausführbar sind.

   Man kann annehmen, dass die maximale Zeit zur Generierung einer Anordnung durch eine Funktion $t_A : \mathbb{N} \to \mathbb{R}$ beschrieben wird, die von der Anzahl der Aufträge im Zyklus abhängt und mit dem bekannten Zeitfaktor n_0 skaliert werden muss. Da sie eine Permutation berechnet, ist der Funktionswert für jedes $n \in \mathbb{N}$ endlich.

   Allgemein kann man also die Zeit zur Erzeugung einer Anordnung auf einem Zyklus Z mit $\|Z\|_{\mathfrak{A}} = \sum_{Y \in Z} |Y|$ Aufträgen durch $n_0 t_A(\|Z\|_{\mathfrak{A}})$ berechnen. Gemäß den Kommentaren zu Algorithmus 6 wird also maximal die Zeit $t_M n_0(\|Z\|_{\mathfrak{A}}^2 + \|Z\|_{\mathfrak{A}}^3)$ für die Matrixmodifiktionen (dabei werden hier die p_Z Anordnungen durch $\|Z\|_{\mathfrak{A}}^2$ Modifikationen der Matrix $\mathfrak{A}$ abgeschätzt) und die Zeit $t_V n_0 \|Z\|_{\mathfrak{A}}^2$ für die benötigten Vergleiche benötigt. Insgesamt wird in diesem Fall also die folgende maximale Zeit t_C für einen Optimierungsschritt berechnet:

$$t_C(\|Z\|_{\mathfrak{A}}) = n_0(t_A(\|Z\|_{\mathfrak{A}}) + t_M(\|Z\|_{\mathfrak{A}}^2 + \|Z\|_{\mathfrak{A}}^3) + t_V \|Z\|_{\mathfrak{A}}^2) \tag{3.12}$$

   Da hier ein Zyklus maximaler Größe mit allen k Aufträgen des Systems $\mathcal{S}$ betrachtet wird, kann man den Zeitaufwand wie folgt abschätzen:

$$t_C(k) = n_0(t_A(k) + t_M(k^2 + k^3) + t_V k^2) \tag{3.13}$$

Nun definiere man:

$$T_H(k) := \max\{t_B^{\max}(k),\, t_C(k)\}$$

Satz 3.4.4. *Sei $\mathcal{S}$ ein System von Pfaden mit $\|\mathcal{S}\|_{\mathfrak{A}} = k$ Aufträgen und für die Planerstellung gelte die harte Zeitschranke T_D. Wenn $T_O := T_D - T_G^{\max} \geq 0$ gilt, so können mindestens*

$$\#\textit{Optimierungsschritte} = \left\lfloor \frac{T_O}{T_H(k)} \right\rfloor$$

Optimierungsschritte bezogen auf einen Zyklus der Größe k durchgeführt werden.

Beweis. Klar nach dem oben Gesagten. □

Bemerkung 3.4.2. Durch diese Abschätzung wird ausgedrückt, wieviele Optimierungsschritte im schlimmsten Fall durchgeführt werden können, nämlich wenn alle Pfade aus $\mathcal{S}$ in einem einzigen Zyklus liegen. Dabei wird nicht unterschieden, ob ein Vergleich in der Bibliothek oder eine eigene Anordnung als Optimierungsschritt gewählt wird, das bleibt dem Scheduler überlassen.

3.4.2.2. Auslegung der Optimierungszeit T_O

Wenn keine harten Zeitschranken eingehalten werden müssen und der Benutzer die Optimierungszeit selber beeinflussen darf, so gibt es im Wesentlichen zwei Möglichkeiten dafür. Die eine Möglichkeit besteht darin, nach der Vorberechnung die Optimierung nicht sofort zu starten, sondern die Optimierungszeit auf der Kenntnis der Struktur des Systems $\mathcal{S}$ auszulegen, das heißt, Größen wie Anzahl und Größe von vorhandenen Zyklen einzubeziehen. Die zweite Möglichkeit besteht darin, die Optimierungszeit auf typischen Informationen über Anzahl und Größe von Zyklen zu berechnen, die man etwa durch Profiling der Pfade ähnlicher Anwendungen gewonnen hat.

In jedem der beiden Fälle sind also zusätzlich zur Systemgröße die eben erwähnten charakteristische Eigenschaften wie Anzahl und Größe von Zyklen bekannt. Wenn man nun annimmt, dass im System $\mathcal{S}$ genau q Zyklen Z_1, Z_2, ..., Z_q mit jeweils $\|Z_1\|$, $\|Z_2\|$, ..., $\|Z_q\|$ Pfaden und jeweils $\|Z_1\|_{\mathfrak{A}} = k_{Z_1}$, $\|Z_2\|_{\mathfrak{A}} = k_{Z_2}$, ..., $\|Z_q\|_{\mathfrak{A}} = k_{Z_q}$ Aufträgen vorliegen, so kann man für jeden Zyklus eine gewisse Zeit einräumen, die er für Vergleiche in der Bibliothek und für Optimierungsversuche durch Anordnen erhält. Dabei sollten zunächst die Optimierungsmuster aus der Bibliothek untersucht werden, und falls der Scheduler dort nicht fündig wird sollten Anordnungen erzeugt und getestet werden.

Daher kann man die Optimierungszeit für einen Zyklus Z_i ($1 \leq i \leq q$) als Summe der Zeiten für Vergleiche in der Bibliothek zusammen mit der Zeit für das Anordnen des Zyklus veranschlagen. Die Optimierungszeit $t_O(k_{Z_i})$ für einen Zyklus Z_i kann man nach eben Gesagtem und Gleichung (3.12) (Optimierungszeit bei Anordnungen) nun wie folgt bestimmen:

$$t_O(k_{Z_i}) = \sum_{\substack{V \in Bibliothek \\ \|V\|_{\mathfrak{A}} = k_{Z_i}}} t_B(V) + N_i n_0 (t_A(k_{Z_i}) + t_M(k_{Z_i}^2 + k_{Z_i}^3) + t_V k_{Z_i}^2) \tag{3.14}$$

Dabei ist N_i ein Skalierungsfaktor für den Anwender, der bestimmt wie lange Anordnungen getestet werden dürfen.

Die gesamte Optimierungszeit T_O kann nun als Summe der Optimierungszeiten der einzelnen Zyklen berechnet werden:

$$T_O := \sum_{i=1}^{q} t_O(k_{Z_i}) \tag{3.15}$$

Beispiel 3.4.1. *Dieses Beispiel soll den Zeitaufwand $T_G^{\max}$ und T_O verdeutlichen, der für eine Vorberechnung und Optimierung benötigt wird. Man gehe von einem System $\mathcal{S}$ mit* 150 *Pfaden aus, die zusammen aus* 4000 *Aufträgen bestehen. In $\mathcal{S}$ existieren fünf Zyklen Z_1, Z_2, Z_3, Z_4, Z_5 mit den Kardinalitäten $\|Z_1\| = \|Z_2\| = \|Z_3\| = 5$ Pfade, $\|Z_4\| = 10$ Pfade und $\|Z_5\| = 20$ Pfade. Hierbei gelte:*

- Z_1 *enthält genau* 100 *Aufträge.*
- Z_2 *enthält genau* 93 *Aufträge.*
- Z_3 *enthält genau* 120 *Aufträge.*
- Z_4 *enthält genau* 200 *Aufträge.*
- Z_5 *enthält genau* 450 *Aufträge.*

Insgesamt befinden sich also 45 *Pfade in Zyklen, die zusammen aus* 963 *Aufträgen bestehen.*

Nun müssen noch die Zeiten für die einzelnen Operationen betrachtet werden. Für die Basiszeit gelte $n_0 = 4ns = 4 * 10^{-9}s$. *Für die verschiedenen Operationen gelten folgende Skalierungsfaktoren*

- *für Vergleiche:* $t_V = 1$,
- *für Matrixmultiplikationen:* $t_M = 3$,
- *für die Zuordnung von einem nicht freien Pfad zu seinem Zyklus:* $t_Z = 2$ *und*
- *für das Löschen von einem Matrixeintrag* $t_L = 1,5$.

Die WCET $T_G^{\max}$ *ergibt sich nun nach Satz 3.4.2:*

$$\begin{aligned} T_G^{\max} &= T_V + T_M + T_Z + T_L \\ &= t_V n_0(32000000 + 4000 + 67500 + 600 - 2 + 11250 + 37,5 \\ &\quad - 2812,5 + 1124950) \\ &\quad + t_M n_0(64000000000 + 3375000 + 112500 + 300) \\ &\quad + 450 t_Z n_0 \\ &\quad + 22500 t_L n_0 \\ &= 33205523 t_V n_0 + 64003487800 t_M n_0 + 450 t_Z n_0 + 22500 t_L n_0 \\ &= 192043703573 n_0 \\ &\approx 768,2s \approx 12,8min \end{aligned} \tag{3.16}$$

Im schlimmsten Fall beträgt die Vorberechnungszeit $T_G^{\max}$ *hier also etwa* 13 *Minuten. Der Hauptanteil an dieser Zeit kommt dabei durch die arithmetischen Operationen des Algorithmus von Warshall bei der Berechnung von* $<_{\mathfrak{A}}$ *und* $<$ *zustande, die im schlimmsten Fall einen kubischen Aufwand verursachen. Die Anteile der anderen Operationen gehen darin mehr oder weniger unter. Diese Berechnung ist aber zwingend notwendig, um* $<_{\mathfrak{A}}$ *und* $<$ *genau zu kennen. Wenn man also keine Kenntnis von Satz 3.2.8 hätte, so wüsste man, dass für jede Optimierung dieses Pfadsystems im schlimmsten Fall ein Aufwand von knapp* 13 *Minuten für die Neuberechnung von*

$<_{\mathfrak{A}}$ *und* $<$ *benötigt würde, wobei die Zeit für die Erzeugung einer Anordnung und die resultierenden Modifikationen der Matrix* $\mathfrak{A}$ *nicht mit eingerechnet sind. Dabei ist dann nicht einmal sichergestellt, dass das System überhaupt ausführbar ist. Im nächsten Unterkapitel wird nun gezeigt, welche Ersparnis man durch Satz 3.2.8 gewinnt, aufgrund dessen Anordnungen zur Optimierung nur auf den Zyklen durchgeführt werden müssen.*

Nun zur Berechnung der WCET T_O*. Zunächst wird von einer Anwendung mit harten oder festen Zeitschranken ausgegangen. Also wird angenommen, dass ein Zyklus optimiert werden soll, der alle* 150 *Pfade des Systems beinhaltet. Für die Suche nach Mustern aus der Bibliothek wird hier angenommen, dass ein relativ geringer Vergleichsaufwand ausreicht und deshalb unerheblich ist (*$t_B^{\max}(150) \approx 0$*). Also wird nur der Zeitaufwand für einen Optimierungsschritt bestehend aus genau einer Anordnung des gesamten Systems berechnet. Unter der Annahme, dass für diese Anordnung bestehend aus allen Pfaden (*4000 *Aufträge) die Zeit* $n_0 t_A(150) = 8000n_0 = 8000 * 4 * 10^{-9}s = 0,000032s$ *benötigt wird, ergibt sich für die Optimierung:*

$$\begin{aligned} t_C(4000) &= 4 * 10^{-9}(8000 + 3(16000000 + 64000000000) + 16000000)s \\ &\approx 12,8min \end{aligned}$$

Nach den obigen Annahmen gilt also $T_O = t_C(4000)$ *und weiter* $T_O \approx T_G^{\max}$ *nach vorangehender Rechnung. Dies liegt daran, dass der größte Teil des Zeitaufwandes bei der Ausführung der arithmetischen Operationen des Algorithmus von Warshall benötigt wird, und dieser wird sowohl in der Vorberechnung als auch in der Optimierung mit harten Deadlines auf dem gesamten System* $\mathcal{S}$ *berechnet.*

Wenn man nun keine harten Deadlines einhalten muss und Zeit für eine Analyse von $\mathcal{S}$ *hat, so kann man wesentlich bessere Aussagen für die Anzahl der Optimierungen treffen. Es wird hier wieder davon ausgegangen, dass man die Zeit für Vergleiche in der Bibliothek vernachlässigen kann. Weiter gelte, dass die Zeiten für die Erzeugung einer Anordnung auf dem Zyklus* Z_1 *höchstens* $n_0 t_A(100) = 200n_0$*, für* Z_2 *höchstens* $186n_0$*, für* Z_3 *höchstens* $240n_0$*, für* Z_4 *höchstens* $400n_0$ *und für* Z_5 *höchstens* $900n_0$ *betragen und für die Skalierungsfaktoren* $N_1 = N_2 = N_3 = N_4 = N_5 = 1$ *gilt, dann ergeben sich für die fünf Zyklen in* $\mathcal{S}$ *die jeweils die folgenden Optimierungszeiten nach Gleichung* (3.14)*:*

- $t_O(k_{Z_1}) = n_0(200 + 3(100^2 + 100^3) + 100^2) = 3040200n_0 = 0,0121608s$
- $t_O(k_{Z_2}) = n_0(186 + 3(93^2 + 93^3) + 93^2) = 2447853n_0 = 0,009791412s$
- $t_O(k_{Z_3}) = n_0(240 + 3(120^2 + 120^3) + 120^2) = 5241840n_0 = 0,02096736s$
- $t_O(k_{Z_4}) = n_0(400 + 3(200^2 + 200^3) + 200^2) = 24160400n_0 = 0,0966416s$
- $t_O(k_{Z_5}) = n_0(900 + 3(450^2 + 450^3) + 450^2) = 274185900n_0 = 1,0967436s$

Die Gesamtoptimierungszeit ergibt sich in diesem Fall gemäß Gleichung (3.15) *zu* $T_O = \sum_{i=1}^{5} t_O(k_{Z_i}) = 1,236304772s \approx 1,24s$.

An dieser Rechnung werden nun zwei Tatsachen deutlich: Erstens ist die Abschätzung der Optimierungszeit T_O *bei harten Deadlines zu grob, was aber nicht besser geht, da man die Struktur von* $\mathcal{S}$ *bezüglich* $<$ *a priori nicht kennt. Zweitens gilt aber: Wenn man die Struktur von* $\mathcal{S}$ *kennt, dann kann man erhebliche Einsparungen durch Satz 3.2.8 erhalten. Das Beispiel zeigt, dass im Gegensatz zu einem Optimierungsschritt des ganzen Systems, der hier etwa* 13 *Minuten dauert, die Summe der Zeiten für jeweils einen Optimierungsschritt genau auf den Zyklen des Systems nur etwa* $1,3$ *Sekunden dauert. Man erzielt also in diesem Beispiel eine durchschnittliche Geschwindigkeitssteigerung um den Faktor* 590 *gegenüber der oben genannten, ersten Variante.*

Umgekehrt kann man sich zur Verdeutlichung auch anschauen, wieviele Optimierungsschritte durch Satz 3.2.8 und Gleichung (3.15) *in den* 768 *Sekunden der ersten Variante durchgeführt werden können. So könnte man in dieser Zeit beispielsweise* $N_4 = N_5 = 600$ *Optimierungsschritte für die Zyklen* Z_4 *und* Z_5, $N_1 = N_3 = 1000$ *Optimierungsschritte für die Zyklen* Z_1 *und* Z_3 *und* $N_2 = 1900$ *Optimierungsschritte für den Zyklus* Z_2 *durchführen. Nach Gleichung* (3.15) *ergibt sich dann für die Optimierungszeit* $T_O = 767,7629628s < 768s$. *Ohne diesen Satz könnte man also schlimmstenfalls einen Optimierungsschritt in dieser Zeit durchführen, durch die Hilfe des Satzes kann man auf jedem Zyklus mehrere Hundert Optimierungsschritte durchführen.*

3.5. Laufzeiteinsparungen

In diesem Unterkapitel soll der durch Satz 3.2.8 gewonnene Vorteil nochmals systematisch dargestellt werden. Dazu werden verschiedene Messungen durchgeführt, die dies demonstrieren. Es wird einerseits der Zeitaufwand dargestellt, der sich für eine Ausführbarkeitsprüfung ohne die Hilfe von Satz 3.2.8 ergeben würde, und andererseits wird der Zeitaufwand für die Ausführbarkeitsprüfung für verschiedene Zyklengrößen unter Verwendung des Satzes gezeigt.

Wie im letzten Unterkapitel wird auch hier immer (mit derselben Argumentation wie im letzten Beispiel) angenommen, dass keine Bibliotheken zur Optimierung von Zyklen existieren. Dies bedeutet, dass die entsprechenden Zeiten in den Formeln (3.13) und (3.15) zu Null angenommen werden. Weiterhin wird (wieder wie im letzten Beispiel) für alle Versuche angenommen, dass die Basiszeit $n_0 = 4ns = 4 * 10^{-9}s$ beträgt, und dass für die verschiedenen Operationen zur Optimierung wieder die Skalierungsfaktoren für Vergleiche $t_V = 1$ und für arithmetische Operationen $t_M = 3$ gelten. Die Bemerkungen 3.5.1 und 3.5.2 erläutern später, dass der Laufzeitgewinn unabhängig von diesen Konstanten ist. Für die Anordnungszeit t_k wird hier angenommen, dass sie linear in der Anzahl der Aufträge ist und das gilt $t_A(k) := 2k$.

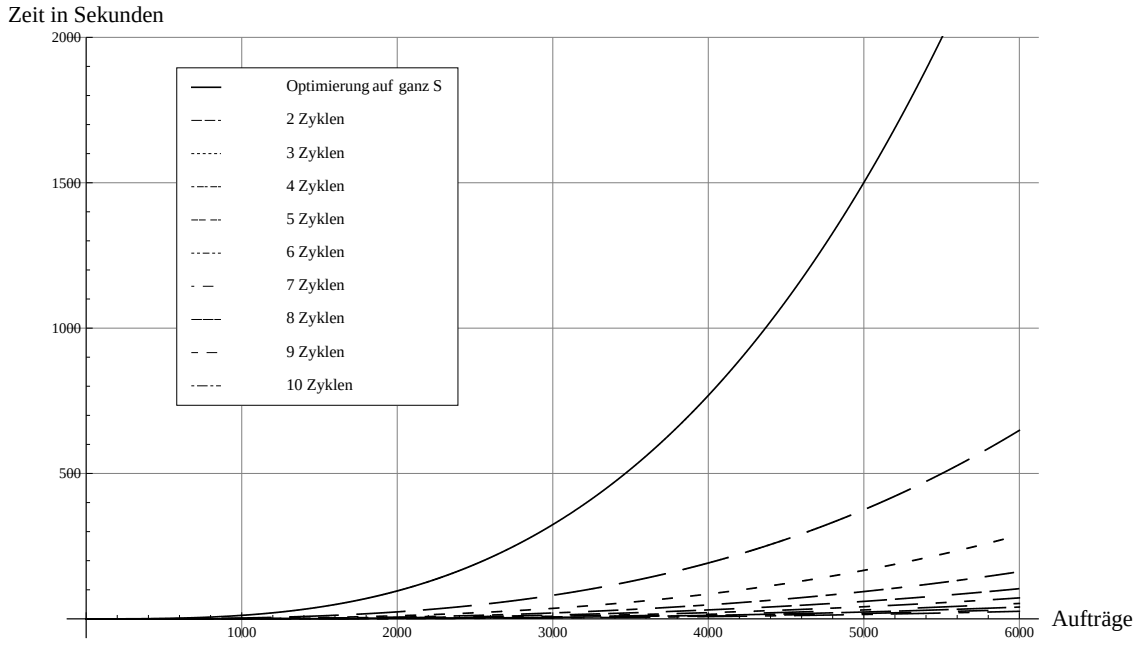


Abbildung 3.20.: Laufzeit der Optimierung

In Abbildung 3.20 werden die Laufzeiten für verschiedene Konfigurationen eines Systems $\mathcal{S}$ gezeigt. Dabei ist auf der Abszisse die Anzahl der im System $\mathcal{S}$ vorkommenden Aufträge aufgetragen, und auf der Ordinate wird die Zeit dargestellt. Das Schaubild zeigt die Ergebnisse von zehn Messungen. Zunächst wurde eine Referenzmessung (siehe Linie „Optimierung auf ganz $\mathcal{S}$“) durchgeführt unter der Annahme, dass $<_{\mathfrak{A}}$ auf allen Aufträgen von $\mathcal{S}$ neu berechnet werden muss. Dies modelliert eine Ausführbarkeitsprüfung nach einer versuchten Optimierung unter der Annahme, dass Satz 3.2.8 nicht zur Verfügung steht. Für die anderen Messreihen wurde angenommen, dass das System $\mathcal{S}$ aus jeweils entweder zwei oder drei usw. bis zehn Zyklen besteht (siehe Linien „2 Zyklen“, „3 Zyklen“, etc.), auf die die Aufträge gleichmäßig verteilt wurden[3] und auf denen man den Ausführbarkeitstest wegen Satz 3.2.8 einzeln durchführen kann. Weiterhin wurde hier die Annahme getroffen, dass alle Aufträge des Systems in Zyklen liegen, denn dies kennzeichnet den schlechtesten Fall für solch eine Verteilung. Auf der Ordinate ist also die Zeit aufgetragen, die benötigt wird, um die Ausführbarkeit eines Systems nach einem Anordnungsversuch zu überprüfen. Die Abbildung zeigt deutlich, dass der Ausführbarkeitstest auf

[3]Beispielsweise galt unter der Annahme, dass das System aus drei Zyklen besteht und insgesamt 80 Aufträge enthält, dass es zwei Zyklen zu jeweils 27 Aufträgen und einen Zyklus zu 26 Aufträgen gab.

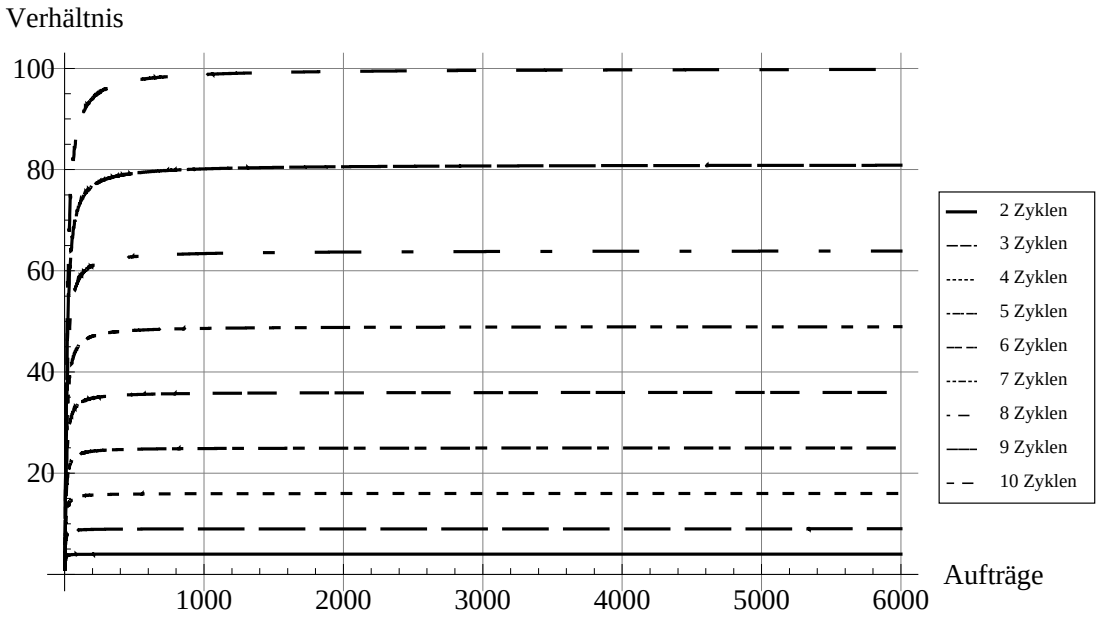


Abbildung 3.21.: Verhältnis der Laufzeiten der Optimierung

ganz $\mathcal{S}$ immer am meisten Zeit benötigt, und wegen dem kubischen Anteil in seiner Berechnung entfernt er sich auch immer mehr von den Kennlinien der anderen Messungen. Weiter ist klar zu sehen, dass ein Ausführbarkeitstest mit mehr Zyklen immer weniger Zeit in Anspruch nimmt. So benötigt ein Ausführbarkeitstest mit 5000 Aufträgen ohne Satz 3.2.8 1500 Sekunden (das sind 25 Minuten), während derselbe Test unter der Annahme, dass das System aus zwei Zyklen besteht, grade mal noch etwa 350 Sekunden (weniger als sechs Minuten) benötigt. Wenn das System aus mehr als vier Zyklen besteht, so wird der Test in weniger als 100 Sekunden, also weniger als zwei Minuten durchgeführt.

Es ist auch interessant, das sich das Verhältnis dieser Zeiten anzuschauen, also das Verhältnis „Optimierung auf ganz $\mathcal{S}$/Optimierung mit i Zyklen“ (die genaue Definition folgt weiter unten, siehe Gleichung (3.19)). Dies wird in Abbildung 3.21 dargestellt und zeigt, wieviele Optimierungen unter der Annahme, dass das System aus i Zyklen ($2 \leq i \leq 10$) besteht, in der Zeit einer Optimierung ohne Anwendung von Satz 3.2.8 stattfinden können. Die Abbildung zeigt, dass dieses Verhältnis beschränkt ist: So können etwa unter der Annahme, dass das System aus zwei Zyklen besteht, welche jeweils gleich viele Aufträge enthalten, vier Optimierungsversuche auf diesen zwei Zyklen in derselben Zeit durchgeführt werden wie ein Optimierungsversuch auf dem ganzen System (für große Anzahlen von Aufträgen). Wenn man die anderen Messlinien betrachtet, so stellt man fest, dass man unter diesen Voraussetzungen unter der Annahme, dass das System aus q Zyklen besteht, q^2 Optimierungsversuche

auf diesen durchführen kann in derselben Zeit, die für einen Optimierungsversuch auf dem ganzen System benötigt wird (für große Anzahlen von Aufträgen).

Dies kann man auch nachweisen: Unter der Annahme, dass das System aus q Zyklen[4] ($q \in \mathbb{N}$) besteht und k Aufträge ($k \in \mathbb{N}$) enthält, so kann man die oben erwähnte gleichmäßige Verteilung der Aufträge wie folgt beschreiben. Jeder Zyklus i ($1 \leq i \leq q$) enthält

$$k_{Z_i} = \|Z_i\|_{\mathfrak{A}} := k \div q + \left\lceil \frac{(k \bmod q) \div i}{q} \right\rceil \tag{3.17}$$

Aufträge. Dabei wird durch den ersten Summanden $k \div q$ die Anzahl von Aufträgen festgelegt, die jeder Zyklus mindestens enthält. Wenn noch Aufträge übrig bleiben (dies sind dann auf jeden Fall weniger als q Aufträge), dann wird den Zyklen durch den zweiten Summanden in Gleichung (3.17) beginnend ab $i = 1$ jeweils ein weiterer Auftrag zugeteilt, solange noch ein solcher übrig ist. Man kann nachrechnen, dass diese Aufteilung eine Partition der Zahl k ist und es gilt:

$$k = \sum_{i=1}^{q} \left(k \div q + \left\lceil \frac{(k \bmod q) \div i}{q} \right\rceil \right)$$

Weiterhin gilt für jedes Z_i ($1 \leq i \leq q$), $k \in \mathbb{N}$ und $q \in \mathbb{N}$ die Abschätzung:

$$\frac{k}{q} - 1 \leq k \div q + \left\lceil \frac{(k \bmod q) \div i}{q} \right\rceil \leq \frac{k}{q} + 1 \tag{3.18}$$

Das Verhältnis „Optimierung auf ganz $\mathcal{S}$/Optimierung mit q Zyklen“ ergibt sich damit gemäß den Formeln (3.13) und (3.15) unter der Annahme, dass nur die Optimierungszeiten für Anordnungen miteinbezogen werden zu:

$$Q_q(k) := \frac{n_0(2k + t_M(k^2 + k^3) + t_V k^2)}{\sum_{i=1}^{q} n_0(2k_{Z_i} + t_M(k_{Z_i}^2 + k_{Z_i}^3) + t_V k_{Z_i}^2)} \tag{3.19}$$

Nun definiere man

$$\begin{aligned} Q_q^-(k) : &= \frac{n_0(2k + t_M(k^2 + k^3) + t_V k^2)}{\sum_{i=1}^{q} n_0(2(\frac{k}{q} + 1) + t_M((\frac{k}{q} + 1)^2 + (\frac{k}{q} + 1)^3) + t_V(\frac{k}{q} + 1)^2)} \\ &= \frac{n_0(2k + t_M(k^2 + k^3) + t_V k^2)}{q n_0(2(\frac{k}{q} + 1) + t_M((\frac{k}{q} + 1)^2 + (\frac{k}{q} + 1)^3) + t_V(\frac{k}{q} + 1)^2)} \end{aligned} \tag{3.20}$$

[4]Man beachte, dass der Fall $q = 1$ gerade dem Referenzfall „Optimierung auf ganz $\mathcal{S}$“ entspricht.

und

$$Q_q^+(k) := \frac{n_0(2k + t_M(k^2 + k^3) + t_V k^2)}{\sum_{i=1}^{q} n_0(2(\frac{k}{q} - 1) + t_M((\frac{k}{q} - 1)^2 + (\frac{k}{q} - 1)^3) + t_V(\frac{k}{q} - 1)^2)}$$
$$= \frac{n_0(2k + t_M(k^2 + k^3) + t_V k^2)}{q n_0(2(\frac{k}{q} - 1) + t_M((\frac{k}{q} - 1)^2 + (\frac{k}{q} - 1)^3) + t_V(\frac{k}{q} - 1)^2)} \tag{3.21}$$

Wegen der Ungleichungen in (3.18) gilt offensichtlich für $k \in \mathbb{N}$ und $q \in \mathbb{N}$:

$$Q_q^-(k) \leq Q_q(k) \leq Q_q^+(k) \tag{3.22}$$

Weiterhin konvergieren Q_q^- und Q_q^+ für ein festes $q \in \mathbb{N}$ und es gilt:

$$\lim_{k \longrightarrow \infty} Q_q^-(k) = q^2 = \lim_{k \longrightarrow \infty} Q_q^+(k) \tag{3.23}$$

Aus (3.22) und (3.23) folgt nun mit [Heu98] Satz 22.2 die Konvergenz von Q_q und es gilt für ein festes $q \in \mathbb{N}$:

$$\lim_{k \longrightarrow \infty} Q_q(k) = q^2 \tag{3.24}$$

Bemerkung 3.5.1. Diese Rechnung zeigt nun auch formal die Rechenzeiteinsparungen, die man durch Satz 3.2.8 gewinnt: Wenn ein System aus q Zyklen besteht, in denen die Aufträge gleichverteilt sind, so kann man q^2 Optimierungsversuche für Zyklen in der Zeit einer einzigen Optimierung des ganzen Systems $\mathcal{S}$ durchführen. Dies ist sogar unabhängig von den vorkommenden Rechenkonstanten n_0, t_V und t_M. Die einzige Bedingung ist dabei, dass die Anordnungszeit t_A nicht mehr als quadratischen Aufwand benötigt.

Natürlich kann man eine Betrachtung des Verhältnisses „Optimierung auf ganz $\mathcal{S}$/Optimierung mit q Zyklen“ auch durchführen, wenn die Größen der einzelnen Zyklen nicht gleichverteilt sind, sondern proportional zu konstanten Bruchteilen der Systemgröße k sind (Trotzdem gilt weiterhin $k = \sum_{i=1}^{q} k_{Z_i}$, was bedeutet, dass angenommen wird, dass alle Aufträge in Zyklen liegen.). Dann erhält man Abschätzungen auf Basis der minimalen und maximalen Größen der Zyklen, um welchen Zeitfaktor man sich mindestens und höchstens verbessert. Dazu definiere man $k_{\min} := \min_{1 \leq i \leq q}\{k_{Z_i}\}$ und $k_{\max} := \max_{1 \leq i \leq q}\{k_{Z_i}\}$. Wegen dem eben Gesagten gibt es positive reelle Konstanten d und e mit:

$$k_{\min} = \frac{k}{d} \tag{3.25}$$

und

$$k_{\max} = \frac{k}{e} \tag{3.26}$$

Weiterhin gilt für $k \in \mathbb{N}$ und $q \in \mathbb{N}$:

$$\begin{aligned}
&\sum_{i=1}^{q} n_0(2k_{Z_i} + t_M(k_{Z_i}^2 + k_{Z_i}^3) + t_V k_{Z_i}^2) \\
&\quad = n_0 \left(2\sum_{i=1}^{q} k_{Z_i} + t_M \left(\sum_{i=1}^{q} k_{Z_i}^2 + \sum_{i=1}^{q} k_{Z_i}^3 \right) + t_V \sum_{i=1}^{q} k_{Z_i}^2 \right) \\
&\quad \leq n_0 \left(2\sum_{i=1}^{q} k_{Z_i} + t_M \left(\sum_{i=1}^{q} k_{Z_i} k_{\max} + \sum_{i=1}^{q} k_{Z_i} k_{\max}^2 \right) + t_V \sum_{i=1}^{q} k_{Z_i} k_{\max} \right) \\
&\quad = n_0 \left(2\sum_{i=1}^{q} k_{Z_i} + t_M \left(\left(\sum_{i=1}^{q} k_{Z_i} \right) k_{\max} + \left(\sum_{i=1}^{q} k_{Z_i} \right) k_{\max}^2 \right) \right. \\
&\qquad \left. + t_V \left(\sum_{i=1}^{q} k_{Z_i} \right) k_{\max} \right) \\
&\quad = n_0 \left(2ek_{\max} + t_M(ek_{\max}^2 + ek_{\max}^3) + t_V ek_{\max}^2 \right)
\end{aligned} \tag{3.27}$$

Analog erhält man eine umgekehrte Abschätzung für $k \in \mathbb{N}$ und $q \in \mathbb{N}$:

$$\begin{aligned}
&\sum_{i=1}^{q} n_0(2k_{Z_i} + t_M(k_{Z_i}^2 + k_{Z_i}^3) + t_V k_{Z_i}^2) \\
&\qquad \geq n_0(2dk_{\min} + t_M(dk_{\min}^2 + dk_{\min}^3) + t_V dk_{\min}^2)
\end{aligned} \tag{3.28}$$

Für die beiden Funktionen $Q_q^{\min}$ und $Q_q^{\max}$ definiert durch

$$Q_q^{\min}(k) := \frac{n_0(2k + t_M(k^2 + k^3) + t_V k^2)}{n_0\left(2ek_{\max} + t_M(ek_{\max}^2 + ek_{\max}^3) + t_V ek_{\max}^2\right)} \tag{3.29}$$

und

$$Q_q^{\max}(k) := \frac{n_0(2k + t_M(k^2 + k^3) + t_V k^2)}{n_0(2dk_{\min} + t_M(dk_{\min}^2 + dk_{\min}^3) + t_V dk_{\min}^2)} \tag{3.30}$$

gilt nun für $q \in \mathbb{N}$ und $k \in \mathbb{N}$:

$$Q_q^{\min}(k) \leq Q_q(k) \leq Q_q^{\max}(k) \tag{3.31}$$

Hier wird davon ausgegangen, dass gilt $k_{\min} \neq \emptyset$. Diese Annahme ist zulässig für große Werte von k, für die $Q_q^{\max}$ ja schließlich untersucht werden soll, und für ein konstantes d. Unter Benutzung von (3.25) und (3.26) sieht man, dass für ein festes $q \in \mathbb{N}$ gilt:

$$\lim_{k \longrightarrow \infty} Q_q^{\min}(k) = e^2 \tag{3.32}$$

und

$$\lim_{k \longrightarrow \infty} Q_q^{\max}(k) = d^2 \tag{3.33}$$

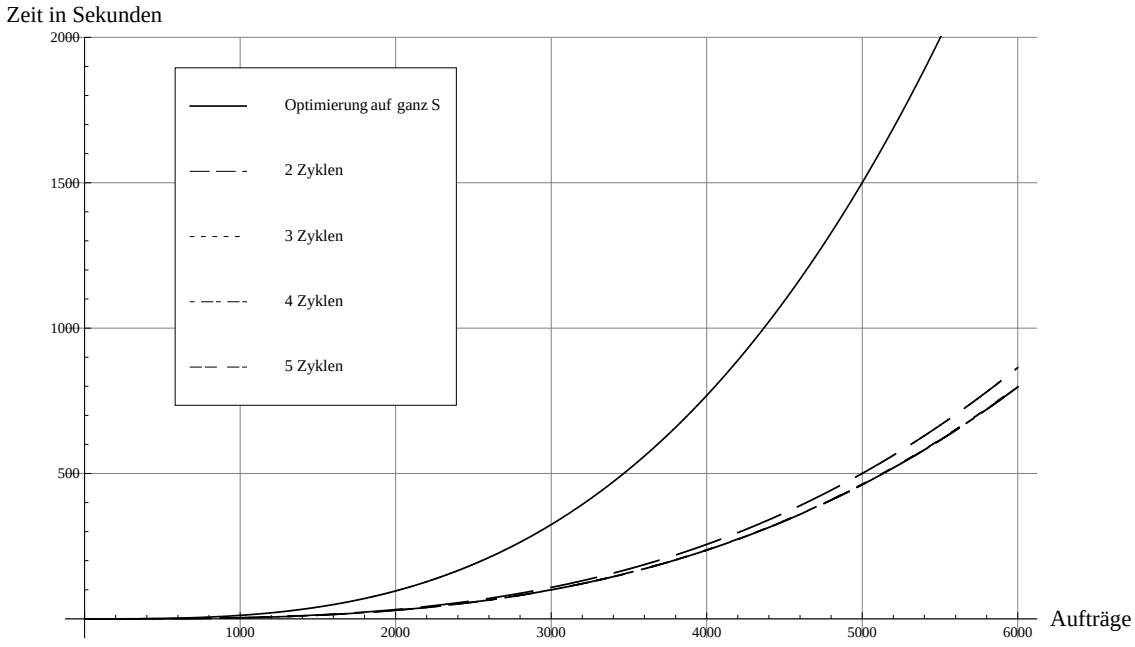


Abbildung 3.22.: Ausführbarkeitstest-Laufzeit bei Zyklen verschiedener Größe, $e = \frac{3}{2}$

Wegen den Definitionen von d und e in (3.25) und (3.26), aber auch wegen (3.31) und [Heu98] Satz 22.1 muss gelten $e^2 \leq d^2$. Daran sieht man, dass man in diesem allgemeinen Fall, wenn die Größen der Zyklen verschieden sein können, eine Verbesserung um den Zeitfaktor von mindestens e^2 und höchstens d^2 erhält.

Bemerkung 3.5.2. Aus dieser Abschätzung fällt als Spezialfall die Abschätzung in (3.24) heraus. Im Falle einer Gleichverteilung konvergieren d und e nämlich gegen q, was zu der eben erwähnten Abschätzung führt.

Weiterhin sieht man auch hier, dass die Ober- und Untergrenze der Leistungssteigerung jeweils unabhängig von den Optimierungskonstanten ist.

Die Abbildungen 3.22 und 3.23 zeigen Messungen mit Zyklen verschiedener Größen. Es wurde wieder eine Referenzmessung („Optimierung auf ganz $\mathcal{S}$“) durchgeführt, in der davon ausgegangen wird, dass das gesamte System aus einem einzigen Zyklus besteht. Dann wurden Messungen mit zwei, drei, vier und fünf Zyklen durchgeführt. Bei zwei Zyklen wurde angenommen, dass ein Zyklus zwei Drittel aller Aufträge enthält, während der andere Zyklus das andere Drittel enthält (Linie „2 Zyklen“). Bei drei Zyklen wurde angenommen, dass ein Zyklus zwei Drittel, ein weiterer Zyklus zwei Neuntel und der letzte Zyklus ein Neuntel aller Aufträge enthalten (Linie „3 Zyklen“). Diese Reihe wird für vier und fünf Zyklen analog

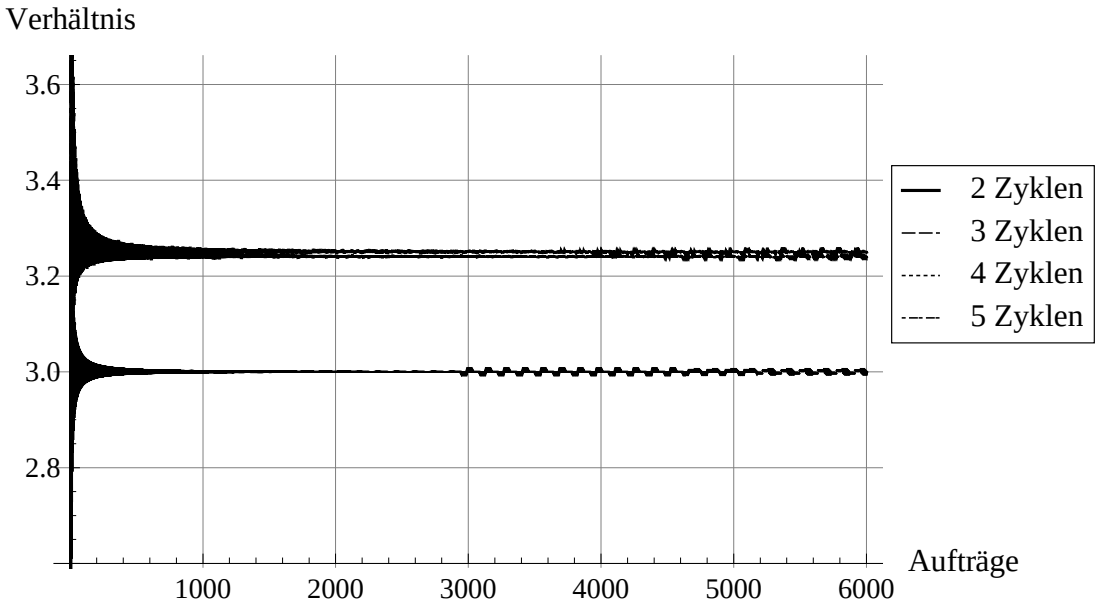


Abbildung 3.23.: Verhältnis der Ausführbarkeitstest-Laufzeiten bei Zyklen verschiedener Größe, $e = \frac{3}{2}$

fortgesetzt (siehe die Linien „4 Zyklen“ und „5 Zyklen“), wobei der vorletzte Zyklus immer zwei Drittel und der letzte Zyklus ein Drittel der Aufträge des letzten Zyklus der aktuellen Reihe enthalten. Auch bei diesen Messungen sieht man den sich durch Satz 3.2.8 ergebenden Vorteil deutlich. Während die Referenzmessung für die Ausführbarkeitsberechnung von 5000 Aufträgen wieder einen Aufwand von 25 Minuten ergab, werden für die Ausführbarkeitsberechnungen auf den Zyklen weniger als 500 Sekunden (das sind weniger als achteinhalb Minuten) benötigt. Auch ist deutlich zu sehen, dass die Aufteilung in immer mehr Zyklen bei dieser Versuchsanordnung keinen nennenswerten Vorteil mehr bringt, da die hinzukommenden Zyklen vergleichsweise wenige Aufträge enthalten, und die Größe des größten Zyklus konstant bleibt.

Abbildung 3.23 zeigt wieder das Verhältnis „Optimierung auf ganz $\mathcal{S}$/Optimierung mit q Zyklen“. Die Messlinien in der Abbildung überlagern sich. Deutlich zu erkennen ist die Messung für das Verhältnis mit „2 Zyklen“. Sie schwingt bei einem Wert von 3. Die Linien für die anderen Messungen liegen ineinander und schwingen bei $3,25$. Dies bedeutet, dass man durch Satz 3.2.8 eine Leistungssteigerung um mindestens den Faktor 3 erhält. In allen Messungen enthält der größte Zyklus zwei Drittel aller Aufträge, deshalb gilt $k_{\max} = \frac{2}{3}k = \frac{k}{\frac{3}{2}}$, woraus $e = \frac{3}{2}$ folgt. Die untere Schranke der Leistungssteigerung liegt somit gemäß Ausdruck (3.32) bei $e^2 = \left(\frac{3}{2}\right)^2 = 2,25$. Die oben erwähnte Leistungssteigerung von mindestens 3 bestätigt diese Grenze. Der

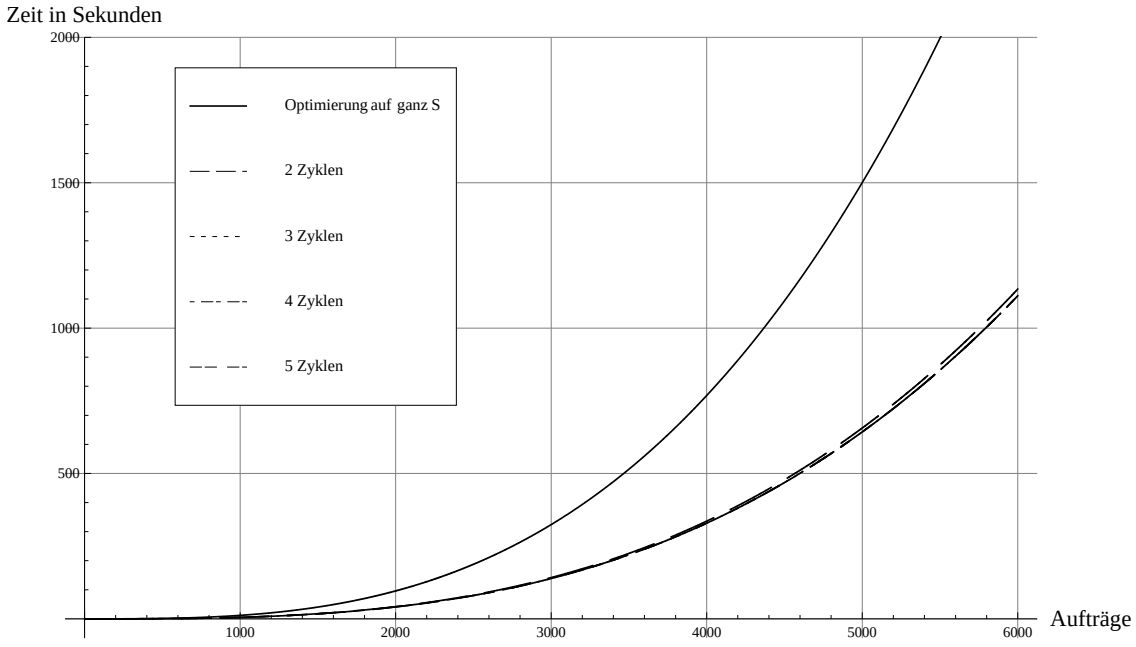


Abbildung 3.24.: Ausführbarkeitstest-Laufzeit bei Zyklen verschiedener Größe, $e = \frac{4}{3}$

kleinste Zyklus enthält je nach Messreihe ${}_1k_{\min} = \frac{k}{3}$, ${}_2k_{\min} = \frac{k}{9}$, ${}_3k_{\min} = \frac{k}{27}$ und ${}_4k_{\min} = \frac{k}{81}$ Pfade, wobei sich für die obere Schranke der Leistungssteigerung gemäß Ausdruck (3.33) die Werte $d_1^2 = 9$, $d_2^2 = 81$, $d_3^2 = 27^2$ und $d_4^2 = 81^2$ ergeben. Sie liegen damit weit über der tatsächlich erreichten Leistungssteigerung. Dies ist aber nicht weiter verwunderlich, da in dieser Abschätzung davon ausgegangen wird, dass das System aus lauter Zyklen von der Größe des kleinsten Zyklus besteht. Diese Abschätzung ist daher für das vorliegende Beispiel viel zu optimistisch.

Die Abbildungen 3.24 und 3.25 zeigen nochmals Messungen mit Zyklen verschiedener Größe. Hier wurde $k_{\max} = \frac{3}{4}k$ gewählt, also $e = \frac{4}{3}$. Das bedeutet, dass der größte Zyklus immer drei Viertel aller Aufträge enthält. Die Dimensionierung der Größen der restlichen Zyklen wurde analog zum letzten Beispiel durchgeführt. Auch hier zeigen sich Leistungssteigerungen durch das Einsparen der Optimierung auf allen Aufträgen des Systems. Diese sind allerdings erwartungsgemäß geringer als in obigem Beispiel, da der größte Zyklus drei Viertel der Aufträge des Systems enthält. Die Aufteilung in immer mehr Zyklen macht sich kaum noch bemerkbar, da die hinzukommenden Zyklen wieder vergleichsweise wenige Aufträge enthalten. Auch die durch (3.32) und (3.33) berechneten Schranken werden eingehalten.

Schließlich wird in Abbildung 3.26 die Zeit für eine Ausführbarkeitsprüfung in Abhängigkeit zur Anzahl der Aufträge (Definitionsbereich {1, ..., 500}) und zur An-

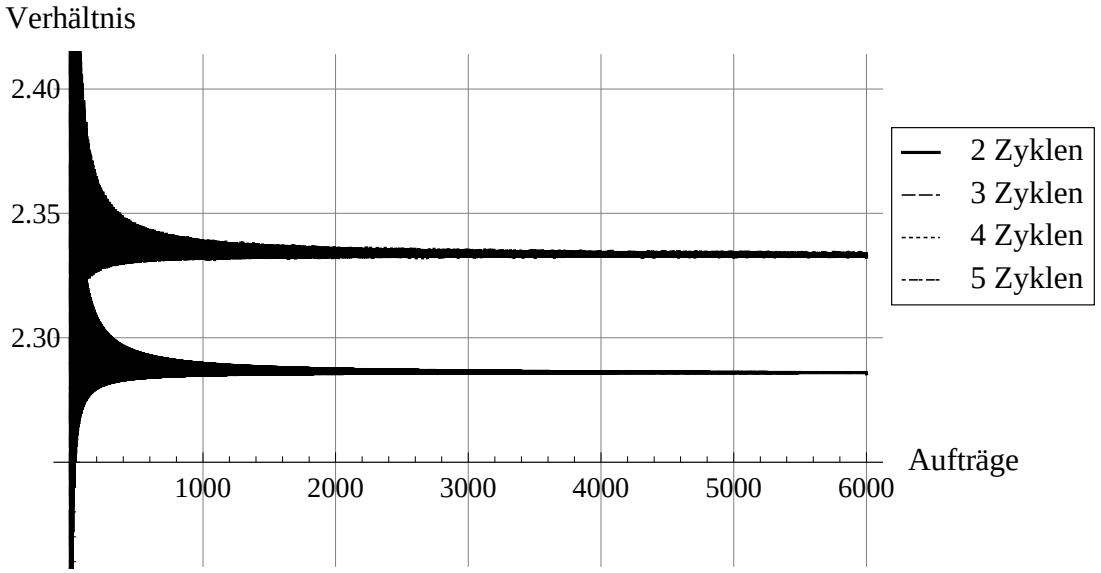


Abbildung 3.25.: Verhältnis der Ausführbarkeitstest-Laufzeiten bei Zyklen verschiedener Größe, $e = \frac{4}{3}$

zahl der vorkommenden Zyklen (Definitionsbereich {1, ..., 30}) gezeigt. Wie in der ersten Messreihe wird auch hier angenommen, dass die Aufträge gleichmäßig auf die vorhandenen Zyklen aufgeteilt werden. Die Abbildung stellt gut dar, wie die Prüfungszeit immer größer wird mit der wachsenden Anzahl von Aufträgen bis zu einer maximalen Zeit von $0,015$ Sekunden, siehe die grün, gelb und rot markierten Bereiche. Dieser Effekt wird abgeschwächt, wenn man eine wachsende Anzahl von Zyklen betrachtet, siehe den blau markierten Bereich. Auch die Abbildungen 3.27 und 3.28 zeigen die Zeit für eine Ausführbarkeitsprüfung in Abhängigkeit zur Anzahl der Aufträge (Definitionsbereich wieder {1, ..., 500}) und zur Anzahl der vorkommenden Zyklen (Definitionsbereich auch wieder {1, ..., 30}). Bei der Messung in Abbildung 3.27 werden die Aufträge wie im zweiten Beispiel beschrieben auf die Zyklen verteilt, das bedeutet, dass der größte Zyklus zwei Drittel aller Aufträge enthält und so weiter. Bei der Messung in Abbildung 3.28 werden die Aufträge wie im dritten Beispiel beschrieben auf die Zyklen verteilt, das bedeutet, dass der größte Zyklus drei Viertel aller Aufträge enthält und so weiter.

Der Verlauf der beiden letzten Messreihen ist sehr ähnlich. Trotz desselben Definitionsbereichs wird in der Messung in Abbildung 3.27 eine maximale Prüfungszeit von $0,4$ Sekunden benötigt, während bei der Messung in Abbildung 3.28 eine maximale Prüfungszeit von $0,6$ Sekunden benötigt wird. Beide Messungen zeigen deutlich, dass die Anzahl der Zyklen hier kaum eine Rolle für die Dauer der Messungen spielt,

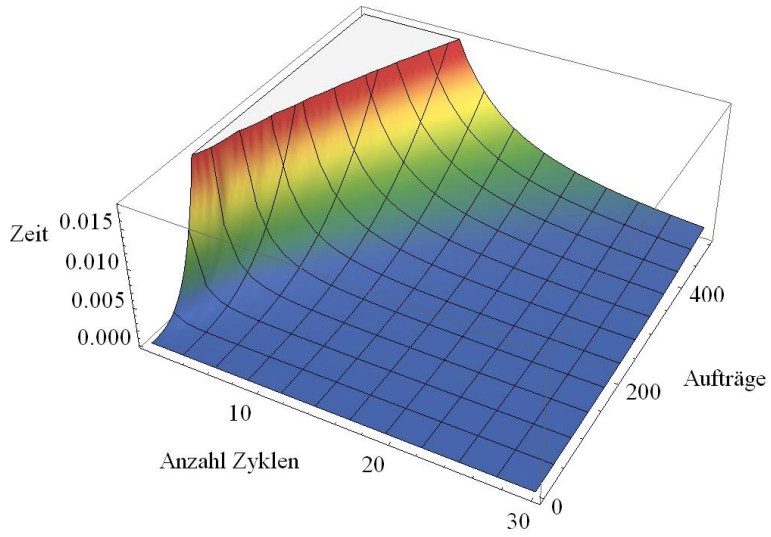


Abbildung 3.26.: Prüfungszeit in Abhängigkeit von Aufträgen und Zyklen bei Gleichverteilung

sondern fast ausschließlich die Anzahl der Aufträge. Der Grund dafür ist, wie oben bereits angedeutet, dass hier der größte Zyklus ausschlaggebend für die Prüfungszeit ist, und seine Größe ändert sich auch bei einer wachsenden Anzahl von Zyklen nicht. Dies erklärt auch die größere maximale Prüfungszeit bei der Messung in Abbildung 3.28, denn hier ist der größte Zyklus größer als in der anderen Messung.

Interessant ist auch ein Vergleich der letzten beiden Messungen mit der Messung bezüglich der Gleichverteilung der Aufträge in Abbildung 3.26. Zunächst sieht man, dass der Wertebereich bei den Messungen deutlich unterschiedlich ist (trotz desselben Definitionsbereichs aller Messungen). In der Messung bezüglich der Gleichverteilung der Aufträge ist die maximale Prüfungszeit um mindestens den Faktor 20 kleiner als in den anderen beiden Messungen. Dies liegt eben an der Gleichverteilung der Aufträge, was bedeutet, dass sich der größte Zyklus bei wachsender Anzahl von Zyklen und konstanter Anzahl von Aufträgen verkleinert.

3.6. Dynamische Änderungen am System

Nun soll noch untersucht werden, inwiefern ein Ablaufplan für ein bestehendes System $\mathcal{S}$ weiterverwendet werden kann, wenn Pfade aus dem System entfernt werden oder wenn Pfade zu dem System hinzukommen. Dabei ist mit dem *Entfernen* von einem oder mehreren Pfaden gemeint, dass das System $\mathcal{S}$ ohne eine bestimmte Anzahl von Pfaden betrachtet wird. $\mathcal{S}$ wird also auf ein System $\mathcal{S}^-$ reduziert.

Mit dem *Hinzufügen* von Pfaden ist gemeint, dass einer oder mehrere Pfade zum System $\mathcal{S}$ hinzugefügt werden und auch die Relationen $<_{\mathfrak{A}}$, $<$, $<_{\sim}$ und $<_D$ auf das

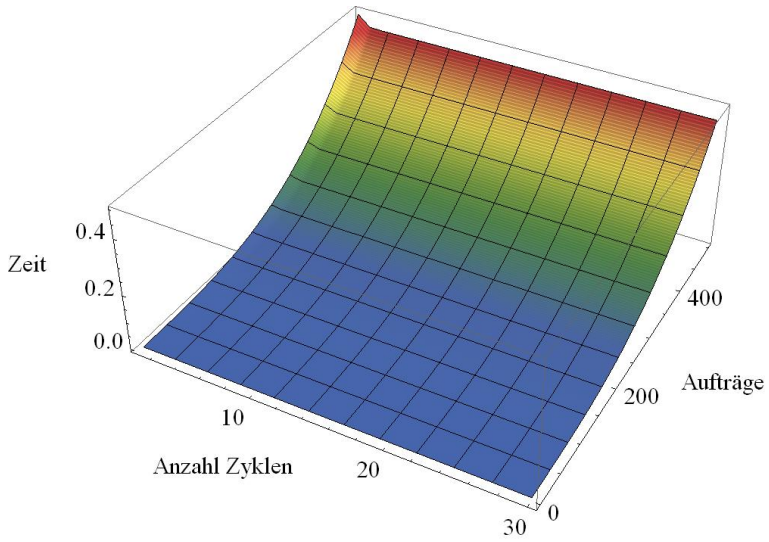


Abbildung 3.27.: Prüfungszeit in Abhängigkeit von Aufträgen und Zyklen bei Zwei-Drittel-Gewichtung

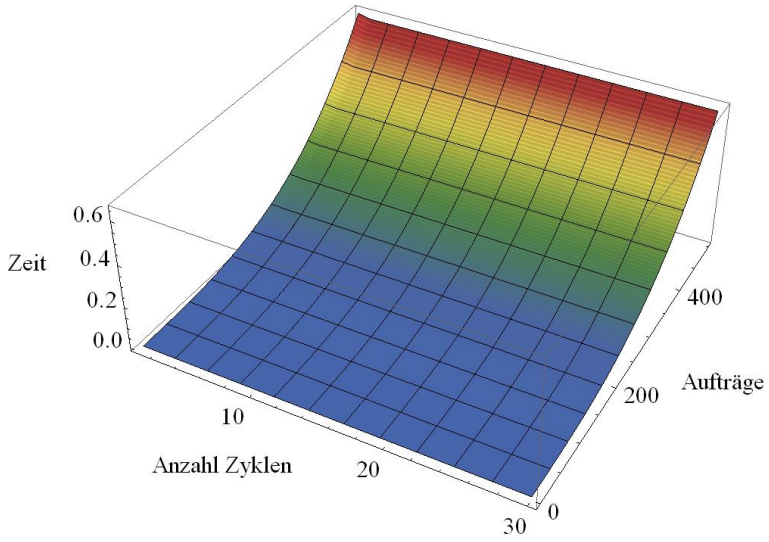


Abbildung 3.28.: Prüfungszeit in Abhängigkeit von Aufträgen und Zyklen bei Drei-Viertel-Gewichtung

neu entstehende System gemäß ihren Definitionen mit eingeschlossen werden. $\mathcal{S}$ wird also auf ein System $\mathcal{S}^+$ erweitert.

Es ist klar, dass man in beiden Fällen eine komplett neue Untersuchung der Struktur der Pfade durchführen könnte. Hier wird nun aber untersucht, ob und wenn ja, wie man einen gegebenen Ablaufplan weiterverwenden kann.

Natürlich muss man beim Entfernen oder Hinzufügen von Pfaden beachten, dass die Semantik des Systems in dem Sinne verändert wird, den man als Benutzer anstrebt. Dies ist jedoch nicht Teil dieser Untersuchung.

3.6.1. Entfernen von Pfaden

Wenn man eine Anzahl von Pfaden aus einem System $\mathcal{S}$ entfernt, so gibt es zwei Möglichkeiten, die Relation $<_{\mathfrak{A}}$ zu betrachten. Die eine Möglichkeit besteht darin, das reduzierte System $\mathcal{S}^-$ mit $<_{\mathfrak{A}}$ gemäß Definition 2.2.2 neu zu berechnen. Die andere Möglichkeit besteht darin, die Einschränkung von $\mathcal{S}$ auf $\mathcal{S}^-$ zu betrachten. Zu den Unterschieden der beiden Vorgehensweisen vergleiche Anhang A.5. Welche der beiden Möglichkeiten man anwendet ist anwendungsabhängig.

Satz 3.6.1. *Gegeben sei ein ausführbares System durch* $(\mathcal{S}, \mathcal{S}_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$ *und ein reduziertes System durch* $(\mathcal{S}^-, \mathcal{S}^-_{\mathfrak{A}}, <_{\mathfrak{A}}, <)$. *Dann ist* $\mathcal{S}^-$ *ausführbar.*

Beweis. Annahme, $\mathcal{S}^-$ wäre nicht ausführbar. Dann ist $\mathcal{S}$ auch nicht ausführbar, da $<_{\mathfrak{A}}$ in $\mathcal{S}^-_{\mathfrak{A}}$ eine Teilmenge von $<_{\mathfrak{A}}$ in $\mathcal{S}_{\mathfrak{A}}$ ist, Widerspruch. □

Bemerkung 3.6.1. Man beachte, dass es für diesen Satz unerheblich ist, welche der beiden Möglichkeiten zur Reduktion eines Pfadsystems gewählt wird.

Im Beweis dieses Satzes steckt die wichtige Aussage, dass $<_{\mathfrak{A}}$ in $\mathcal{S}^-_{\mathfrak{A}}$ eine Teilmenge von $<_{\mathfrak{A}}$ in $\mathcal{S}_{\mathfrak{A}}$ ist. Da die Relationen $<, <_\sim$ und $<_D$ aber alle auf $<_{\mathfrak{A}}$ aufbauen, überträgt sich diese Inklusion auf sie. Dies bedeutet aber, dass man den Ablaufplan für das System $\mathcal{S}$ einfach anpassen kann:

- Wenn einer oder mehrere freie Pfade entfernt werden, so können sie bei der Abarbeitung des Ablaufplanes einfach ignoriert werden.
- Dasselbe gilt auch für nicht freie Pfade. Dabei mache man sich klar, dass eventuelle Optimierungen schon im ausführbaren System $\mathcal{S}$ enthalten sind, was bedeutet, dass man sich nicht um deren Auswirkung auf andere Zyklen kümmern muss.

Der Vorteil dieser Vorgehensweise liegt darin, dass nur die Relation $<_{\mathfrak{A}}$ neu berechnet werden muss; dies ist besonders einfach, wenn man für die Durchführung einer Reduktion die Einschränkung wählt. Der Nachteil dieser Vorgehensweise liegt darin, dass Optimierungen, die durch eine Analyse der Struktur von $\mathcal{S}^-$ ermöglicht

werden, so nicht erkannt werden, und möglicherweise eine schnellere Abarbeitung des Systems durch eine stärkerere Parallelisierung als in $\mathcal{S}$ nicht möglich ist. Hier muss ein Nutzer abwägen, ob $\mathcal{S}^-$ schnell ausgeführt werden soll, oder ob sich der Zeitaufwand für eine Neuberechnung lohnt.

3.6.2. Hinzufügen von Pfaden

Das Hinzufügen von Pfaden zu einem System $\mathcal{S}$ stellt sich nicht so einfach dar und muss zunächst einmal definiert werden.

Definition 3.6.1. *Sei $\mathcal{S}$ ein System von n Pfaden X_1, ..., X_n und seien Y_1, ..., Y_m m Pfade, die nicht in $\mathcal{S}$ liegen. Werde mit x_β^α und y_κ^ι der Auftrag mit der Nummer β des Pfades X_α aus $\mathcal{S}$ und der Auftrag mit der Nummer κ des Pfades Y_ι bezeichnet. Weiter müssen jeweils paarweise x_β^α, ..., x_δ^γ auf y_κ^ι, ..., y_μ^λ und y_ξ^ν, ..., y_ρ^π auf x_ζ^ϵ, ..., x_θ^η und y_τ^σ, ..., y_ϕ^φ auf y_ω^χ, ..., $y_\varkappa^\Gamma$ warten. Dabei seien die Kardinalitäten der Mengen aufeinander wartender Aufträge gleich. Dann ist $<_{\mathfrak{A}^*}$ die Relation mit den Eigenschaften von Definition 2.2.2, die die Relation $<_{\mathfrak{A}}$ enthält und für die $y_\kappa^\iota <_{\mathfrak{A}^*} x_\beta^\alpha$, ..., $y_\mu^\lambda <_{\mathfrak{A}^*} x_\delta^\gamma$ und $x_\zeta^\epsilon <_{\mathfrak{A}^*} y_\xi^\nu$, ..., $x_\theta^\eta <_{\mathfrak{A}^*} y_\rho^\pi$ und $y_\omega^\chi <_{\mathfrak{A}^*} y_\tau^\sigma$, ..., $y_\varkappa^\Gamma <_{\mathfrak{A}^*} y_\phi^\varphi$ gilt.*

Bemerkung 3.6.2. Da für die Formulierung dieses Satzes viele Indices benötigt werden, wurden für diesen Zweck Buchstaben aus dem griechischen Alphabet gewählt. Es werde dabei vorausgesetzt, dass sich jede Indexvariable innerhalb ihres Wertebereichs bewegt.

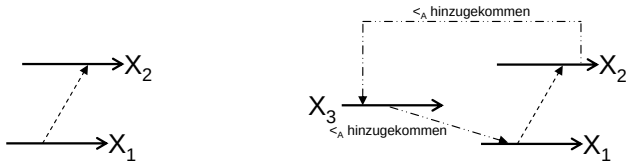
Bemerkung 3.6.3. Die *Existenz* und *Eindeutigkeit* der Relation $<_{\mathfrak{A}^*}$ sind bei einer Erweiterung klar, vergleiche Definitionen 3.2.8 und 3.2.9. Genauso gilt definitionsgemäß die *Transitivität*.

Anschaulich werden durch diese Definition neue Abhängigkeiten zwischen den neuen Pfaden und den Pfaden von $\mathcal{S}$ und zwischen den neuen Pfaden untereinander zugelassen. Neue Abhängigkeiten zwischen den Pfaden aus $\mathcal{S}$ können daher nur durch Transitivität zustande kommen.

Es werde vereinbart, dass o.B.d.A. $<_{\mathfrak{A}^*}$ nach einer Erweiterung der Einfachheit halber wieder mit $<_{\mathfrak{A}}$ bezeichnet werde und das erweiterte System mit ($\mathcal{S}^+$, $\mathcal{S}^+_{\mathfrak{A}}$, $<_{\mathfrak{A}}$, $<$) bezeichnet werde. Als Abkürzung kann man vom erweiterten System $\mathcal{S}^+$ sprechen.

Die Aufführbarkeit des erweiterten Systems $\mathcal{S}^+$ muss nun mittels Satz 2.3.2 geprüft werden.

Beispiel 3.6.1. *Abbildung 3.29 zeigt ein ausführbares System $\mathcal{S}$, welches durch Hinzufügen eines einzigen Pfades zu einem nicht ausführbaren System $\mathcal{S}^+$ erweitert wird. Hierbei wurde bei $\mathcal{S}^+$ auf die Darstellung der Transitivität von $<_{\mathfrak{A}}$ der Übersicht halber verzichtet.*



(a) System S **(b) Erweitertes System S⁺**

Abbildung 3.29.: System, das durch Erweitern nicht ausführbar wird

Wenn $\mathcal{S}^+$ ausführbar ist, dann müssen nun die Relationen $<$ und $<_\sim$ neu berechnet werden. Insofern besteht also keine Möglichkeit Rechenzeit zu sparen, da diese Schritte notwendig sind, denn das Hinzufügen von neuen Pfaden zu einem System kann ganz neue Relationen zwischen Aufträgen einführen. Dies bedeutet, dass die Struktur des Systems $\mathcal{S}^+$ komplett neu berechnet werden muss.

Wenn allerdings Zyklen aus dem System $\mathcal{S}$ vollständig und ohne Erweiterung in $\mathcal{S}^+$ auftauchen, dann ist es möglich, Zeit für die Optimierung der Ausführung von Zyklen zu sparen.

Satz 3.6.2. *Sei $\mathcal{S}$ ein ausführbares System von Pfaden und $\mathcal{S}^+$ eine ausführbare Erweiterung von $\mathcal{S}$. Seien X_1, ..., X_n nicht freie Pfade in $\mathcal{S}$, die darin paarweise zu verschiedenen Zyklen gehören, und seien ${}_{\mathcal{S}}\zeta(X_1)$, ..., ${}_{\mathcal{S}}\zeta(X_n)$ die zugehörigen Zyklen in $\mathcal{S}$ und ${}_{\mathcal{S}^+}\zeta(X_1)$, ..., ${}_{\mathcal{S}^+}\zeta(X_n)$ die zugehörigen Zyklen in $\mathcal{S}^+$. Es gelte ${}_{\mathcal{S}}\zeta(X_1) = {}_{\mathcal{S}^+}\zeta(X_1)$, ..., ${}_{\mathcal{S}}\zeta(X_n) = {}_{\mathcal{S}^+}\zeta(X_n)$. Dann ist jede ausführbare Anordnung der Zyklen ${}_{\mathcal{S}}\zeta(X_i)$ $(1 \leq i \leq n)$ in $\mathcal{S}$ gemäß Satz 3.2.8 auch eine ausführbare Anordnungen von ${}_{\mathcal{S}^+}\zeta(X_i)$ in $\mathcal{S}^+$.*

Beweis. Wegen ${}_{\mathcal{S}}Z\zeta(X_i) = {}_{\mathcal{S}^+}\zeta(X_i)$ $(1 \leq i \leq n)$ sind die Pfade der jeweiligen Zyklen identisch. Die Relation $<_{\mathfrak{A}}$ ist jeweils auf den Einschränkungen auf diese Zyklen, also auf ${}_{\mathcal{S}}\zeta_|(X_i)$ und ${}_{\mathcal{S}^+}\zeta_|(X_i)$ $(1 \leq i \leq n)$ identisch, denn für alle i zwischen 1 und n gilt: Wenn x und y zwei Aufträge aus ${}_{\mathcal{S}}\zeta(X_i)$ beziehungsweise ${}_{\mathcal{S}^+}\zeta(X_i)$ sind mit $x <_{\mathfrak{A}} y$ in ${}_{\mathcal{S}}\zeta_|(X_i)$, dann gilt dies offensichtlich auch in ${}_{\mathcal{S}^+}\zeta_|(X_i)$. Wenn $x <_{\mathfrak{A}} y$ in ${}_{\mathcal{S}^+}\zeta_|(x_i)$ gilt, dann gilt dies auch in ${}_{\mathcal{S}}\zeta_|(X_i)$. Annahme dies würde nicht gelten: Dann kann $x <_{\mathfrak{A}} y$ in ${}_{\mathcal{S}^+}\zeta_|(X_i)$ nach Definition 3.6.1 nicht auf direktem Wege entstanden sein, da die einzelnen Zyklen nach Voraussetzung nicht erweitert wurden. Also muss diese Beziehung durch Transitivität entstanden sein, und zwar durch einen der neu hinzugekommenen Pfade. Dann gibt es aber einen Auftrag z auf einem Pfad Z, welcher nicht in ${}_{\mathcal{S}^+}\zeta(X_i)$ liegt mit $x <_{\mathfrak{A}} z <_{\mathfrak{A}} y$, woraus folgt, dass Z doch in ${}_{\mathcal{S}^+}\zeta(X_i)$ liegt, Widerspruch. Die Behauptung des Satzes folgt nun unmittelbar aus Satz 3.2.8. □

Bemerkung 3.6.4. Da $\mathcal{S}^+$ eine Erweiterung von $\mathcal{S}$ ist und die dortige Relation $<_{\mathfrak{A}}$ höchstens erweitert, aber nicht eingeschränkt wird, gilt offensichtlich ${}_{\mathcal{S}}\zeta(X_i) \subseteq$

$_{\mathcal{S}^+}\zeta(X_i)$ $(1 \leq i \leq n)$. Daher bedeutet die Forderung der Gleichheit von sich entsprechenden Zyklen in den beiden Systemen lediglich, dass ein Zyklus durch die Erweiterung nicht vergrößert wird.

Der Gewinn der Aussage dieses Satzes ist, dass man Anordnungen auf Zyklen, die sich in ihrer Größe bei einer Erweiterung nicht verändert haben, weiterverwenden kann und keine neue Optimierung durchführen muss. Man beachte dabei allerdings, dass die Ordnung, die durch $<$ in $\mathcal{S}^+$ gegeben ist, durch eine Erweiterung verändert worden sein kann im Vergleich zur ursprünglichen Ordnung von $<$ in $\mathcal{S}$.

Definition 3.6.1 lässt nur neue Beziehungen auf direktem Wege zwischen Aufträgen des Systems $\mathcal{S}$ und neuen Pfaden und von Aufträgen neuer Pfade unter sich zu. Man könnte die Definition auch allgemeiner fassen und beliebige Neudefinitionen von $<_{\mathfrak{A}}$ zulassen. Um einen analogen Satz zu Satz 3.6.2 zu formulieren müsste darin zusätzlich die Bedingung hinzugefügt werden, dass $<_{\mathfrak{A}}$ in $\mathcal{S}$ auf der Einschränkung auf die gleich gebliebenen Zyklen identisch zu $<_{\mathfrak{A}}$ in $\mathcal{S}^+$ auf der Einschränkung auf die gleich gebliebenen Zyklen ist. Dann würde die Aussage dieses Satzes auch unter dieser Definition der Erweiterung gelten.

4. Zeitschranken für die Pfadausführung

In diesem Kapitel wird untersucht, wie lange die Ausführung eines Systems von Pfaden dauert. Dabei wird vorausgesetzt, dass ein Ausführungsplan, wie im letzten Kapitel beschrieben, bereits existiert.

Anmerkung In einem verteilten System laufen die Ressourcen und damit deren Uhren im Allgemeinen nicht synchron. Das macht es schwierig, einen „globalen" Zeitpunkt anzugeben, zu dem ein Auftrag oder ein Pfad ausgeführt wurde.

In [Lam78] wird jedoch gezeigt, dass es möglich ist, die Uhren von Ressourcen in einem verteilten System bis auf eine kleine Abweichung ε zu synchronisieren. Deshalb wird in diesem Kapitel davon ausgegangen, dass dieses ε so vernachlässigbar klein ist, dass die Uhren aller Ressourcen synchron laufen. Daher ist es nun möglich, globale Zeitpunkte anzugeben.

4.1. Zeitschranken durch Rekursionen

Unter der Voraussetzung, dass man die Ausführungszeit von jedem Auftrag auf jeder Ressource kennt, so ist es möglich die Ausführungszeit eines Pfades, eines Zyklus, einer Kette von Pfaden und schließlich des gesamten Systems $\mathcal{S}$ zu berechnen, siehe Abbildung 4.1.

Dafür gehe man davon aus, dass es eine Schedulingfunktion $r : \mathcal{S} \longrightarrow R$ gibt, welche die Zuordnung von Pfaden aus dem System $\mathcal{S}$ zu Ressourcen durchführt und die oben erwähnten Eigenschaften besitzt. Der genaue Aufbau von r wird ab Kapitel 5.2 beschrieben und diskutiert. Für die Betrachtungen hier reicht die Existenz der Funktion. Zusätzlich darf man annehmen, dass es eine Abbildung $k : \mathcal{S}_{\mathfrak{A}} \longrightarrow \mathcal{S}$ gibt, die jedem Auftrag seinen Pfad zuordnet.

Weiter gehe man davon aus, dass es eine Funktion $t_R : \mathcal{S}_{\mathfrak{A}} \times R \longrightarrow \mathbb{R}^+$ gibt, die zu einem ausführbereiten Auftrag aus der Menge der Aufträge $\mathcal{S}_{\mathfrak{A}}$ auf einer Ressource aus der Menge der Ressourcen R angibt, wie lange seine Ausführung darauf benötigt. Sie berechnet diese Zeit basierend auf dem aktuellen Zustand der Ressource wie zum Beispiel ihres Batteriezustandes. Des Weiteren muss die Zeit berücksichtigt werden, zu der der erste Auftrag eines Pfades ausgeführt werden kann, zum Beispiel wegen der Zuteilung einer Ressource. Für einen beliebigen Auftrag

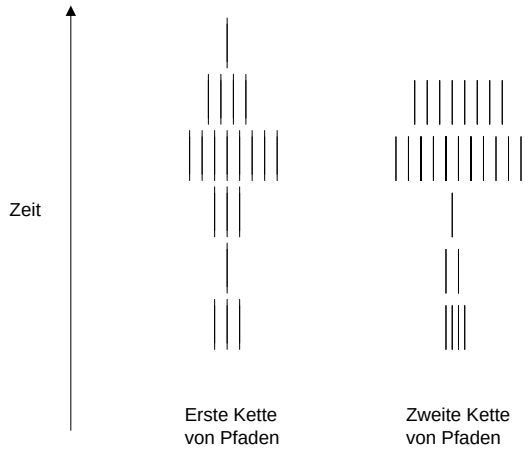


Abbildung 4.1.: Beispiel für zwei verschiedene Ketten von Pfaden

$x \in \mathcal{S}_{\mathfrak{A}}$ eines Pfades werde diese Zeit mit $t_{k(x)}$ gekennzeichnet und ist nach obiger Anmerkung eine globale Zeitangabe. Schließlich muss man noch davon ausgehen, dass jede Ressource nicht kontinuierlich prüfen kann, ob ein Auftrag ausführbereit ist. Es werde daher angenommen, dass alle Ressourcen eine gemeinsame maximale Abtastrate $t_A \in \mathbb{R}^+ \cup \{0\}$ besitzen, nach der die Ausführbarkeit eines Auftrages spätestens geprüft ist.

Definition 4.1.1. *Die Funktion $t_V : \mathcal{S}_{\mathfrak{A}} \longrightarrow \mathbb{R}^+$ gibt für einen Auftrag x des ausführbaren Systems $\mathcal{S}$ an, wann er ausführbar ist. t_V ist dabei wie folgt definiert:*

$$t_V(x) := \begin{cases} t_{k(x)} + t_A & \text{, falls Vorgängermenge } \mathcal{V}(x) = \emptyset \\ \max_{y \in \mathcal{V}(x)}\{t_V(y) + t_R(y,\, r(k(y))),\ t_{k(x)}\} + t_A & \text{, sonst} \end{cases}$$

Bemerkung 4.1.1. Zunächst ist klar, dass t_V wohldefiniert ist, da für einen Auftrag x in einem ausführbaren System von Pfaden insbesondere $x \notin \mathcal{V}(x)$ gilt. Weiterhin existiert das Maximum, da die Kardinalität der Vorgängermenge $\mathcal{V}(x)$ eines Auftrages x in einem ausführbaren System endlich ist, wegen der Irreflexivität von $<_{\mathfrak{A}}$ und Endlichkeit von $\mathcal{S}_{\mathfrak{A}}$.

Anschaulich gesprochen untersucht t_V, wann die Vorgängeraufträge von x spätestens ausgeführt sind. Dann wird noch mit der Startzeit $t_{k(x)}$ des Pfades von x verglichen, und spätestens zu diesem Zeitpunkt ist x ausführbereit.

Lemma 4.1.1. *In einem ausführbaren System $\mathcal{S}$ ist die Funktion t_V monoton. Das bedeutet: Für zwei Aufträge x und y mit $x <_{\mathfrak{A}} y$ gilt $t_V(x) \leq t_V(y)$.*

Beweis. Es gelte also $x <_{\mathfrak{A}} y$ für zwei Aufträge $x,\ y \in \mathcal{S}_{\mathfrak{A}}$. Dann gilt $x \in \mathcal{V}(y)$ und bei der Maximumsbildung für die Berechnung von $t_V(y)$ nimmt $t_V(x) + t_R(x,\ r(k(x)))$ teil. Also ist $t_V(x) \leq t_V(x) + t_R(x,\ r(k(x))) \leq t_V(y)$ wahr. □

Bemerkung 4.1.2. Man betrachte nun nochmals die Definition von t_V. Sei x_1 der erste Auftrag eines Pfades und sei x_2 ein weiterer Auftrag dieses Pfades. Es gilt $t_{k(x_1)} = t_{k(x_2)}$, da beide Aufträge auf demselben Pfad liegen. Insbesondere ist nun $t_V(x_1) \geq t_{k(x_1)}$ wahr. Weiterhin gilt offensichtlich $x_1 <_{\mathfrak{A}} x_2$ und wegen Lemma 4.1.1 gilt $t_V(x_1) \leq t_V(x_2)$. Wegen $x_1 \in \mathcal{V}(x_2)$ wird $t_V(x_1) + t_R(x_1,\ r(k(x_1)))$ bei der Berechnung von $t_V(x_2)$ in der Maximumsbildung berücksichtigt. Wegen eben Gesagtem gilt aber $t_V(x_1) + t_R(x_1,\ r(k(x_1))) \geq t_{k(x_1)} = t_{k(x_2)}$. Dies bedeutet nun, dass man für alle Aufträge x eines Pfades, die nicht an erster Stelle im Pfad stehen, die Zeit $t_{k(x)}$ bei der Berechnung von $t_V(x)$ nicht berücksichtigen muss. Für sie gilt deshalb $t_V(x) = \max_{y \in \mathcal{V}(x)}\{t_V(y) + t_R(y,\ r(k(y)))\} + t_A$.

Dieses Lemma bestätigt auch nochmal die Aussage aus Bemerkung 2.2.4, in der die Monotonie bereits beschrieben wurde.

Man kann sich die Berechnung von t_V vereinfachen, falls man die direkten Vorgängermengen der Aufträge des Systems $\mathcal{S}$ kennt. Dann reicht es nämlich für die Berechnung der Ausführungszeit eines Auftrages x nur die Aufträge aus seiner direkten Vorgängermenge zu betrachten, was im folgenden Satz beschrieben wird.

Lemma 4.1.2. *Sei $\mathcal{S}$ ein ausführbares System von Pfaden und sei $t'_V : \mathcal{S}_{\mathfrak{A}} \longrightarrow \mathbb{R}^+$ definiert wie folgt:*

$$t'_V(x) := \begin{cases} t_{k(x)} + t_A & \text{, falls } \max \mathcal{V}(x) = \emptyset \\ \max_{y \in \max \mathcal{V}(x)}\{t'_V(y) + t_R(y,\ r(k(y))),\ t_{k(x)}\} + t_A & \text{, sonst} \end{cases}$$

Dann gilt $t_V \equiv t'_V$ auf $\mathcal{S}_{\mathfrak{A}}$.

Beweis.
Wegen Satz 3.2.9 gilt für ein $x \in \mathcal{S}_{\mathfrak{A}}$: $\mathcal{V}(x) = \emptyset \Longleftrightarrow \max \mathcal{V}(x) = \emptyset$. Mit Definition 4.1.1 ist nun klar, dass für einen Auftrag $x \in \mathcal{S}_{\mathfrak{A}}$ mit $\mathcal{V}(x) = \max V(x) = \emptyset$ gilt: $t_V(x) = t'_V(x) = t_{k(x)} + t_A$.

Wegen Bemerkung 3.2.19 gilt für ein $x \in \mathcal{S}_{\mathfrak{A}}$: $\mathcal{V}(x) \neq \emptyset \Longleftrightarrow \max \mathcal{V}(x) \neq \emptyset$. Das bedeutet, dass man sich in diesem Fall für den Beweis der Gleichheit von t_V und t'_V auf die beiden unteren Zweige in deren Definition zurückziehen kann. In diesem Fall gilt für einen Auftrag $x \in \mathcal{S}_{\mathfrak{A}}$:

$$t_V(x) = \max_{y \in \max \mathcal{V}(x)}\{t_V(y) + t_R(y,\ r(k(y))),\ t_{k(x)}\} + t_A \tag{4.1}$$

Es ist offensichtlich, dass $t_V(x) \geq \max_{y \in \max \mathcal{V}(x)}\{t_V(y) + t_R(y,\ r(k(y))),\ t_{k(x)}\} + t_A$ gilt, denn alle Aufträge auf der rechten Seite der Gleichung nehmen an der Maximumsbildung von $t_V(x)$ teil. Annahme, es würde gelten $t_V(x) > \max_{y \in \max \mathcal{V}(x)}\{t_V(y)+$

$t_R(y,\, r(k(y)))\,,\; t_{k(x)}\} + t_A$. Dann gäbe es einen Auftrag $x_1 \in \mathcal{V}(x)$ mit $x_1 \notin \max \mathcal{V}(x)$ und es würde gelten:

$$t_V(x_1) + t_R(x_1,\, r(k(x_1))) + t_A > \max_{y \in \max \mathcal{V}(x)} \{t_V(y) + t_R(y,\, r(k(y)))\,,\; t_{k(x)}\} + t_A \quad (4.2)$$

Wegen Bemerkung 3.2.19 muss es aber einen Auftrag $x_F \in \max \mathcal{V}(x)$ geben mit $x_1 <_{\mathfrak{A}} x_F <_{\mathfrak{A}} x$ und wegen dem Beweis von Lemma 4.1.1 gilt die Ungleichung $t_V(x_1) + t_R(x_1,\, r(k(x_1))) \leq t_V(x_F)$. Sie ist äquivalent zu Ungleichung $t_V(x_1) + t_R(x_1,\, r(k(x_1))) + t_A \leq t_V(x_F) + t_A$, welche im Widerspruch zu Ungleichung (4.2) steht. Also gilt die Gleichheit in (4.1) und es bleibt zu zeigen, dass für einen Auftrag $x \in \mathcal{S}_{\mathfrak{A}}$ gilt:

$$t'_V(x) = \begin{cases} t_{k(x)} + t_A & \text{, falls } \max \mathcal{V}(x) = \emptyset \\ \max_{y \in \max \mathcal{V}(x)} \{t_V(y) + t_R(y,\, r(k(y)))\,,\; t_{k(x)}\} + t_A & \text{, sonst} \end{cases} \quad (4.3)$$

Dies geschieht durch strukturelle Induktion über die Aufträge des Systems $\mathcal{S}$. Durch $<$ sind die Pfade schon vorgeordnet wie in Abbildung 4.1 schematisch gezeigt wird. Dabei gibt es, wie bereits bekannt ist, Ketten von Pfaden, die unabhängig untereinander sind bezüglich $<_{\mathfrak{A}}$ und $<$. Dies bedeutet, dass man sie jeweils separat betrachten kann. Eventuelle gegenseitige Beeinflussungen durch einen gemeinsame Nutzung von Ressourcen werden hier durch die Startzeit eines Pfades berücksichtigt.

Induktionsanfang: Wenn in der Pfadkette zuerst ein freier Pfad kommt, dann ist klar, dass für seinen ersten Auftrag x gilt $\mathcal{V}(x) = \max \mathcal{V}(x) = \emptyset$ (da $\mathcal{S}$ ausführbar ist und Satz 3.2.9), und es gilt offensichtlich $t_V(x) = t'_V(x) = t_{k(x)} + t_A$, siehe oben. Wenn in der Pfadkette zunächst ein Zyklus von Pfaden bezüglich $<$ kommt, dann muss es im Zyklus zumindest einen Pfad geben für dessen ersten Auftrag wieder diesselben Bedingungen wie im Fall des freien Pfades gelten und daher folgt für diesen Auftrag ebenso $t_V(x) = t'_V(x) = t_{k(x)} + t_A$.

Induktionsvoraussetzung: Die Behauptung gelte bis zu einem beliebigen, aber festen Ausführungsschritt. Das bedeutet, dass Gleichung (4.3) auf allen Aufträgen einer Kette von Pfaden bis zu einer beliebigen, aber festen Menge von Aufträgen gilt, siehe auch Abbildung 4.2.

Induktionsschritt: Da das System $\mathcal{S}$ ausführbar ist, muss mindestens ein Auftrag x der Kette ausführbar sein. Wenn $\max \mathcal{V}(x) = \mathcal{V}(x) = \emptyset$ gilt, dann gilt offensichtlich $t'_V(x) = t_{k(x)} + t_A$ nach Definition von t'_V. Dies verifiziert den oberen Zweig in Gleichung (4.3).

Sonst sind alle Aufträge aus $\mathcal{V}(x)$ (äquivalent: $\max \mathcal{V}(x)$) bereits ausgeführt und für diese Aufträge gilt die Gleichheit von t_V und t'_V nach Induktionsvoraussetzung und Gleichung (4.1). Dann gilt aber auch sicherlich $t'_V(x) = \max_{y \in \max \mathcal{V}(x)} \{t'_V(y) +$

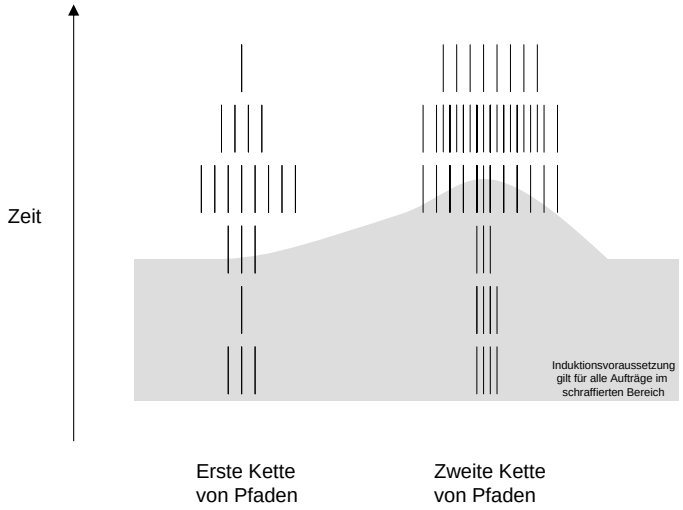


Abbildung 4.2.: Veranschaulichung der Induktionsvoraussetzung

$t_R(y,\, r(k(y))),\ t_{k(x)}\} + t_A = \max_{y \in \max \mathcal{V}(y)}\{t_V(y) + t_R(y,\, r(k(y))),\ t_{k(x)}\} + t_A$, was den unteren Zweig von Gleichung (4.3) verifiziert. □

Bemerkung 4.1.3. Wegen der Gleichheit von t_V und t'_V auf $\mathcal{S}$ und der Monotonie von t_V ist auch t'_V monoton, das heißt, wenn für zwei Aufträge x und y aus dem ausführbaren System $\mathcal{S}$ gilt $x <_{\mathfrak{A}} y$, dann gilt auch $t'_V(x) \leq t'_V(y)$.

Nun ist es leicht, die Zeit bis zum Abschluss der Ausführung eines Pfades, eines Zyklus, einer Kette von Pfaden und des ganzen Systems $\mathcal{S}$ zu berechnen.

Satz 4.1.1. *Sei $\mathcal{S}$ ein ausführbares System von Pfaden. Unter der Voraussetzung einer störungsfreien Ausführung, das heißt, dass ausreichend Ressourcen zur Verfügung stehen und diese auch keine Ausfälle haben, gelten die folgenden Ausführungszeiten.*

1. *Sei $X \in \mathcal{S}$ ein Pfad und x_E sein letzter Auftrag. Dann ist die Ausführung von X genau nach der Zeit*

$$\textit{Ausführungszeit Pfad} = t_V(x_E) + t_R(x_E,\, r(k(x_E)))$$

   beendet.

2. *Sei ζ ein Zyklus von Pfaden in $\mathcal{S}$. Seien X_1, ..., X_k die k letzten Pfade des Zyklus entsprechend seines Ausführungsplanes. Weiter seien x_E^1, ..., x_E^k die letzten Aufträge dieser Pfade. Dann ist die Ausführung des Zyklus genau nach der Zeit*

$$\textit{Ausführungszeit Zyklus} = \max_{i \in \{1, \ldots k\}} \{t_V(x_E^i) + t_R(x_E^i,\, r(k(x_E^i)))\}$$

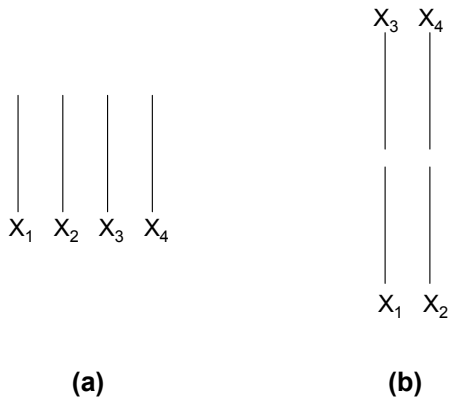


Abbildung 4.3.: Beispiele für „letzte Pfade“

beendet.

3. *Gegeben sei eine Kette von Pfaden, wie beispielsweise in Abbildung 4.1 dargestellt. Wenn als letztes in der Kette ein freier Pfad ausgeführt werden soll, so ist die Kette genau nach der Zeit wie unter Punkt* 1 *beschrieben vollständig bearbeitet. Wenn als letztes in der Kette ein Zyklus ausgeführt werden soll, so ist die Kette genau nach der Zeit wie unter Punkt* 2 *beschrieben vollständig bearbeitet.*

4. *Bestehe das System $\mathcal{S}$ aus j Ketten K_1, ... K_j von Pfaden. Dann ist die Ausführung von $\mathcal{S}$ genau nach der Zeit*

$$\textit{Ausführungszeit System } \mathcal{S} = \max_{i \in \{1, \dots j\}} \{\textit{Ausführungszeit Kette } K_i\}$$

beendet.

Beweis. Klar. □

Bemerkung 4.1.4. Die letzten Pfade eines Zyklus sind entweder alle seine Pfade, falls keine Ressourcen-Optimierung stattgefunden hat, oder die letzten Pfade gemäß den Anordnungen, siehe dazu Abbildung 4.3. Im Fall (a) hat keine Anordnung stattgefunden, weshalb die Pfade X_1, X_2, X_3, X_4 die letzten Pfade des Zyklus sind. Im Falle (b) sind die Pfade X_3 und X_4 wegen der durch die Reihenfolge beschriebenen Anordnung die letzten Pfade des Zyklus.

Beispiel 4.1.1. *Abbildung 4.4 zeigt drei Pfade eines (nicht ressourcen-optimierten) Zyklus mit ihren jeweiligen Startzeiten. In diesem Beispiel wird davon ausgegangen, dass für die maximale Abtastrate idealerweise $t_A = 0$ gilt. Die Startzeit von Pfad X_1 beträgt* 20 *Zeiteinheiten (T), die von Pfad X_2 beträgt $0\,T$, und die von Pfad*

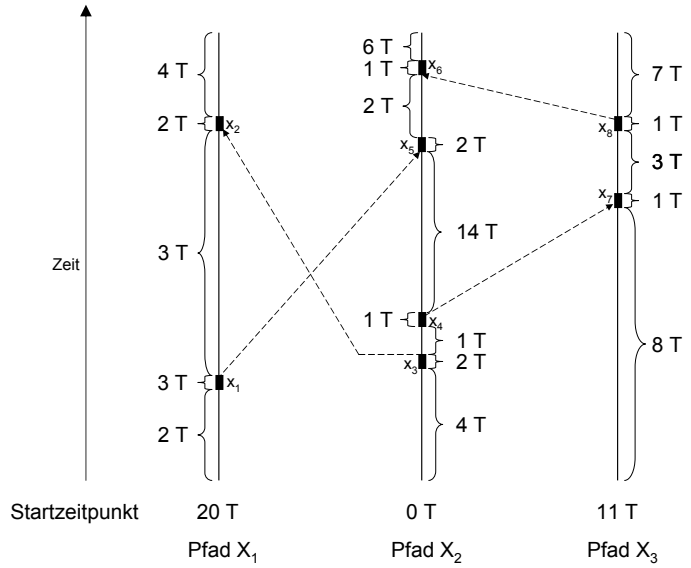


Abbildung 4.4.: Beispiel für die Ausführungszeiten eines Zyklus

X_3 beträgt $11\,T$. Nun soll die Ausführungszeit der einzelnen Pfade und des gesamten Zyklus berechnet werden. Hier wird das Zeitmaß t'_V berechnet und entsprechend werden auch nur die Abhängigkeiten bezüglich der direkten Vorgängermengen in der Abbildung angegeben. Die schwarzen Blöcke in der Abbildung kennzeichnen die abhängigen Aufträge. Durch die Klammern werden die Zeitdauern zur Ausführung der Aufträge beziehungsweise von Blöcken von Aufrägen angegeben. Dabei können die Ausführungszeiten von Aufträgen verschieden sein, abhängig vom Auftrag und der ausführenden Ressource. Weiterhin sagt die Länge eines Blocks auch nichts über die Zeitdauer seiner Ausführung aus. Dies wird in Pfad X_3 dargestellt, in dem der Block von Anfang bis zum Auftrag x_7 genau $8\,T$ zur Ausführung und der relativ kleine Block nach Auftrag x_8 nur $7\,T$ benötigt. Es ist klar, dass $t'_V(x_1) = 22\,T$ und $t'_V(x_3) = 4\,T$ gilt. Nach Definition kann man nun $t'_V(x_2) = \max\{28\,E,\ 6\,E\} = 28\,T$ berechnen. Somit beträgt die Ausführungszeit für Pfad X_1 definitionsgemäß $34\,T$. Weiter gilt offensichtlich $t'_V(x_4) = 7\,T$ und $t'_V(x_7) = \max\{8\,E,\ 19\,E\} = 19\,T$. Damit kann man $t'_V(x_8) = 23\,T$ berechnen, und die Gesamtausführungszeit für Pfad X_3 beträgt dann $31\,T$. Weiter berechnet man $t'_V(x_5) = \max\{22\,E,\ 25\,E\} = 25\,T$ und $t'_V(x_6) = \max\{29\,E,\ 24\,E\} = 29\,T$. Somit ergibt sich die Ausführungszeit für Pfad X_2 zu $36\,T$. Die Gesamtausführungszeit des Zyklus ergibt sich somit zu $\max\{34\,E,\ 36\,E,\ 31\,E\} = 36\,T$.

An diesem Beispiel sieht man, dass es nicht unbedingt darauf ankommt, wann die Ausführung eines Pfades durch eine Ressource gestartet wird, da seine Ausführung durch Abhängigkeiten mit anderen Pfaden blockiert werden kann.

Als Vergleich kann man die Ausführungszeit der optimierten Variante des Zyklus

berechnen, in der Pfad X_3 und Pfad X_1 in dieser Reihenfolge auf derselben Ressource ausgeführt werden. Dabei wird angenommen, dass die Ausführungszeiten der Aufträge unabhängig von der ausführenden Ressource sind, der Startzeitpunkt für Ressource X_3 weiterhin bei $11\,T$ liegt und Pfad X_1 unmittelbar nach Pfad X_3 ausgeführt werden kann. Damit ergeben sich für Pfad X_3 die Ausführungszeit $31\,T$, für Pfad X_2 die Ausführungszeit $47\ T$ und für Pfad X_1 die Ausführungszeit $45\,T$. Die längere Ausführungszeit durch die Ressourcen-Optimierung ist hier also gar nicht so groß, was an dem frühen Startzeitpunkt von Pfad X_2 und seiner langen Ausführungszeit liegt.

5. Eine organische Pfadverteilung

In diesem Kapitel werden zwei Ansätze vorgestellt, die eine Verteilung der Pfade eines ausführbaren Systems in einem verteilten System von Ressourcen leisten. Dabei garantieren sie organische Eigenschaften wie Selbstkonfiguration und Selbstheilung unter Echtzeitbedingungen. Die Ansätze bauen auf den in [BPR08, BPR07, BRP08, RBW08, BR09] beschriebenen Arbeiten auf. Daher wird ihr Kern, das künstliche Hormonsystem (KHS), im Folgenden kurz eingeführt.

5.1. Das künstliche Hormonsystem (KHS)

Die Idee des künstlichen Hormonsystems ist, eine Verteilung von Tasks zu Ressourcen in einem verteilten System mit ω Ressourcen zu ermöglichen ($\omega \in \mathbb{N}$), ohne dafür eine zentrale Kontrollinstanz zu benötigen. Der Vorteil dieser Idee liegt darin, dass kein „Single-Point-of-Failure“ durch einen Ausfall der Kontrollinstanz vorkommen kann. Außerdem werden durch die Natur des KHS organische Eigenschaften garantiert, auf die später nochmal eingegangen wird.

Für die Taskzuordnung werden drei Grundtypen von Hormonen benutzt:

Eignungswert: Sein Wert gibt an, wie gut eine Ressource eine bestimmte Task bearbeiten kann.

Suppressor: Ein Suppressor hemmt die Ausführung einer Task auf einer Ressource. Ein Wert eines Suppressors wird von dem Wert eines Eignungswertes subtrahiert, da er sich umgekehrt zu dessen Wirkungsweise verhalten soll.

Accelerator: Ein Accelerator begünstigt die Ausführung einer Task auf einer Ressource. Ein Wert eines Accelerators wird zum Wert eines Eignungswertes addiert und dient dazu Tasks, die miteinander kommunizieren sollen, auf Ressourcen zu platzieren, die beispielweise nahe beieinander liegen. Acceleratoren werden daher nur lokal versendet, das heißt sie werden nur in einer gewissen Umgebung der Ressource verteilt. Dabei heißen zwei Tasks *verwandt*, wenn sie Acceleratoren für jeweils die andere Task senden.

Es werde vereinbart, dass im Folgenden mit dem Größenvergleich und arithmetischen Operationen auf Hormonen die jeweiligen Operationen auf ihren Werten gemeint ist. Ebenso darf man von einer Task T_i anstatt von einer Task des Typs T_i sprechen. Die beiden Vereinbarungen dienen zur Vereinfachung der Notation.

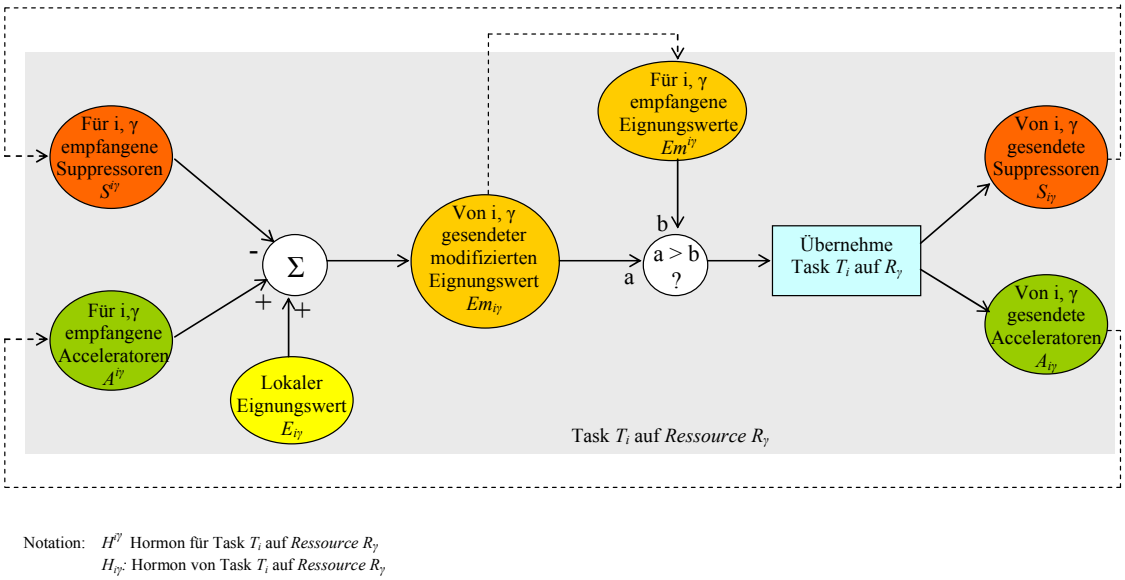


Abbildung 5.1.: Hormon-Regelkreis

Abbildung 5.1 skizziert den grundlegenden Regelkreis, mit dem eine Task T_i einer Ressource R_γ zugeordnet wird. Dieser Regelkreis wird für jede Task auf jeder Ressource durchgeführt und bestimmt aufgrund der Pegel der drei Hormontypen, ob die Task T_i auf R_γ ausgeführt wird oder nicht.

Der lokale, statische Eignungswert $E_{i\gamma}$ gibt an, wie gut Task T_i auf R_γ ausgeführt werden kann. Hiervon werden alle für Task T_i auf R_γ empfangenen Suppressoren $S_{i\gamma}$ subtrahiert und alle für Task T_i auf R_γ empfangenen Acceleratoren $A_{i\gamma}$ addiert. Hieraus errechnet sich der modifizierte Eignungswert $Em_{i\gamma}$ für Task T_i auf R_γ. Dieser modifizierte Eignungswert wird an alle anderen Ressourcen gesendet und mit den von den anderen Ressourcen empfangenen Eignungswerten $Em^{i\gamma}$ verglichen. Ist $Em_{i\gamma}$ größer als alle empfangenen Eignungswerte $Em^{i\gamma}$, so wird die Task T_i von R_γ übernommen (bei Gleichheit entscheidet ein zweites Kriterium, zum Beispiel die Nummer der Ressource). Daraufhin sendet Task T_i auf R_γ ihrerseits Suppressoren $S_{i\gamma}$ und Acceleratoren $A_{i\gamma}$ aus. Dieser Vorgang wiederholt sich periodisch und wird in Abbildung 5.2 dargestellt. Im Folgenden wird er als *Hormonzyklus* bezeichnet.

Er startet naturgemäß mit „Sende Hormone (S)“, wobei eine Ressource alle Hormontypen als Entscheidungsgrundlage sendet (zumindest die Eignungswerte müssen hierfür bekannt sein). Sind die Hormone gesendet, so wird nach Ablauf der Zeit t_S basierend auf den empfangenen Hormonwerten eine „Entscheidung (E)“ über die *Vergabe von genau einer Task auf einer Ressource* getroffen. Nach Ablauf der Zeit t_E startet der Hormonzyklus von neuem mit dem Senden der Hormone.

Besonders wichtig ist hierbei, dass auf jeder Ressource in jedem Hormonzyklus die Hormone für alle Tasks, die die Ressource übernehmen kann, gesendet werden,

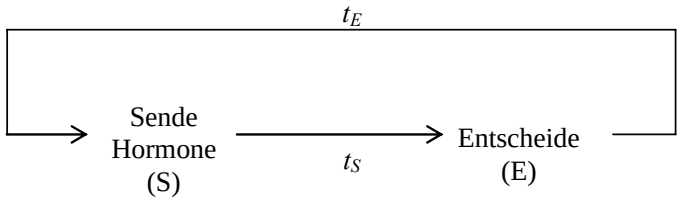


Abbildung 5.2.: Periodischer Ablauf von Senden der Hormone und Entscheidung zur Taskübernahme

aber pro Entscheidung in diesem Hormonzyklus über die Übernahme genau einer Task entschieden wird. Im nächsten Hormonzyklus wird dann über die nächste Task entschieden und so weiter. Diese Vorgehensweise ist daher wichtig, da so die gesendeten Suppressoren und Acceleratoren zur Wirkung kommen können. Wenn nämlich in einem einzigen Hormonzyklus über alle Tasks entschieden werden würde, würden diese beiden Hormontypen sich nicht auswirken können. In [BPR08] auch werden die Dauern der beiden Zeiten t_E und t_S besprochen. Dabei wird angenommen, dass man

$$t_E \approx 0$$

annehmen darf, da die Rechnungen für eine Entscheidung sehr einfach sind. Für die Zeit t_S zwischen dem Senden der Hormone und dem Entscheiden muss gelten, dass sie die Zeit t_E zwischen dem Entscheiden und dem Senden und die Kommunikationszeit $t_{Kom} \in \mathbb{R}^+$, die es dauert ein Hormon im ganzen System zu verteilen (broadcasten), mit einschließt. Nach [BPR08] muss gelten:

$$t_S \geq t_E + 2t_{Kom}$$

In dieser Ungleichung wird t_{Kom} doppelt gewichtet, um eine konsistente Auslieferung aller Hormone im verteilten System zu gewährleisten, siehe [BPR08]. Da man die Annahme $t_E \approx 0$ getroffen hat, vereinfacht sich die letzte Ungleichung zu:

$$t_S \geq 2t_{Kom} \tag{5.1}$$

In einer Implementierung des KHS muss man daher sicherstellen, dass die „*Zykluszeit*“ t_S gemäß Ungleichung (5.1) dimensioniert wird.

Zusätzlich wird in der beschriebenen Arbeit ein weiterer Suppressortyp eingeführt, der sogenannte *Lastsuppressor*. Dieser Suppressor wird nur lokal an die Ressource gesendet, welche eine Task T_i übernommen hat. Er wirkt sich jedoch nicht nur auf Task T_i, sondern auf alle Tasks auf dieser Ressource aus. Der Lastsuppressor bestimmt somit, wieviele Tasks eine Ressource übernehmen kann. Ein sehr starker Lastsuppressor sorgt dafür, dass eine Ressource beispielweise nur eine Task übernehmen kann, ein schwächerer Suppressor erlaubt die Übernahme mehrerer Tasks.

An der hier beschriebenen Wirkungsweise des KHS erkennt man Analogien zu biologischen Hormonsystemen, da hier Hormone chemische Botenstoffe sind, die entweder in den Nachbarbereich von Zellen oder durch die Blutbahnen in den ganzen Körper eines Lebewesens gelangt. Die Hormone des KHS sind kurze Nachrichten, die entweder zu benachbarten Ressourcen per Multicast oder im ganzen System per Broadcast verteilt werden. Details zu den Analogien finden sich in [BPR08].

Weiter besitzt das KHS die folgenden organischen Eigenschaften:

- Es ist *selbstorganisierend*, da kein äußerer Einfluss die Taskvergabe zur Laufzeit steuert.
- Es ist *selbstkonfigurierend*, durch Austausch der Hormone wird eine initiale Taskvergabe bewirkt. Die Selbstkonfiguration ist beendet, sobald alle Eignungswerte zu 0 geworden sind, das heißt keine weitere Task mehr übernommen werden möchte. Dies geschieht durch Aussenden der Suppressoren.
- Es ist selbstheilend, bei Ausfall einer Task oder einer Ressource fallen alle davon gesendeten Hormone, insbesondere die Suppressoren weg. Dies bewirkt die automatische Neu-Übernahme dieser Tasks von derselben Ressource (wenn diese noch aktiv ist) oder von anderen Ressourcen. Die einzige zusätzliche Anforderung hier ist, dass ein von einer Task T_i auf einer Ressource für eine Task T_j auf einer Ressource gesendetes Hormon mit einer Verfallszeit belegt ist. Erhält die empfangende Task T_j auf einer anderen Ressource innerhalb dieser Verfallszeit keinen neuen Hormonwert, so wird der alte Hormonwert verworfen. Hierdurch kann der Wegfall eines Hormons nach Ablauf der Verfallszeit erkannt werden.

5.1.1. Zeitschranken

Man kann Garantien bezüglich der maximalen Zeitdauern (Worst Case Execution Time, kurz WCET) der organischen Eigenschaften des KHS geben. Diese werden in [BPR08] hergeleitet und somit wird gezeigt, dass das KHS echtzeitfähig ist.

Die Zeitschranken für die Selbstkonfiguration und Selbstheilung werden hier zitiert, da sie im Folgenden benötigt werden. Weiter wird angenommen, dass sich m Tasks im System ($m \in \mathbb{N}$) befinden und die Dauer eines Zyklus gemäß Ungleichung (5.1) gewählt wurde.

Für die WCET $t_{K_{\max}}$ der Selbstkonfiguration gilt unter der Annahme, dass die m Tasks im System verteilt werden sollen:

$$t_{K_{\max}} = 2m \text{ Hormonzyklen} \tag{5.2}$$

Dies bedeutet, dass das KHS höchstens $2m$ Hormonzyklus benötigt um alle Tasks im System zu verteilen. Die Konfigurationszeit ist also linear zur Anzahl der Tasks

im System.

Beim Ausfall einer Ressource oder einer Task steht deren Funktionalität im Allgemeinen nicht zur Verfügung, bis eine Neuzuordnung erfolgt ist. Da der Zeitpunkt des Ausfalls nicht vorhersehbar ist, kann durch das KHS hier nur ein eingeschränktes Echtzeitverhalten garantiert werden. Man muss einige Fälle unterscheiden:

1. Im schlimmsten Fall fallen alle Ressourcen gleichzeitig aus. Dann ist natürlich keine Selbstheilung möglich.

2. Sonst gilt die Annahme, dass auch nach dem Ausfall einer Ressource oder Task noch genug Ressourcen übrig bleiben, so dass alle Tasks übernommen werden können. Dann gilt im Wesentlichen diesselbe Zeit wie bei der Selbstkonfiguration. Es kommt lediglich die Zeit hinzu, die die anderen Ressource benötigen um den Ausfall zu erkennen. Dies wird durch die bereits erwähnte Verfallszeit von Hormonen realisiert und es wird angenommen, dass diese a Hormonzyklen mit $a \in \mathbb{N}$ betrage.

   Für die WCET $t_{H_{\max}}$ der Selbstheilung gilt dann:

$$t_{H_{\max}} = 2m + a \text{ Hormonzyklen} \tag{5.3}$$

Es ist interessant, dass die WCETs (auch die für die Selbstoptimierung, siehe [BPR08]) zunächst unabhängig von der Anzahl ω der sich im System befindlichen Ressourcen sind. Dies ist aber nur scheinbar richtig, denn die Kommunikationszeit t_{Kom} vergrößert sich mit wachsender Ressourcenzahl im System. Auf diese Weise geht die Anzahl der Ressourcen dann doch in die oben genannten WCETs ein.

5.1.2. Übernahmehäufigkeit einer Task

Es ist wichtig vorherzusagen, wieviele Tasks eines Typs von Ressourcen übernommen werden können. Wenn man alle Hormontypen aus [BPR08] einbezieht, dann ergeben sich verschiedenste Abschätzungen.

Für diese Arbeit reicht es aber, die Eignungswerte und Suppressoren (ohne Lastsuppressoren) wie oben beschrieben zu analysieren, da diese in den späteren Betrachtungen nicht zum Tragen kommen. Dazu wird im Folgenden angenommen, dass der Eignungswert $E_{i\gamma}$ für eine Task vom Typ T_i auf jeder Ressource γ beschränkt sind, das heißt es existieren $E^i_{\min}$, $E^i_{\max} \in \mathbb{R}^+$ mit:

$$0 < E^i_{\min} \leq E_{i\gamma} \leq E^i_{\max}$$

Die Forderung, dass sogar der kleinste Eignungswert echt größer als Null ist, soll deutlich machen, dass hier davon ausgegangen wird, dass jede Ressource jede Task (wenn auch eventuell mit einem sehr geringen Eignungswert) übernehmen kann.

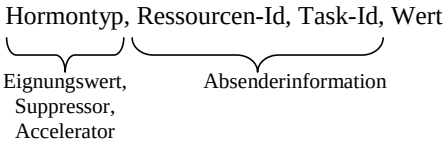


Abbildung 5.3.: Aufbau eines künstlichen Hormons

Diesselbe Annahme gilt auch für den Suppressor $S_{i\gamma}$ einer Task vom Typ T_i, die an eine beliebige Ressource γ gesendet wird. Das bedeutet es existieren $S^i_{\min}$, $S^i_{\max} \in \mathbb{R}^+$ und es gilt:

$$0 < S^i_{\min} \leq S_{i\gamma} \leq S^i_{\max}$$

Hier zeigt die Forderung $0 < S_{\min}$ an, dass durch die Übernahme einer Task ein Suppressor ausgesendet wird, der auch tatsächlich die weitere Übernahme einer Task dieses Typs hemmt, da er echt größer als Null ist.

Nun kann man sich klar machen, wieviele Tasks vom Typ T_i *mindestens* übernommen werden: Jedes Mal, wenn wenn eine Task vom Typ T_i im System übernommen wird, sendet die übernehmende Ressource einen Suppressor aus, im schlimmsten Fall einen Suppressor der Stärke $S^i_{\max}$. Jeder Suppressor wird vom Eignungswert der Task vom Typ T_i auf allen Ressource im System subtrahiert. Eine Ressource ist in der Lage eine Task vom Typ T_i zu übernehmen, solange der modifizierte Eignungswert dieses Tasktyps größer oder gleich Null ist, siehe Abbildung 5.1. Daraus folgt unmittelbar, dass eine untere Schranke für die Anzahl der Übernahmen U_i einer Task vom Typ T_i durch $\frac{E^i_{\min}}{S^i_{\max}}$ gegeben ist. Ganz ähnlich macht man sich klar, dass eine entsprechende obere Schranke durch $\frac{E^i_{\max}}{S^i_{\min}}$ gegeben ist. Insgesamt gilt also für die Anzahl der Übernahmen U_i einer Task vom Typ T_i:

$$\left\lceil \frac{E^i_{\min}}{S^i_{\max}} \right\rceil \leq U_i \leq \left\lceil \frac{E^i_{\max}}{S^i_{\min}} \right\rceil \tag{5.4}$$

5.1.3. Hormondatenaufkommen im KHS

Im Folgenden werden Abschätzungen zum Hormondatenaufkommen, welches an einer beliebigen der ω Ressource pro Hormonzyklus anfällt, gemäß [BPR08] angegeben. Dabei ist klar, dass Eignungswerte, Suppressoren und Acceleratoren neben dem eigentlichen Wert noch eine Absenderinformation benötigen, um alte Hormonwerte durch neue, aktuellere eines Absenders ersetzen zu können. Ein gesendetes Hormon besitzt daher die in Abbildung 5.3 dargestellte Struktur, durch die man konkrete Annahmen über die Datenmenge eines Hormons aufstellen kann.

Die benötigten Hormone, die sich die Ressourcen pro Hormonzyklus untereinander zusenden, werden in Tabelle 5.1 aufgelistet. Alle anderen Hormone werden

Broadcast an alle anderen Ressourcen:	Ein modifizierter Eignungswert pro beworbener Task Ein Supressor pro übernommener Task
Multicast an Nachbarn:	Ein Accelerator pro verwandter Task einer übernommenen Task

Tabelle 5.1.: Hormonaufkommen

nur lokal versandt beziehungsweise genutzt und werden daher nicht in diese Bilanz aufgenommen.

Damit kann man die folgende Abschätzung für die Broadcast-Datenmenge D_γ^B, die durch eine Ressource γ pro Hormonzyklus verursacht wird, aufstellen:

$$D_\gamma^B = D^E k_\gamma + D^S e_\gamma$$

Hierbei ist D^E die Datenmenge, die zur Übertragung eines Eignungswertes benötigt wird und k_γ die Anzahl der Tasks, um die sich Ressource γ beworbenen hat und welche noch nicht vollständig im gesamten System übernommen wurden. Weiter ist D^S die Datenmenge, die zur Übertragung eines Suppressors benötigt wird, und e_γ sei die Anzahl aller auf γ ausgeführter Tasks.

Für die Multicast-Datenmenge D_γ^M, die durch eine Ressource γ pro Hormonzyklus verursacht wird, gilt folgender Zusammenhang:

$$D_\gamma^M = D^A \sum_{T_i \text{ auf } \gamma} v_i$$

In dieser Gleichung ist D^A die Datenmenge, die zur Übertragung eines Accelerators benötigt wird. Die Konstante v_i zeigt an, wie viele Tasks zu einer Task T_i verwandt sind. Wie bereits oben erläutert senden verwandte Tasks Acceleratoren in der Umgebung ihrer ausführenden Ressource aus, um dort ihre gemeinsame Ausführung zu begünstigen. Damit wird mit der letzten Gleichung die Datenmenge für Multicasts gegeben durch die Tasks, die auf einer Ressource laufen und Acceleratoren an ihre verwandten Tasks auf anderen Ressourcen senden.

Aus diesen beiden Beziehungen kann man dann das Gesamtdatenaufkommen, welches an einer Ressource pro Hormonzyklus anfällt, berechnen. Dabei betrachtet man zwei Zustände: Der erste Zustand zeichnet sich dadurch aus, dass das verteilte System am Anfang steht und noch keine Tasks vergeben hat. Der zweite Zustand ist dadurch gekennzeichnet, dass die Taskvergabe abgeschlossen ist und sich das System im Gleichgewicht befindet.

Im ersten Zustand (*Start*) wurde noch keine Task vergeben. Das bedeutet, dass auch keine Suppressoren oder Acceleratoren versendet werden. Wenn mit $k_{\max}$ die maximale Zahl von Tasks gekennzeichnet ist, um die sich eine beliebige Ressource

γ aus dem verteilten System bewerben kann, formal $k_{\max} = \max_{\gamma \in \text{ System}} k_\gamma$, dann ist das maximale Datenaufkommen $D_{\max}^{\text{Start}}$ pro Hormonzyklus an einer beliebigen Ressource gegeben durch:

$$D_{\max}^{\text{Start}} = \omega k_{\max} D^E \tag{5.5}$$

Im zweiten Zustand (*Ende*) wurden alle Tasks vergeben. Das bedeutet, dass keine Eignungswerte mehr gesendet werden und nur noch Suppressoren und Acceleratoren durch die vergebenen Tasks geschickt werden. Das maximale Datenaufkommen $D_{\max}^{\text{Ende}}$ pro Hormonzyklus an einer beliebigen Ressource ist dann gegeben durch:

$$D_{\max}^{\text{Ende}} = \omega e_{\max} D^S + \varphi_{\max} v_{\max} e_{\max} D^A \tag{5.6}$$

Hierbei bezeichnet $e_{\max}$ die maximale Anzahl von Tasks, die eine beliebige Ressource im verteilten System übernehmen kann. Die Konstante $\varphi_{\max}$ gibt an, wieviele Ressourcen höchstens in der Umgebung einer beliebigen Ressource liegen können, und schließlich gibt die Konstante $v_{\max}$ an, wieviele verwandte Tasks eine beliebige Task überhaupt haben kann.

5.2. Einsatz des KHS zur organischen Pfadverteilung

Im Folgenden werden zwei Ansätze vorgestellt, in denen das KHS angepasst wird, um Pfade auf Ressourcen zu verteilen. Das bedeutet, dass anstelle von allgemeinen Tasks nun die Pfade eines ausführbaren Systems $\mathcal{S}$ auf einer Anzahl von Ressourcen verteilt werden sollen, und dies durch das KHS übernommen wird. Im Folgenden wird immer vorausgesetzt, dass $\mathcal{S}$ ausführbar ist, da sonst eine Ausführung seiner Pfade a priori unmöglich ist. Die Vorteile dieser Vorgehensweise sind einerseits, dass die bereits erläuterten organischen Eigenschaften des KHS genutzt und erhalten werden, um eine dezentrale stabile Verteilung der Pfade zu gewährleisten, und andererseits, dass die Echtzeiteigenschaften des KHS erhalten bleiben.

Die Ausführung der Pfade auf einer Ressource wird dann durch eine Ausführungsinstanz, die sich auf jeder Ressource befindet, überwacht.

In dieser Arbeit wird vorausgesetzt, dass immer genügend Ressourcen vorhanden sind, um die Pfade auszuführen, und dass es keine Störungen in der Kommunikation gibt. Das bedeutet, dass Hormone, die durch Broadcasts und Multicasts gesendet werden, die jeweils adressierten Ressourcen sicher erreichen. Weiterhin wird davon ausgegangen, dass alle Ressourcen gleichzeitig oder zumindest mit geringen Zeitversatz gestartet werden, so dass sie in einem konsistenten Zustand sind. Szenarien, die nicht diese Voraussetzungen erfüllen, würden den Rahmen dieser Arbeit überschreiten und werden daher im Kapitel 8 als Ausblick beschrieben.

Im ersten Ansatz, dem einstufigen KHS, das in Unterkapitel 5.2.1 vorgestellt wird, werden die Pfade von $\mathcal{S}$ direkt auf die Ressourcen verteilt. Im zweiten Ansatz, dem

zweistufigen KHS, das in Unterkapitel 5.2.2 vorgestellt wird, werden zunächst exklusiv Ressourcen für die Pfade aus Teilsystemen, die gemäß Algorithmus 2 erzeugt wurden, reserviert, und dann Pfade aus diesen Teilsystemen auf die exklusiven Ressourcen verteilt. Der Algorithmus der Ausführungsinstanz, also wie die Pfade lokal auf einer Ressource ausgeführt werden, wird in Unterkapitel 5.2.3 präsentiert. Ein Vergleich des einstufigen und des zweistufigen KHS wird in Unterkapitel 5.2.4 vorgenommen. Wenn sich jedoch Änderungen im Zeitverhalten oder Datenaufkommen des ein- oder zweistufigen KHS gegenüber den Aussagen des originalen KHS ergeben, so werden diese der Übersichtlichkeit halber sofort besprochen.

In allen Fällen müssen auf den Ressourcen die Eignungswerte, Acceleratoren und Suppressoren für die verschiedenen Pfade von $\mathcal{S}$ festgelegt werden, je nach Eignung einer Ressource zur Ausführung eines Pfades und dem daraus resultierenden Energieverbrauch und so weiter. Die Wahl der entsprechenden Werte hängt jedoch von der konkreten Anwendung ab und wird daher in dieser Arbeit nicht weiter diskutiert, außer wenn Werte für bestimmte Systemeigenschaften definiert werden müssen, siehe etwa die nächsten beiden Unterkapitel.

5.2.1. Das einstufige KHS

In diesem Ansatz werden die Pfade eines Systems $\mathcal{S}$ direkt auf die Ressourcen verteilt. Dies geschieht analog zu der Beschreibung aus Unterkapitel 5.1. Als Beispiel stelle man sich das Pfadsystem aus Abbildung 5.4 vor. Die Aufgabe des KHS ist nun, die Pfade auf eine gegebene Menge von Ressourcen abzubilden. Dazu sind jedoch mehrere Adaptionen des KHS nötig, die nun besprochen werden.

5.2.1.1. Beschränkung der Übernahmehäufigkeit von Pfaden

Jeder Pfad muss im gesamten verteilten System genau einmal übernommen werden. Denn wenn er gar nicht übernommen würde, dann könnte die Semantik des Systems $\mathcal{S}$ möglicherweise nicht erfüllt werden. Ein mehrfaches Übernehmen eines Pfades könnte hingegen zu Inkonsistenzen führen: Wenn man beispielsweise an den Palettierungsauftrag in Kapitel 2.1 denkt, so könnten durch eine mehrfache Ausführung zuviele Werkstücke geliefert werden, was insbesondere dann ein Problem ist, wenn nicht genügend Staukapazität vorhanden ist.

Die erste Bedingung kann durch eine ausreichende Anzahl von zur Verfügung stehenden Ressourcen erfüllt werden. Für die Erfüllung der zweiten Bedingung muss bei Übernahme eines Pfades X_i nur der zugehörige Suppressor $S_{i\gamma}$ hoch genug eingestellt werden, dass der Pfad durch das KHS auf keiner anderen Ressource nochmal übernommen werden kann.

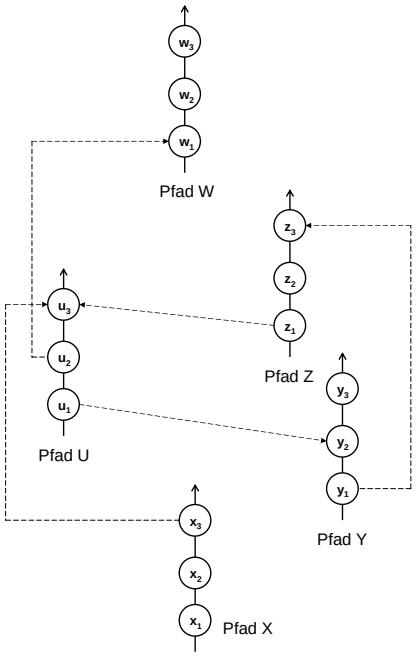


Abbildung 5.4.: Ausführbares Pfadsystem, dessen Pfade durch das KHS zu Ressourcen zugeordnet werden sollen

Nun definiere man:

$$S_\infty := \max_{\substack{X_i \in \mathcal{S} \\ \gamma \text{ aus Ressourcen}}} E_{i\gamma} + \sum_{\substack{X_i \in \mathcal{S} \\ \gamma \text{ aus Ressourcen}}} A_{i\gamma} \tag{5.7}$$

Satz 5.2.1. *Sei $S_{i\gamma} := S_\infty$ für alle Pfade $X_i \in \mathcal{S}$ und jede Ressource γ im verteilten System. Dann tritt jeder Pfad, der durch das KHS vergeben wird, zu jedem Zeitpunkt höchstens einmal im verteilten System auf.*

Beweis. Sei der Pfad X_i bisher nicht vergeben. Wenn er also zum ersten Mal von einer Ressource γ übernommen wurde, so sendet diese den Suppressor S_∞ an alle Ressourcen. Die anderen Ressourcen haben den Pfad nicht übernommen, da sie nicht den größten Eignungswert hatten, und nach der Übernahme ist der modifizierte Eignungswert *aller Ressourcen* für diesen Pfad aber höchstens 0, da S_∞ größer als ihr jeweiliger Eignungswert plus der Summe aller möglichen Acceleratoren ist. Also wird der Pfad nicht mehr übernommen. □

Dieser Satz gibt vor, wie die Suppressoren im KHS gewählt werden müssen, um die zu Beginn dieses Unterkapitels beschriebenen Bedingungen zu erfüllen. Wenn eine Ressource einen Pfad verliert, so wird der zugehörige Suppressor nicht mehr gesendet und eine Selbstheilung beginnt (soweit möglich). Der Satz stellt also keine Einschränkung an die Funktionalität des KHS dar.

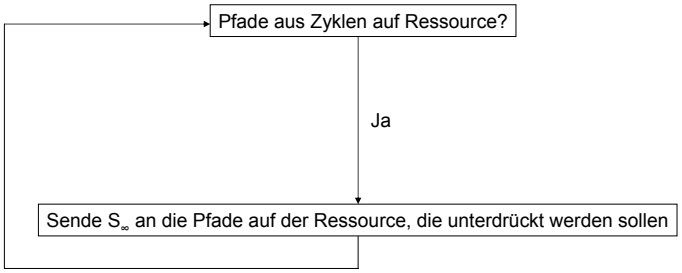


Abbildung 5.5.: Ablaufschema von Suppressoren zur partiellen Unterdrückung von Pfaden

5.2.1.2. Gegenseitiger Ausschluß bei der Übernahme von Pfaden

Aus den Betrachtungen in Kapitel 2 geht hervor, dass es Pfade Y und Z eines Systems $\mathcal{S}$ geben kann, die nicht zusammen auf einer Ressource ausgeführt werden dürfen, weil sie in einem Zyklus bezüglich $<$ liegen. Als Beispiel betrachte man die Pfade Y, Z und U in Abbildung 5.4, welche in solch einem Zyklus liegen. Das KHS muss nun Werkzeuge bereitstellen, durch die ein solcher gegenseitiger Ausschluß von Pfaden ermöglicht werden kann.

Dazu wurde im einstufigen KHS ein weiterer *lokaler Suppressortyp zur partiellen Unterdrückung* von Pfaden eingeführt: Beim Einlesen der Konfigurationsdateien in die Ressourcen wird jeder Ressource mitgeteilt, welche Pfade in einem Zyklus liegen. Wenn eine Ressource nun einen Pfad aus einem Zyklus über nimmt, so sendet sie pro Hormonzyklus je einen lokalen Suppressor zur partiellen Unterdrückung an alle Pfade aus dem Zyklus auf sich selbst. Diese Vorgehensweise wird in Abbildung 5.5 skizziert.

Satz 5.2.2. *Sei der Wert des lokalen Suppressors zur partiellen Unterdrückung auf allen Ressourcen für alle Pfade auf S_∞ wie in Gleichung* (5.7) *gesetzt. Dann gilt: Solange eine Ressource γ einen Pfad X_i aus einem Zyklus Z ausführt, übernimmt sie keinen anderen Pfad des Zyklus.*

Beweis. Führe γ noch keinen Pfad aus Z aus. Wenn sie dann einen Pfad $X_i \in Z$ übernimmt, so sendet sie an alle Pfade aus dem Zyklus auf sich selbst den Suppressor S_∞. Bei allen späteren Entscheidungen über die Übernahme eines Pfades aus Z auf γ ist dessen modifizierter Eignungswert aber höchstens 0 nach Definition von S_∞. Dies gilt, solange X_i auf γ ausgeführt wird. □

Bemerkung 5.2.1. Ein Sonderfall tritt ein, wenn man Zyklen optimiert. Dazu betrachte man Abbildung 5.4. Man kann die Pfade Z und U aus dem Zyklus der Pfade Y, Z und U mittels der Anordnung $A_{u_1}^{z_3}$ für die Ausführung auf Ressourcen optimieren. Das bedeutet also, dass die Pfade Z und U auf einer Ressource in dieser Reihenfolge ausgeführt werden können. Daher muss man alle Ressourcen in diesem

Beispiel so konfigurieren, dass sie einen Suppressor zur partiellen Unterdrückung an Y schicken, falls sie Z *oder* U übernommen haben, und dass sie einen Suppressor zur partiellen Unterdrückung an Z *und* U schicken, falls sie Y übernommen haben.

Verallgemeinert bedeutet das, dass bei optimierten Zyklen die Pfade eines Zyklus, die nacheinander ausgeführt werden können, sich nicht auf einer Ressource durch einen Suppressor zur partiellen Unterdrückung gegenseitig blockieren dürfen, aber alle anderen Pfade des Zyklus. Im später folgenden Unterkapitel 5.2.3 wird zusätzlich beschrieben, wie eine korrekte Ausführungsreihenfolge bei den Pfaden auf einer Ressource erreicht wird, auch innerhalb eines Zyklus.

Auswirkungen auf das Zeitverhalten Das Zeitverhalten für Selbstkonfiguration und Selbstheilung (auch Selbstoptimierung) ändert sich durch die Einführung des Suppressors zur partiellen Unterdrückung nicht. Wenn man die Beweise in [BPR08] zum Zeitverhalten betrachtet, dann sieht man, dass sie genau von sich ändernden globalen Hormonwerten beliebigen Typs abhängen. Hormonwerte ändern sich dort *genau* bei einer Übernahme oder dem Verlust von Tasks im verteilten System. Dies ist bei den Suppressoren zur partiellen Unterdrückung nur lokal der Fall: Sie werden genau so lange von einer Ressource ausgesendet, solange ein Pfad aus einem Zyklus auf ihr ausgeführt wird. Daher gelten die Beweise auch für das KHS mit Suppressoren zur partiellen Unterdrückung.

Auswirkungen auf das Datenaufkommen Gemäß [BPR08] ändert sich das Datenaufkommen im verteilten System nicht, da die Suppressoren zur partiellen Unterdrückung nur lokal gesendet werden, also von der Ressource an sich selbst.

5.2.1.3. Termination von Pfaden im einstufigen KHS

Die Ausführungszeit von Pfaden eines ausführbaren Systems ist endlich. Daraus ergibt sich, dass das KHS die Pfade nur genau so lange im verteilten System halten muss, so lange ihre Ausführung dauert. Also muss das KHS um einen Mechanismus zur *verteilten Termination von Pfaden* erweitert werden. Er muss es ermöglichen, dass die Pfade auf den Ressourcen konsistent terminiert werden können.

Definition 5.2.1 (Termination eines Pfades)**.** *Gegeben sei ein verteiltes System von Ressourcen auf denen Pfade genau durch das KHS verteilt werden. Ein Pfad heißt terminiert genau dann wenn seine Ausführung auf einer Ressource beendet wurde, das KHS den Pfad auf der ausführenden Ressourcen entfernt, und sich keine Ressource im verteilten System mehr für diesen Pfad durch das KHS bewirbt.*

Definition 5.2.2 (Termination eines Systems von Pfaden)**.** *Ein System von Pfaden heißt terminiert genau dann wenn alle seine Pfade terminiert sind.*

Der Mechanismus, der die Termination eines Pfades X_i ermöglicht, wird durch das Aussenden eines neuen Typs von Hormonen, den *Terminationssuppressoren*, realisiert. Dabei wird parallel zu dem „normalen“ Suppressor, der bei der Übernahme von X_i von einer Ressource ausgesendet wird, ein Terminationssuppressor F_i an alle Ressourcen im verteilten System gesendet, durch den angezeigt wird, dass X_i ablaufwillig ist und nicht terminiert werden soll. Die Ressource sendet den Terminationssuppressor für X_i in jedem Hormonzyklus, bis der Pfad abgearbeitet ist und somit terminiert werden soll.

Wenn eine Ressource nur Eignungswerte mit dem Wert 0 für X_i und keine Terminationssuppressoren dafür empfängt, dann kann der Pfad unter bestimmten Bedingungen, die in den nächsten beiden Abschnitten erläutert werden, terminiert werden. Dies wird in Abbildung 5.6 dargestellt.

Die folgenden Betrachtungen zum Zeitverhalten beziehen sich genau auf die Terminierung von Pfaden; eventuelle Einflüsse auf Selbstkonfiguration oder Selbstheilung werden nicht angestellt, da man davon ausgehen kann, dass dies bei terminierungswilligen Pfaden nicht mehr benötigt wird.

Zeitverhalten des Terminationsmechanismus beim Start der Ressourcen Da alle Ressourcen mehr oder weniger gleichzeitig gestartet werden, sind sie in einem konsistenten Zustand. Das bedeutet, dass alle Ressourcen diesselben Informationen über die auszuführenden Pfade besitzen. Da der Terminationsmechanismus einige Hormonzyklen benötigt, bis er einen Pfad auf einer Ressource terminiert, kann er sofort beim Start der Ressourcen aktiviert werden. Die Verzögerung bei der Terminierung ergibt sich aus den Anforderungen, die im nächsten Abschnitt beschrieben werden.

Da laut Voraussetzung genügend Ressourcen zur Ausführung aller Pfade eines Systems im verteilten System vorhanden sind, werden auch keine Pfade aufgrund von Ressourcenmangel und einer resultierenden eingeschränkten Übernahme von Pfaden terminiert.

Zeitverhalten des Terminationsmechanismus während der Ausführung der Ressourcen Wenn eine Ressource keinen Terminationssuppressor mehr für einen Pfad X_i empfängt, dann kann das erstens daran liegen, dass die Ausführung von X_i durch einen Software- oder Hardwarefehler abgebrochen wurde oder zweitens, dass X_i terminiert werden soll. Eine Ressource muss nun zwischen diesen beiden Fällen unterscheiden können.

Satz 5.2.3. *Sei X_i ein Pfad, der auf einer Ressource γ ausgeführt werde. Die Verfallszeit eines Hormones betrage a Hormonzyklen auf allen Ressourcen. Die folgenden Bedingungen seien wahr:*

1) γ sendet im Hormonzyklus h_γ mit $h_\gamma \in \mathbb{N}$ zum letzten Mal einen Terminationssuppressor für Pfad X_i.

2) γ läuft ab Hormonzyklus $h_\gamma + 1$ fehlerfrei.

3) Eine beliebige Ressource δ empfängt einen Terminationssuppressor für X_i zum letzten Mal im Hormonzyklus h_δ mit $h_\delta \in \mathbb{N}$.

4) Eine beliebige Ressource δ empfängt zwischen Hormonzyklus h_δ und $h_\delta + a + 2$ keinen echt positiven Eignungswert für X_i.

5) Eine beliebige Ressource δ beendet die Bewerbung für X_i als letzte Aktion im Hormonzyklus $h_\delta + a + 2$. γ entfernt den Pfad X_i.

Dann gilt: X_i wurde terminiert gemäß Definition 5.2.1.

Beweis. Annahme, dass die Ausführung von X_i auf γ im Hormonzyklus h_γ abgebrochen wurde. Dann ist X_i jetzt nicht mehr im verteilten System vorhanden, da ein Pfad zu jedem Zeitpunkt höchstens einmal im verteilten System vorkommt, wie bereits besprochen wurde. Nun gilt für eine beliebige Ressource δ: Im Hormonzyklus $h_\delta + a$ sind die für X_i von γ gesendeten Hormone abgelaufen. Das heißt, im Hormonzyklus $h_\delta + a + 1$ werden sie von δ nicht mehr mitgerechnet. Also ist der Suppressor S_∞, der von γ für X_i ausgesendet wurde, nicht mehr gültig. Da nach Unterkapitel 5.1.2 aber alle (nicht modifizierten) Eignungswerte echt größer als Null sind und laut Voraussetzung genügend Ressourcen zur Ausführung da sind, sendet mindestens eine Ressource δ_1 im Hormonzyklus $h_{\delta_1} + a + 1$ nun seinen echt positiven Eignungswert an alle Ressourcen. Da die Dauer eines Hormonzyklus auf jeder Ressource gleich ist, und alle Ressourcen gleichzeitig (oder mit einem infinitesimal kleinen Zeitabstand) den Terminationssuppressor zum letzten Mal erhalten haben, erhalten sie diesen positiven Eignungswert spätestens im Hormonzyklus $h_\delta + a + 2$ und Bedingung 4) ist verletzt, Widerspruch. □

Bemerkung 5.2.2. Dieser Satz beschreibt nun, wann der Terminationsmechanismus einer Ressource greifen muss: Wenn die Ressource nämlich *nicht* bis zum $h_\delta + a + 2$. Hormonzyklus mindestens einen positiven Eignungswert erhalten hat, heißt das, dass γ den Pfad X_i nach wie vor allokiert hat und Suppressoren dafür aussendet. Dann bedeutet der Wegfall des Terminationssuppressor tatsächlich den Wunsch nach Termination, und jede Ressource entscheidet folglich für sich, X_i als terminiert anzusehen. Die Terminierungszeit für einen Pfad beträgt also $a+2$ Hormonzyklen auf jeder Ressource. Über alle Ressourcen hinweg gesehen beträgt die Terminierungszeit für einen Pfad damit im schlechtesten Fall $a + 3$ Hormonzyklen vom letzten Aussenden einen Terminationssuppressors bis zu dem Zeitpunkt, zu dem die letzte Ressource den Pfad auf sich als terminiert betrachtet, was an dem maximalen Versatz von einem Hormonzyklus der Ressourcen liegt.

Bemerkung 5.2.3. Eine wichtige Tatsache in diesem Beweis ist, dass man für jede Ressource einen lokalen Zeitpunkt angeben kann, zu dem sie den Terminationssuppressor X_i zum letzten Mal empfangen hat.

Bemerkung 5.2.4. Das sich ergebende Verfahren funktioniert auch, wenn mehrere Pfade im selben Zeitraum terminiert werden sollen, da das Auswerten und Senden der Hormone für verschiedene Pfade quasi gleichzeitig auf einer Ressource durchgeführt wird.

Bemerkung 5.2.5. Bedingung 2) in Satz 5.2.3 hat eine wichtige Bedeutung für einen Sonderfall der Termination. Falls nämlich nach dem Start der Termination die Pfadallokation durch einen Fehler abgebrochen wird, dann kann es zu Inkonsistenzen kommen: Wenn ein Pfad X_i auf Ressource γ abgebrochen wird, nachdem der Terminationssuppressor nicht mehr ausgesendet wird, dann terminieren alle Ressourcen folgerichtig X_i. Allerdings bietet γ alleine für diesen Pfad und übernimmt ihn wenn möglich wegen des Rücksetzens der Variablen. Daher wurde im KHS ein weiterer Mechnismus eingebaut, durch den auf jeder Ressource bis zum Hormonzyklus $h_\delta + a + 4$ auf Eignungswerte und Terminationssuppressoren von X_i gelauscht wird. Sollte ein solcher auftauchen, so wird auf allen Ressourcen lokal entschieden, dass die Termination für X_i zurückgenommen wird. Dann müssen die Ressourcen einen weiteren Terminationsvorgang anstoßen.

Nun wird beschrieben wie man die Terminationszeit unter der Voraussetzung berechnen kann, dass der eben beschriebene Sonderfall eintritt. Wenn man davon ausgeht, dass q die zeitinvariante Wahrscheinlichkeit mit $0 \leq q \leq 1$ dafür ist, dass eine Terminierung genau wie eben beschrieben fehlschlägt, dann liegt die Wahrscheinlichkeit P, dass der Pfad nach $a + 3 + n(a + 5)$ Hormonzyklen terminiert ist bei

$$P\{\text{Pfad terminiert nach } a + 3 + n(a + 5) \text{ Hormonzyklen}\} = (1 - q) \sum_{i=0}^{n} q^i$$

Im Fall $q = 1$ wird die Terminierung nie funktionieren. Andernfalls ist $0 \leq q < 1$ wahr und es gilt

$$P\{\text{Pfad terminiert nach } a + 3 + n(a + 5) \text{ Hormonzyklen}\} = 1 - q^{n+1}$$

Diese Wahrscheinlichkeit konvergiert gegen 1. Wenn man also eine konkrete Fehlerwahrscheinlichkeit q_1 mit $0 \leq q_1 < 1$ gegeben hat, kann man aus der letzten Formel das kleinste n berechnen, so dass die Wahrscheinlichkeit der Terminierung des Pfades nach $a + 3 + n(a + 5)$ Hormonzyklen gerade größer oder gleich einer gegebenen Sicherheitsschranke s ist. Daraus erhält man dann die Zeit, nach der der Pfad mit einer Wahrscheinlichkeit von mindestens s terminiert ist.

Als Beispiel gelte $q_1 = 0,1$ und für die gewünschte Sicherheitsschranke gelte $s = 0,9$. Wegen $1 - 0,1 = 0,9$ kann man also davon ausgehen, dass der Pfad nach $a + 3$ Hormonzyklen schon terminiert ist. Wenn $s = 0,99$ gilt ergibt diesselbe Rechnung, dass der Pfad nach $2a + 8$ Hormonzyklen terminiert ist.

Bemerkung 5.2.6. Eine wesentlich Stütze des Satzes ist die Eigenschaft, dass immer genügend Ressourcen zur Ausführung aller Pfade vorhanden sind. Also existiert

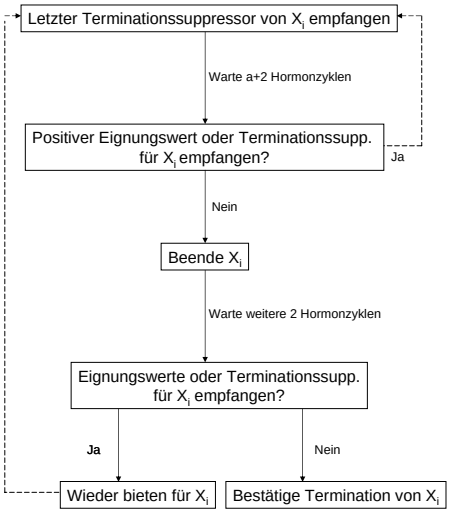


Abbildung 5.6.: Algorithmus zur Terminierung eines Pfades im KHS

eine Ressource δ_1, die Eignungswerte für ihn aussendet. Nur dadurch wird gewährleistet, dass eine Ressource δ_1 existiert, die einen Eignungswert für X_i sendet, wenn er auf γ abgebrochen wird. Wenn solch ein δ_1 nicht existiert oder selber fehlerbehaftet ist und ausfällt, dann kann es passieren, dass Pfade terminiert werden, obwohl sie nicht komplett ausgeführt wurden. Solche Ausfallszenarien, die durch einen Mangel an Ressourcen verursacht werden, überschreiten den Rahmen dieser Arbeit und werden deshalb nicht weiter diskutiert.

Der Algorithmus zur Terminierung eines Pfades wird in Abbildung 5.6 dargestellt.

Auswirkungen auf das Datenaufkommen Durch die Einführung der Terminationssuppressoren ändert sich auch das Datenaufkommen im KHS, da zusätzlich zum Suppressor ein Terminationssuppressor gesendet wird. Dies wird in Unterkapitel 5.2.4 beschrieben.

5.2.2. Das zweistufige KHS

Der wesentliche Nachteil des einstufigen KHS ist, dass Pfade aus verschiedenen Teilsystemen, die gemäß Algorithmus 2 (zur Erstellung eines Ausführungsplanes) entstanden sind, auf eine gemeinsame Ressource zugewiesen werden können. Da auf einer Ressource höchstens ein Pfad zu einem Zeitpunkt ausgeführt wird, sind die Startzeiten von Pfaden verschiedener Teilsysteme in höchsten Maß von der Pfadverteilung des KHS abhängig. Formal gesprochen ist die Startzeit $t_{k(x)}$ eines Pfades also von der Verteilungsfunktion r für Pfade, wie in Kapitel 4 definiert, abhängig.

Wenn man diese zeitlichen Interferenzen, die zwischen Pfaden verschiedener Teilsysteme entstehen können, vermeiden möchte, dann ist es sinnvoll, Pfade verschiedener Teilsysteme auf verschiedene Ressourcen zuzuweisen.

Das zweistufige KHS leistet diese Anforderungen:

1. Stufe: Zunächst werden Ressourcen exklusiv für die verschiedenen Teilsysteme $\mathcal{S}_1$, ..., $\mathcal{S}_m$ reserviert. Dies geschieht, indem sogenannte Teilsystemtasks, kurz T-Tasks durch das KHS auf den Ressourcen vergeben werden. Eine T-Task steht dabei für ein Teilsystem und pro Ressource wird höchstens eine T-Task vergeben. Das KHS sendet permanent Hormone, wie im Unterkapitel 5.1 besprochen. So werden die organischen Eigenschaften für die T-Tasks gewährleistet.

2. Stufe: Sobald die T-Tasks stabil auf den Ressourcen vergeben sind (also die Eignungswerte für die T-Tasks Null sind), beginnt das KHS in seiner zweiten Stufe die Pfade auf den Ressourcen zu verteilen. Allerdings bieten Ressourcen hier genau für Pfade mit, die zum Teilsystem der auf der Ressource laufenden T-Task gehören. Eine Ressource, welche keine T-Task hält, bietet somit für keinen Pfad mit. Wenn allerdings Hormone gesendet werden, dann werden auch sie permanent gesendet, um die organischen Eigenschaften sicherzustellen.

5.2.2.1. Bestimmung der Übernahmehäufigkeit

1. Stufe: Für die Vergabe von T-Tasks im verteilten System ergeben sich andere Anforderungen als für Pfade. Hier muss nämlich zusätzlich berücksichtigt werden, dass Zyklen bezüglich $<$ auf den Pfaden vorkommen können, und dass man daher für die Ausführung eines Teilsystems möglicherweise mehrere Ressourcen benötigt. Daraus folgt, dass eine ausreichende Anzahl von T-Tasks für das Teilsystem allokiert werden muss. Im zweistufigen KHS muss diese Anzahl explizit durch die Wahl der Eignungswerte und Suppressoren, wie unten beschrieben, festgelegt werden, wohingegen dies im einstufigen KHS automatisch durch die Suppressoren zur partiellen Unterdrückung geschieht. Wenn also der größte Zyklus eines Teilsystems aus vier Pfaden besteht, dann müssen durch das zweistufige KHS mindestens vier T-Tasks (und somit mindestens vier Ressourcen) allokiert werden. Dies geschieht beim Einsatz des einstufigen KHS automatisch, da Ressourcen, die Pfade aus Zyklen ausführen, Suppressoren zur partiellen Unterdrückung an sich selbst aussenden.

Nach der Beschreibung der ersten Stufe muss also gewährleistet werden, dass pro Ressource höchstens eine T-Task vergeben wird. Dies kann dadurch realisiert werden, dass eine Ressource bei Übernahme einer T-Task lokal einen Lastsuppressor mit dem Wert S_∞ sendet. Dann wird keine weitere T-Task eines beliebigen Typs übernommen. Der formale Beweis geht analog zu dem Beweis von Satz 5.2.1.

Bemerkung 5.2.7. Der Wert S_∞ kann im zweistufigen KHS genau wie im einstufigen KHS gewonnen werden, indem man Pfade und T-Tasks als Pfade betrachtet und dann die Rechnung aus Gleichung (5.7) durchführt.

Nun muss gewährleistet werden, dass genug Ressourcen zur Ausführung eines Teilsystems allokiert werden.

Lemma 5.2.1. *Sei $U_{Z_{\max}}$ die Anzahl der benötigten Ressourcen zur Ausführung des größten Zyklus in einem Teilsystem. Dann werden mindestens $U_{Z_{\max}}$ Ressourcen für die Ausführung des gesamten Teilsystems benötigt.*

Beweis. Klar. □

Bemerkung 5.2.8. Dieses Lemma formuliert eine *notwendige* Bedingung, unter der eine Ausführung möglich sein kann. Wenn weniger Ressourcen allokiert werden als darin gefordert, so wird eine Ausführung des Teilsystems scheitern. Wenn mindestens $U_{Z_{\max}}$ Ressourcen allokiert werden, dann klappt die Ausführung unter der Bedingung, dass die allokierten Ressourcen ordnungsgemäß arbeiten (beispielweise einen ausreichenden Energievorrat besitzen).

Lemma 5.2.1 gibt also an, wieviele Ressourcen und damit T-Tasks mindestens für die Ausführung eines Teilsystems allokiert werden müssen.

Satz 5.2.4. *Sei $U_{Z_{\max}}$ wie in Lemma 5.2.1 gegeben. Dann ist eine notwendige Bedingung für die Ausführung eines Teilsystems $\mathcal{S}_i$ an den minimalen Eignungswert $E^i_{\min}$ und den maximalen Suppressor $S^i_{\max}$ der zugehörigen T-Task T_i:*

$$U_{Z_{\max}} \leq \left\lceil \frac{E^i_{\min}}{S^i_{\max}} \right\rceil$$

Beweis. Sei also $U_{Z_{\max}}$ durch Betrachten des Pfadsystems gemäß Lemma 5.2.1 gegeben. Dann müssen die Werte von $E^i_{\min}$ und $S^i_{\max}$ gemäß Ungleichung (5.4) gewählt werden, damit das Teilsystem $\mathcal{S}_i$ ausführbar ist. Daraus folgt dann die Behauptung. □

Bemerkung 5.2.9. Ein Anwender muss durch die Wahl der Werte $E^i_{\min}$ und $S^i_{\max}$ gemäß diesem Satz für jedes Teilsystem $\mathcal{S}_i$ festlegen wieviele T-Tasks vom Typ T_i mindestens übernommen werden sollen.

Nun wird man im Allgemeinen auch noch eine obere Schranke für die Übernahmehäufigkeit der T-Tasks definieren wollen, um zu verhindern, dass durch ein Teilsystem zuviele Ressourcen in Beschlag genommen werden. Dies könnte man analog zum letzten Satz durch eine entsprechende Einstellung der maximalen Eignungswerte und minimalen Suppressoren von Teilsystemen durchführen. Insgesamt ist

diese Lösung jedoch wegen der Zusammenhänge zwischen minimalen und maximalen Eignungswerten und Suppressoren unpraktisch. Daher wurde in dieser Arbeit eine andere Lösung entwickelt: Ein Benutzer kann in der Beschreibung des Teilsystems die maximale Anzahl u_i festlegen, die eine zu einem Teilsystem $\mathcal{S}_i$ gehörige T-Task T_i höchstens übernommen werden kann. Da jede Ressource zu Systemstart die Beschreibung einliest, kennt sie diese Zahl für jedes Teilsystem. Die Funktionsweise zur Begrenzung der T-Task-Übernahme wird im Folgenden beschrieben.

Satz 5.2.5. *Gegeben sei ein verteiltes System von Ressourcen, auf dem die T-Task T_i verteilt werde. Weiter sei $u_i \in \mathbb{N}$ fest gegeben und jeder Ressource bekannt. Weiter seien die folgenden Voraussetzungen wahr:*

1. *Jede Ressource zähle vor dem Senden der eigenen Hormonwerte in jedem Hormonzyklus für jede T-Task die Anzahl der empfangenen Suppressoren.*

2. *Sei γ eine Ressource, die im Hormonzyklus h_γ eine T-Task vom Typ T_i übernehme. Die Zählung gemäß Punkt 1 für T_i ergebe $u_i - 1$ empfangene Suppressoren im Hormonzyklus $h_\gamma + 1$.*

   γ erhöht im Hormonzyklus $h_\gamma + 1$ den Suppressor für T_i auf S_∞.

3. *Sei δ eine Ressource. Solange δ beim Zählen der Suppressoren gemäß Punkt 1 ein Vorkommen von u_i Suppressoren für T_i erkennt und sie T_i ausführt, so sendet sie den Suppressor S_∞ für diese T-Task.*

Dann werden im verteilten System höchstens u_i T-Tasks vom Typ T_i allokiert.

Beweis. Seien bereits $u_i - 1$ T-Tasks vom Typ T_i auf den Ressourcen verteilt. Nach oben Gesagtem tragen also $u_i - 1$ Ressourcen T_i und senden dafür Suppressoren aus. Wenn nun Punkt 2 der Voraussetzungen wahr ist, dann wird keine andere Ressource T_i übernehmen wollen, weil sie weiß, dass γ den dafür höchsten Eignungswert besitzt. Nachdem γ T_i übernommen hat, sendet sie aber sofort den Suppressor S_∞ für diese T-Task zu allen Ressourcen. Deshalb wird auch keine weitere Ressource T_i übernehmen. □

Bemerkung 5.2.10. Der in diesem Satz beschriebene Mechanismus bietet eine sichere Möglichkeit zur Einschränkung der Übernahme von T-Tasks unabhängig von den gewählten Hormonwerten, vergleiche dazu mit den Ungleichungen (5.4).

Bemerkung 5.2.11. Das KHS wird in diesem Satz dahingegehend verändert, dass sich die Suppressorwerte für eine T-Task dynamisch verändern können. Wenn eine Ressource nämlich feststellt, dass genügend T-Tasks eines Types vorhanden sind, und sie eine solche ausführt, dann erhöht sie den dafür ausgesendeten Suppressor auf S_∞.

Bemerkung 5.2.12. Punkt 3) dieses Satzes dient der Redundanz. Es ist nicht unbedingt notwendig, dass alle Ressourcen, die de T-Task T_i tragen, im Grenzfall S_∞ aussenden. Diese Variante wurde jedoch aus Symmetriegründen gewählt.

Auswirkungen auf das Zeitverhalten Das Zeitverhalten bei der Selbstkonfiguration entspricht offensichtlich genau dem in Unterkapitel 5.1 beschriebenen. Das Zeitverhalten bei der Selbstheilung ändert sich aber nun wegen der dynamischen Erhöhung der Suppressoren.

Satz 5.2.6. *Gegeben sei ein verteiltes System von Ressourcen mit einem zweistufigen KHS wie bisher beschrieben. Sei T_i die zu einem Teilsystem $\mathcal{S}_i$ gehörende T-Task. Dann gilt: Wenn eine Ressource δ im Hormonzyklus h_δ zum letzten Mal einen Suppressor für T_i von einer bestimmten Ressource γ erhält, so kann sie im schlimmsten Fall frühestens ab dem Hormonzyklus $h_\delta + a + 2$ über die Übernahme von T_i entscheiden.*

Beweis. Wenn δ den Suppressor von γ im Hormonzyklus h_δ zum letzten Mal erhält, dann verwirft sie ihn im Hormonzyklus $h_\delta + a + 1$. Falls sie T_i selber trägt, sendet sie nur noch ihren vordefinierten Suppressor dafür; sonst sendet sie keinen Suppressor für T_i. Wegen der Asynchronität der Ressourcen und weil T_i auf mehreren Ressourcen laufen kann, muss δ noch einen weiteren Hormonyzklus warten, bis ihr eigener Eignungswert für T_i stabil ist. □

Bemerkung 5.2.13. Global gesehen ist wegen der Asynchronität der Ressourcen eine obere Zeitschranke für die Selbstheilung einer T-Task also $a + 3$ Hormonzyklen, gerechnet vom letzten Aussenden des Suppressors bis zur einer erneuten Übernahme. Eine obere Schranke für die Selbstheilung im allgemeinen Fall, dass alle T-Tasks gleichzeitig ausfallen, wird im später folgenden Unterkapitel 5.2.4 berechnet.

Auswirkungen auf das Datenaufkommen Da nur die Werte des Suppressors dynamisch verändert werden, ändert sich am Datenaufkommen durch den in Satz 5.2.5 beschriebenen Mechanismus nichts gegenüber den bekannten Abschätzungen.

2. Stufe: Für die Vergabe der Pfade gelten die gleichen Anforderungen und Eigenschaften wie im einstufigen KHS.

Auswirkungen auf das Zeitverhalten Wie im einstufigen KHS unter Berücksichtigung, dass die Pfade verschiedener Teilsysteme nun zeitlich parallel vergeben werden, siehe Unterkapitel 5.2.4.

Auswirkungen auf das Datenaufkommen Wie im einstufigen KHS unter Berücksichtigung, dass nun nicht mehr alle Ressourcen für alle Pfade bieten, siehe Unterkapitel 5.2.4.

5.2.2.2. Gegenseitiger Ausschluß bei der Übernahme von Pfaden

1. Stufe: Da pro Ressource höchstens eine T-Task läuft und dies, wie oben beschrieben, durch einen Lastsuppressor S_∞ erreicht wird, sind hier keine Modifikationen nötig.

2. Stufe: Der gegenseitige Ausschluß bei der Übernahme von Pfaden wird genau wie im einstufigen KHS realisiert. Es ergeben sich daher bezüglich Selbstkonfiguration und Selbstheilung keine Änderungen im Zeitverhalten oder Datenaufkommen gegenüber dem einstufigen KHS.

5.2.2.3. Terminierung von T-Tasks und Pfaden im zweistufigen KHS

Auch im zweistufigen KHS soll eine Terminierung von T-Tasks und Pfaden möglich sein, um nach der erfolgreichen Ausführung eines Teilsystems von Pfaden auf den Ressourcen für andere Teilsysteme Platz zu machen.

Dafür muss zunächst aber definiert werden, was die Terminierung einer T-Task bedeutet.

Definition 5.2.3 (Terminierung einer T-Task)**.** *Gegeben sei ein verteiltes System von Ressourcen auf denen T-Tasks und Pfade genau durch das zweistufige KHS verteilt werden. Eine T-Task T_i heißt terminiert genau dann wenn auf jeder ausführenden Ressource die zu T_i gehörenden Pfade terminiert wurden, das KHS T_i auf den ausführenden Ressourcen entfernt hat, und sich keine Ressource im verteilten System mehr für T_i durch das KHS bewirbt.*

Der Terminierungsmechanismus geht beim zweistufigen KHS ganz ähnlich wie der beim einstufigen KHS: Eine Ressource sendet einen Terminierungssuppressor für einen Pfad genau so lange, bis sie ihn ausgeführt hat. Sie sendet einen Terminierungssuppressor für eine T-Task T_i genau so lange, bis alle Pfade, die zum Teilsystem von T_i gehören, terminiert sind.

Die Bedingungen, unter denen eine Ressource nun einen Pfad oder eine T-Task aufgrund des Ausbleibens von Terminierungssuppressoren terminieren darf werden im Folgenden beschrieben. Um ein besseres Verständnis zu gewährleisten, wird zunächst der Terminierungsmechanismus der zweiten und dann der ersten Stufe erklärt.

2. Stufe: Die Terminierung von Pfaden funktioniert ganz ähnlich wie im einstufigen KHS. Es muss nur als weitere Bedingung geprüft werden, dass auch die zu den Pfaden gehörenden T-Tasks stabil laufen. Wenn nämlich eine T-Task auf einer Ressource ausfällt, unterbricht diese sofort die Ausführung der zugehörigen Pfade, was zu einer unerwünschten Terminierung führen könnte.

Zeitverhalten Der folgende Satz beschreibt den Terminierungsmechanismus für Pfade im zweistufigen KHS:

Satz 5.2.7. *Laufe eine T-Task T_i und ein Pfad X_i, der zum Teilsystem von T_i gehört, auf einer Ressource γ. Die Verfallszeit eines Hormones betrage a Hormonzyklen auf allen Ressourcen. Die folgenden Bedingungen seien wahr:*

1) γ sendet im Hormonzyklus h_γ mit $h_\gamma \in \mathbb{N}$ zum letzten Mal einen Terminationssuppressor für Pfad X_i.

2) γ läuft ab Hormonzyklus $h_\gamma + 1$ fehlerfrei.

3) Eine beliebige Ressource δ empfängt einen Terminationssuppressor für X_i zum letzten Mal im Hormonzyklus h_δ mit $h_\delta \in \mathbb{N}$.

4) Eine beliebige Ressource δ empfängt zwischen Hormonzyklus h_δ und $h_\delta + a + 3$ keinen echt positiven Eignungswert für X_i und keinen echt positiven Eignungswert für T_i.

5) Eine beliebige Ressource δ beendet die Bewerbung für X_i als letzte Aktion im Hormonzyklus $h_\delta + a + 3$. γ entfernt zusätzlich den Pfad X_i.

Dann gilt: X_i wurde terminiert gemäß Definition 5.2.1.

Beweis. Kann wie im Beweis zu Satz 5.2.3 geführt werden. Der einzige Unterschied ist, dass durch die um einen Hormonzyklus längere Wartezeit bis zur Terminierung eines Pfades auch sichergestellt wird, dass X_i nicht wegen eines fehlerhaften Abbruchs von T_i abgebrochen wurde, vergleiche dazu Satz 5.2.6. □

Bemerkung 5.2.14. Auch im zweistufigen KHS wurde ein Mechanismus zur Wiederbelebung von Pfaden analog zum einstufigen KHS, Bemerkung 5.2.5, implementiert.

Auswirkungen auf das Datenaufkommen Wie im einstufigen KHS unter Berücksichtigung, dass nun nicht mehr alle Ressourcen für alle Pfade bieten, siehe Unterkapitel 5.2.4.

1. Stufe: Wenn die Pfade eines Teilsystems bearbeitet wurden und terminiert sind, sollen die zugehörigen T-Tasks ebenfalls beendet werden, um Platz zu schaffen für weitere Pfade anderer Teilsysteme. Daher sendet eine Ressource für eine T-Task genau dann Terminationssuppressoren per Broadcast, solange noch mindestens ein Pfad des zur T-Task gehörigen Teilsystems auf ihr läuft oder dafür Hormone gesendet werden und die T-Task noch nicht terminiert wurde.

Zeitverhalten Der genaue Ablauf des Terminationsmechanismus geht aus dem folgenden Satz hervor.

Satz 5.2.8. *Laufe eine T-Task T_i im verteilten System. Sei γ eine beliebige Ressource, auf der T_i läuft. Die Verfallszeit eines Hormones betrage a Hormonzyklen auf allen Ressourcen. Die folgenden Bedingungen seien wahr:*

1) γ sendet im Hormonzyklus h_γ mit $h_\gamma \in \mathbb{N}$ zum letzten Mal einen Terminationssuppressor für Pfad X_i.

2) γ läuft ab Hormonzyklus $h_\gamma + 1$ fehlerfrei.

3) Eine beliebige Ressource δ empfängt einen Terminationssuppressor für T_i zum letzten Mal im Hormonzyklus h_δ mit $h_\delta \in \mathbb{N}$.

4) Eine beliebige Ressource δ empfängt zwischen Hormonzyklus h_δ und $h_\delta + a + 3$ keinen echt positiven Eignungswert für T_i und keinen Eignungswert eines Pfades, der zum Teilsystem von T_i gehört.

5) Eine beliebige Ressource δ beendet die Bewerbung für T_i als letzte Aktion im Hormonzyklus $h_\delta + a + 3$. γ entfernt zusätzlich die T-Task T_i.

Dann gilt: T_i wurde terminiert gemäß Definition 5.2.3.

Beweis. Analog zur Argumentation vom Beweis zu Satz 5.2.7. □

Bemerkung 5.2.15. Der Unterschied zur Terminierung von Pfaden ist hier, dass mehrere T-Tasks desselben Typs auf verschiedenen Ressourcen laufen können, und durch den hier vorgestellten Terminationsmechanismus fast gleichzeitig wegen der Asynchronität der Ressourcen terminiert werden.

Bemerkung 5.2.16. Auch hier wurde ein Mechanismus zur Wiederbelebung von Pfaden analog zum einstufigen KHS, Bemerkung 5.2.5, implementiert.

Auswirkungen auf das Datenaufkommen Wie im einstufigen KHS, siehe Unterkapitel 5.2.4.

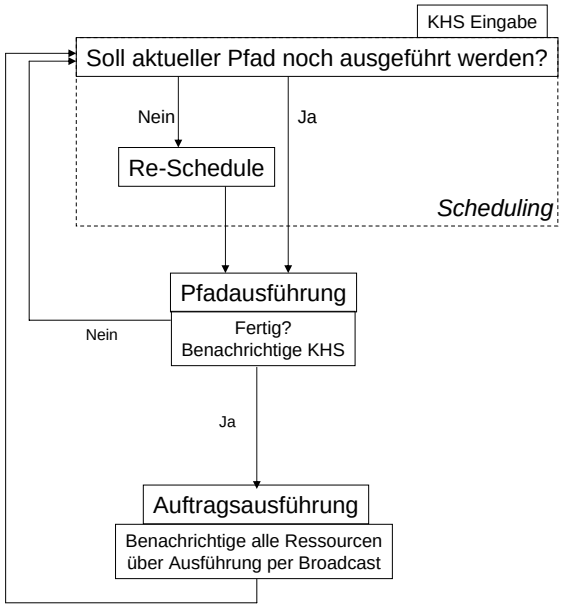


Abbildung 5.7.: Ablauf der lokalen Ausführung eines Pfades

5.2.3. Die Ausführung von Pfaden

Durch das KHS (unabhängig ob ein- oder zweistufig) werden Pfade auf Ressourcen verteilt. Die eigentliche Ausführung wird aber nicht durch das KHS erledigt, sondern muss lokal auf jeder Ressource gesteuert werden. Der Algorithmus für diese Steuerung wird im Folgenden vorgestellt und durch Abbildung 5.7 illustriert.

Die Pfadausführung auf einer Ressource wird zyklisch durchgeführt. Zu Beginn jedes Zyklus wird die Eingabe des KHS überprüft: Es wird geprüft, ob neue Pfade durch das KHS allokiert worden sind, oder ob bereits allokierte Pfade nun nicht mehr von der Ressource ausgeführt werden sollen. Sollte einer dieser beiden Fälle eintreten, so wird ein Scheduling von Pfaden gestartet.

Scheduling: Beim Scheduling von Pfaden wird zunächst geprüft, ob gerade ein Pfad X in der Abarbeitung ist. Wenn dies der Fall ist, dann wird geprüft, ob X auf der Ressource noch ausführbar ist: Wenn nämlich gerade ein weiterer Pfad Y durch das KHS allokiert wurde, dann kann es sein, dass Y vor X ausgeführt werden muss. Dies kann man feststellen, indem man die entsprechenden Einträge in der Adjazenzmatrix zu $<$ anschaut. Wenn dieser Fall nun eintritt, dann wird die Ausführung von X abgebrochen. Genauso wird die Ausführung von X abgebrochen, falls das KHS X der Ressource entzogen hat. Sonst wird X weiter ausgeführt.

Falls gerade kein Pfad in der Abarbeitung ist, oder die Ausführung eines Pfades aus einem der eben genannten Gründe abgebrochen wurde, dann wird nun ein Pfad zur Abarbeitung gesucht. Dabei kommen genau diejenigen Pfade in Betracht, die

der Ressource durch das KHS zugewiesen wurden. Die Pfade werden nun unterteilt nach der Zugehörigkeit zu ihrem Teilsystem und es gibt zwei Strategien, einen Pfad daraus auszuwählen.

Die erste Strategie ist eine *In-Order-Strategie*. Dabei wird der erste Pfad eines Teilsystems zur Ausführung ausgewählt, der noch nicht ausgeführt wurde und der gemäß Algorithmus 2 der nächste in der Reihenfolge der dort etablierten Totalordnung ist. Um hier zu verhindern, dass nur die Pfade eines Teilsystems ausgeführt werden, wird pro Schedulingvorgang zyklisch ein Pfad eines anderen Teilsystems ausgewählt. Diese Strategie ist sehr einfach umzusetzen und nutzt die Informationen aus Algorithmus 2 vollständig aus.

Die zweite Strategie ist eine *Out-Of-Order-Strategie*. Durch Algorithmus 2 wird zwar eine Totalordnung auf den Pfaden eines Teilsystems festgelegt, aber dadurch werden Freiheitsgrade für die Ausführung verdeckt, nämlich, dass es auch innerhalb von Teilsystemen Pfade geben kann, die parallel ausführbar sind, siehe Unterkapitel 3.1. Wenn es nun innerhalb eines Teilsystems zwei parallel ausführbare Pfade X und Y gibt, die laut Algorithmus 2 allerdings in der beschriebenen Reihenfolge ausgeführt werden müssen, dann kann das dazu führen, dass die Ausführung so länger dauert als in der umgekehrten Reihenfolge. Das kann daran liegen, dass X noch auf die Ausführung von Aufträgen anderer Pfade auf anderen Ressourcen warten muss, und somit auch die Ausführung von Y verzögert wird.

Daher wird in dieser Strategie über alle auf der lokalen Ressource auszuführenden Pfade eine Priorisierung aufgestellt. Dabei ist die Priorität eines Pfades um so höher, auf je weniger Aufträge von anderen Pfaden er wartet. Nun wird der Pfad X mit der höchsten Priorität untersucht, ob er ausführbar ist. Dabei wird gemäß Algorithmus 3 geprüft, ob X zu jedem Pfad seines Teilsystems beginnend mit dem ersten noch nicht ausgeführten Pfad in der Reihenfolge, die durch Algorithmus 2 gegeben ist, parallel ausführbar ist. Wenn dies der Fall ist, dann wird X zur Ausführung ausgewählt; anderenfalls wird diese Prozedur mit dem nächsten Pfad der Prioritätenliste durchgeführt. Diese Strategie trägt der Hoffnung Rechnung, dass weniger Latenzzeiten auftreten, wenn ein Pfad ausgeführt wird, der auf so wenige Aufträge wie möglich von anderen Pfaden warten muss. Durch diese Strategie wird auch auf jeden Fall ein Pfad für die Ausführung gefunden, wenn noch mindestens ein noch nicht ausgeführter Pfad auf der Ressource allokiert ist, denn im Extremfall wird dadurch genau die In-Order-Strategie realisiert.

Beide Strategien leisten offensichtlich sowohl für das einstufige als auch für das zweistufige KHS das Gewünschte, so dass bei ihrer Beschreibung keine Differenzierung des zugrundeliegenden KHS vorgenommen wurde. Beide Strategien werden in Kapitel 6 evaluiert.

Pfadausführung: Falls nun ein Pfad, der aus vielen Aufträgen bestehen kann, zur Ausführung bereit steht, wird ein *Auftragszähler* gesetzt. Falls der Pfad gerade

zugewiesen wurde, wird der Zähler auf den ersten Auftrag gesetzt; sonst wird der Auftrag genommen, auf den der Zähler gerade zeigt. Dieser Auftrag wird nun von der Ressource ausgeführt (**Auftragsausführung**), wenn er denn ausführbar ist. Die Ausführbarkeit wird getestet, indem die Vorgängermenge des aktuellen Auftrages untersucht wird. Wenn jeder Auftrag der Vorgängermenge bereits ausgeführt ist, so ist der aktuelle Auftrag ausführbar, siehe dazu Bemerkung 3.2.17 in Kapitel 2. Danach werden alle Ressourcen per Broadcast über die Ausführung des aktuellen Auftrages informiert, so dass sie weiterarbeiten können, falls sie auf diesen Auftrag warten. Dann wird durch Erhöhung des Auftragszählers noch überprüft, ob mit der Ausführung des Auftrages die Ausführung des Pfades beendet wurde. Dies ist genau dann der Fall, wenn der Pfad keine weiteren Aufträge enthält. In diesem Fall wird das KHS über die Beendigung der Ausführung des Pfades informiert, woraufhin sie seine Terminierung gemäß Satz 5.2.3 oder Satz 5.2.7 einleitet. Falls der Auftrag nicht ausführbar ist wird mit dem Scheduling neu begonnen.

Die beschriebene Vorgehensweise stellt also ein *lokales Verfahren* zum Ausführen von Pfaden dar. Ihr Vorteil ist einerseits die Möglichkeit Pfade atomar auszuführen und andererseits wird trotzdem eine Unterbrechbarkeit in dem Sinne zu gewährleistet, dass die Pfadausführung abgebrochen werden kann, wenn der Pfad der Ressource entzogen wird, oder ihr ein neuer Pfad zugewiesen wird, der eine neue Bearbeitungsreihenfolge erfordert.

Eine Evaluation des einstufigen und des zweistufigen KHS und des In-Order- und Out-Of-Order-Schedulings wird in Kapitel 6 durchgeführt. Hier wird auch kurz auf die Implementierungsdetails der verteilten Zustandshaltung bei Pfaden und Aufträgen auf den Ressourcen eingegangen, die allerdings kein Forschungsgegenstand dieser Arbeit ist.

5.2.4. Qualitative Beschreibung von ein- und zweistufigem KHS

In diesem Unterkapitel wird das ein- und das zweistufige KHS bezüglich des Zeitverhaltens, des Datenaufkommens und weiterer Eigenschaften bewertet und verglichen. Es sei nochmals darauf hingewiesen, dass in dieser Arbeit davon ausgegangen wird, dass zu jeder Zeit genügend Ressourcen zur Bearbeitung der Pfade bereit stehen. Es wird angenommen, dass das System von Pfaden aus q Teilsystemen ($q \in \mathbb{N}$) besteht, das heißt also, dass es q T-Tasks gibt, auf die sich Ressourcen im zweistufigen KHS bewerben können. Weiter wird angenommen, dass ein Teilsystem $\mathcal{S}_i$ ($1 \leq i \leq q$) genau k_i Pfade enthalte ($k_i \in \mathbb{N}$). Insgesamt enthält $\mathcal{S}$ also $\sum_{i=1}^{q} k_i$ Pfade. Weiter gehe man davon aus, dass der größte Zyklus pro Teilsystem genau z_i Pfade enthalte ($z_i \in \mathbb{N}$, $z_i \leq k_i$). Dabei ist es egal, ob die Zyklen bereits optimiert wurden oder nicht.

Als weitere Annahme gelte, dass eine T-Task T_i Acceleratoren höchstens an sich

selber aussendet, da verschiedene Teilsysteme unabhängig voneinander laufen sollten. Daraus resultiert, dass Pfade eines Teilsystems auch höchstens Acceleratoren an Pfade desselben Teilsystems senden.

Die hier beschriebenen qualitativen Eigenschaften und Zeitschranken der beiden KHS-Ansätze können als einfache Schlussfolgerungen aus den grundlegenden Eigenschaften in [BR09] gewonnen werden. Diese grundlegenden Eigenschaften wurden dort ausführlich evaluiert, weshalb dies hier nicht mehr benötigt wird.

In den folgenden beiden Unterkapiteln wird das Zeitverhalten der beiden Ansätze ermittelt. Die Angabe von WCET-Schranken zeigt weiterhin, dass beide Ansätze echtzeitfähig sind.

5.2.4.1. Zeitverhalten bei der Selbstkonfiguration

Satz 5.2.9. *Gegeben sei ein verteiltes System, welches das einstufige KHS zur Allokierung von Pfaden nutze. Eine obere Schranke für die maximale Konfigurationszeit ${}_1t_{K_{\max}}$ des einstufigen KHS ist gegeben durch:*

$$ {}_1t_{K_{\max}} = 2\sum_{i=1}^{q} k_i \ \textit{Hormonzyklen} \tag{5.8} $$

Beweis. Klar, da $\sum_{i=1}^{q} k_i$ Pfade verteilt werden sollen und nach Gleichung (5.2). □

Satz 5.2.10. *Gegeben sei ein verteiltes System, welches das zweistufige KHS zur Allokierung von T-Tasks und Pfaden nutze. Gelte für die obere Schranke u_i für die Übernahme einer T-Task T_i: $z_i \leq u_i \leq k_i$. Eine obere Schranke für die maximale Konfigurationszeit ${}_2t_{K_{\max}}$ des zweistufigen KHS ist gegeben durch:*

$$ {}_2t_{K_{\max}} = 2\sum_{i=1}^{q} u_i + 2\max_{i=1}^{q} k_i \ \textit{Hormonzyklen} \tag{5.9} $$

Beweis. ${}_2t_{K_{\max}}$ setzt sich aus der Konfigurationszeit für die T-Tasks und der Konfigurationszeit für die Pfade zusammen. Die Konfigurationszeit für die T-Tasks hängt davon ab, wieviele T-Tasks pro Teilsystem vergeben werden, nämlich u_i. Dies bedeutet, dass $\sum_{i=1}^{q} u_i$ T-Tasks in der ersten Stufe des zweistufigen KHS vergeben werden, und eine obere Schranke für die Konfigurationszeit somit $2\sum_{i=1}^{q} u_i$ Hormonzyklen beträgt nach Gleichung (5.2).

In der zweiten Stufe werden die Pfade vergeben. Allerdings ist hier zu beachten, dass die Pfade verschiedener Teilsystem zeitlich parallel vergeben werden. Daher beträgt die Konfigurationszeit wieder nach Gleichung (5.2) $2\max_{i=1}^{q} k_i$ Hormonzyklen.

Wenn man weiterhin davon ausgeht, dass die Konfiguration der Pfade erst dann beginnen kann, wenn alle T-Tasks vergeben sind, dann folgt die Behauptung für ${}_2t_{K_{\max}}$. □

Bemerkung 5.2.17. Die Schranke u_i für die Übernahme einer T-Task T_i ist also nach unten durch die Anzahl Pfade z_i des größten Zyklus im Teilsystem und nach oben durch die Anzahl Pfade k_i aller Pfade im Teilsystem beschränkt. Dabei ist die untere Schranke z_i eine notwendige Bedingung, um die Ausführung des Teilsystems gewährleisten zu können. Die obere Schranke ist nicht notwendig und kann genutzt werden, um nicht unnötig Ressourcen für Teilsystemausführungen zu allokieren.

Bemerkung 5.2.18. Ein direkter Vergleich der Konfigurationszeiten des einstufigen und zweistufigen KHS ist schwierig, da die Zeit, die für das zweistufige KHS benötigt wird, entscheidend von der Größe der verschiedenen u_i abhängt. So kann es bei kleinen Werten von u_i vorkommen, dass die Konfiguration des zweistufigen KHS schneller als beim einstufigen KHS ist, während bei großen Werten von u_i der umgekehrte Fall auftreten kann.

5.2.4.2. Zeitverhalten bei der Selbstheilung

Das verteilte System befinde sich in einem eingeschwungenen Zustand, das bedeutet, dass alle Pfade beziehungsweise T-Tasks und Pfade vergeben sind. Bei der Selbstheilung kann man nun verschiedene Fälle unterscheiden.

1. Die T-Tasks laufen stabil auf den Ressourcen und nur die Ausführung von Pfaden wird abgebrochen. Solange Pfade genau eines Teilsystems betroffen sind, gilt für die Selbstheilung diesselbe Zeitschranke wie in Gleichung (5.3) und zwar sowohl für das einstufige als auch für das zweistufige KHS.

   Wenn Pfade verschiedener Teilsysteme abgebrochen werden, dann sieht die Situation anders aus. Im Folgenden wird der extremste Fall beschrieben, nämlich dass gleichzeitig alle Pfade abgebrochen werden.

   Satz 5.2.11. *Gegeben sei ein verteiltes System, welches das einstufige beziehungsweise das zweistufige KHS zur Allokierung von Pfaden nutze. Es befinde sich in einem eingeschwungenen Zustand, und es werden zu einem Zeitpunkt gleichzeitig alle Pfade abgebrochen. Die Lebensdauer eines Hormons betrage a Hormonzyklen. Die maximale Heilungszeit ${}_1t_{H_{\max}}$ beträgt dann für das einstufige KHS ${}_1t_{H_{\max}} = 2\sum_{i=1}^{q} k_i + a$ Hormonzyklen.*
   Für das zweistufige KHS beträgt die maximale Heilungszeit ${}_2t_{H_{\max}} = 2\max_{i=1}^{q} k_i +$ a Hormonzyklen.

   Beweis. Beide Behauptungen sind klar nach Gleichung (5.3). Zusätzlich muss man beim zweistufigen KHS beachten, dass die Vergabe von Pfaden verschiedener Teilsysteme wieder zeitlich parallel läuft. □

   Bemerkung 5.2.19. In dem durch diesen Satz beschriebenen Fall dauert die Heilung im einstufigen KHS länger als im zweistufigen, und zwar offensichtlich

$2\sum_{i=1}^{q} k_i - 2\max_{i=1}^{q} k_i$ Hormonzyklen mehr. Man beachte dabei, dass diese Differenz natürlich größer oder gleich Null ist.

2. Auch die T-Tasks laufen nicht stabil auf den Ressourcen. Wenn eine Ressource eine T-Task verliert, so verliert sie auch aufgrund Konstruktion des zweistufigen KHS die zugehörigen Pfade. Im schlimmsten Fall verlieren also alle Ressourcen alle T-Tasks.

   Satz 5.2.12. *Gegeben sei ein verteiltes System, welches das einstufige beziehungsweise das zweistufige KHS zur Allokierung von Pfaden nutze. Es befinde sich in einem eingeschwungenen Zustand, und es werden zu einem Zeitpunkt gleichzeitig alle T-Tasks abgebrochen. Die Lebensdauer eines Hormons betrage a Hormonzyklen. Die maximale Heilungszeit ${}_1t_{H_{\max}}$ beträgt dann für das einstufige KHS ${}_1t_{H_{\max}} = 2\sum_{i=1}^{q} k_i + a$ Hormonzyklen.*

   Für das zweistufige KHS beträgt die maximale Heilungszeit ${}_2t_{H_{\max}} = 2\sum_{i=1}^{q} u_i + 2\max_{i=1}^{q} k_i + a + 1$ Hormonzyklen.

   Beweis. Wie im Beweis zu Satz 5.2.11 unter Beachtung der Zeitschranken aus Unterkapitel 5.2.4.1. □

5.2.4.3. Zeitverhalten bei der Terminierung

Da die Terminierung von Pfaden und T-Tasks zeitlich parallel abläuft sind hier die Zeitschranken aus den Unterkapiteln 5.2.1 und 5.2.2 gültig.

5.2.4.4. Datenaufkommen

Die Gleichungen aus Unterkapitel 5.1.3 lassen sich einfach an das einstufige beziehungsweise das zweistufige KHS anpassen, um das Hormondatenaufkommen pro Hormonzyklus zu ermitteln.

Einstufiges KHS:

Satz 5.2.13. *Gegeben sei ein verteiltes System, welches das einstufige KHS zur Allokierung von Pfaden nutze. Für die obere Schranke ${}_1D_{\max}^{Start}$ des Hormondatenaufkommens an einer Ressource pro Hormonzyklus zu Beginn der Pfadvergabe gilt:*

$$ {}_1D_{\max}^{Start} = \omega \sum_{i=1}^{q} k_i D^E \tag{5.10} $$

Beweis. Klar nach Gleichung (5.5) unter Beachtung, dass sich alle ω Ressourcen auf alle $\sum_{i=1}^{q} k_i$ Pfade bewerben. □

Satz 5.2.14. *Gegeben sei ein verteiltes System, welches das einstufige KHS zur Allokierung von Pfaden nutze. Für jeden Pfad gelte, dass er Acceleratoren höchstens an Pfade seines eigenen Teilsystems sende. Für die obere Schranke ${}_1D_{\max}^{Ende}$ des Hormondatenaufkommens an einer Ressource pro Hormonzyklus im eingeschwungenen Zustand gilt:*

$$ {}_1D_{\max}^{Ende} = 2\sum_{i=1}^{q} k_i D^S + \sum_{i=1}^{q} k_i \max_{i=1}^{q}\{k_i\} D^A \tag{5.11} $$

Beweis. Ergibt sich aus Gleichung (5.6). Zunächst werden die Suppressoren betrachtet: Es wurden alle $\sum_{i=1}^{k} k_i$ Pfade vergeben, daher werden für jeden pro Hormonzyklus zwei Suppressoren, ein „normaler“ und ein Terminationssuppressor, verschickt. Da auch klar ist, dass jeder Pfad zu jedem Zeitpunkt höchstens einmal vergeben ist, kann man den Faktor ω aus Gleichung (5.6) durch den Faktor 1 ersetzen.

Nun zu den Acceleratoren: Jeder der $\sum_{i=1}^{k} k_i$ Pfade verschickt Acceleratoren an die höchstens $\max_{i=1}^{k} k_i$ Pfade aus seinem Teilsystem. Wieder mit der Argumentation dass jeder Pfad zu jedem Zeitpunkt höchstens einmal vergeben ist, kann man den Faktor $\varphi_{\max}$ aus Gleichung (5.6) durch den Faktor 1 ersetzen. □

Zweistufiges KHS: Beim zweistufigen KHS kann man insgesamt vier Zustände unterscheiden und das maximale Datenaufkommen pro Hormonzyklus dafür berechnen:

1. Die erste Stufe läuft an. Das bedeutet, dass die Ressourcen beginnen Eignungswerte für die T-Tasks zu senden. Sei ${}_2D_{\max}^{11}$ eine obere Schranke für das Datenaufkommen an einer Ressource in diesem Zustand.

2. Die Ressourcen haben die T-Tasks vollständig vergeben und senden keine Eignungswerte mehr dafür, sondern nur noch Suppressoren und Acceleratoren. Die Vergabe der Pfade ist noch nicht gestartet. Sei ${}_2D_{\max}^{12}$ eine obere Schranke für das Datenaufkommen an einer Ressource in diesem Zustand.

3. Die zweite Stufe läuft an, das bedeutet, dass die Ressourcen beginnen Eignungswerte für die Pfade zu senden. Sei ${}_2D_{\max}^{21}$ eine obere Schranke für das Datenaufkommen an einer Ressource in diesem Zustand.

4. Die Ressourcen haben die Pfade vollständig vergeben und senden keine Eignungswerte mehr dafür, sondern nur noch Suppressoren und Acceleratoren. Sei ${}_2D_{\max}^{22}$ eine obere Schranke für das Datenaufkommen an einer Ressource in diesem Zustand.

Satz 5.2.15. *Gegeben sei ein verteiltes System, welches das zweistufige KHS zur Allokierung von T-Tasks und Pfaden nutze, und es gelten die Eigenschaften, die zu Beginn dieses Kapitels beschrieben wurden. Weiter gelte für die obere Schranke*

u_i für die Übernahme einer T-Task T_i: $z_i \leq u_i \leq k_i$, und Acceleratoren werden nur innerhalb von Teilsystemen versendet. Dann können die gerade beschriebenen oberen Schranken für das Datenaufkommen an einer Ressource pro Hormonzyklus wie folgt abgeschätzt werden:

$$ {}_2D^{11}_{\max} = \omega q D^E \tag{5.12}$$

$$ {}_2D^{12}_{\max} = 2\sum_{i=1}^{q} u_i D^S + \sum_{i=1}^{q} u_i D^A \tag{5.13}$$

$$ {}_2D^{21}_{\max} = {}_2D^{12}_{\max} + \omega \max_{i=1}^{q} k_i D^E \tag{5.14}$$

$$ {}_2D^{22}_{\max} = {}_2D^{12}_{\max} + 2\sum_{i=1}^{q} k_i D^S + \sum_{i=1}^{q} k_i \max_{i=1}^{q} k_i D^A \tag{5.15}$$

Beweis.
Zu ${}_2D^{11}_{\max}$: Klar nach Gleichung (5.5) unter Beachtung, dass sich alle ω Ressourcen auf alle q T-Tasks bewerben.

Zu ${}_2D^{12}_{\max}$: In der ersten Stufe werden maximal $\sum_{i=1}^{q} u_i$ T-Tasks allokiert. Für jede T-Task werden pro Hormonzyklus zwei Suppressoren ausgesendet. Dies wird durch den ersten Summanden beschrieben, analog zum ersten Summanden in Gleichung (5.6). Da eine T-Task vom Typ T_i Acceleratoren höchstens an sich selber sendet, resultiert der zweite Summand analog zu Gleichung Gleichung (5.6).

Zu ${}_2D^{21}_{\max}$: Der erste Summand resultiert aus dem Datenaufkommen pro Hormonzyklus für die T-Tasks im einschwungenen Zustand, da für diese ja weiterhin Suppressoren und Acceleratoren gesendet werden. Der zweite Summand folgt analog zu Gleichung (5.5), wenn man beachtet, dass eine Ressource sich höchstens für die Pfade ihres Teilsystems, und damit für höchstens $\max_{i=1}^{q} k_i$ Pfade, bewirbt.

Zu ${}_2D^{22}_{\max}$: Der erste Summand resultiert aus dem Datenaufkommen pro Hormonzyklus für die T-Tasks im einschwungenen Zustand, da für diese ja weiterhin Suppressoren und Acceleratoren gesendet werden. Für jeden der $\sum_{i=1}^{q} k_i$ Pfade werden pro Hormonzyklus zwei Suppressoren ausgesendet. Dies wird durch den zweiten Summanden beschrieben, analog zum ersten Summanden in Gleichung (5.6). Da ein Pfad Acceleratoren höchstens an Pfade seines Teilsystems sendet, resultiert der dritte Summand analog zu Gleichung Gleichung (5.6). □

Eine Gegenüberstellung von grundlegenden Eigenschaften des einstufigen und des zweistufigen KHS werden in der folgenden Tabelle 5.2 aufgezeigt. Sie ergeben sich direkt aus den Beschreibungen und Analysen ab Unterkapitel 5.2.

	Einstufiges KHS	Zweistufiges KHS
Selbstkonfigurierend	✓	✓
Selbstheilend	✓	✓
Echtzeitfähige Pfadallokation	✓	✓
Echtzeitfähige Allokation von T-Tasks	—	✓
Wechselwirkungen bei Pfadausführung verschiedener Teilsysteme	✓	X
Exklusive Ressourcennutzung	X	✓
Abschätzbares Hormondatenaufkommen	✓	✓

Tabelle 5.2.: Eigenschaften des einstufigen und des zweistufigen KHS (✓: besitzt Eigenschaft; X: besitzt Eigenschaft nicht; —: existiert nicht in dieser Ausprägung des KHS)

6. Evaluation der organischen Pfadverteilung

Im Folgenden werden verschiedene Benchmarks mit dem einstufigen und dem zweistufigen KHS evaluiert. Der Simulator läuft dabei auf einem Desktop-PC mit einer Intel Core2Quad CPU mit 2,83 GHz, 4 Gigabytes Hauptspeicher und einem 32 Bit Windows Vista Betriebssystem. Um die einzelnen Ressourcen zu simulieren wird für jede ein eigener Thread im Simulator eröffnet, um der Nebenläufigkeit im KHS Rechnung zu tragen. Für jede Ressource wird für die Überwachung der Ausführung von Pfaden, siehe Kapitel 5.2.3, ein weiterer Thread im Simulator eröffnet.

Die dargestellten Evaluationen fokussierten hierbei auf das Zeitverhalten der Pfadausführung. Die Modifikationen am KHS wie zum Beispiel dieBegrenzung der Übernahme von Pfaden und T-Tasks und deren Terminierung funktionierten immer reibungslos und werden daher nicht dargestellt.

6.1. Evaluation des ersten Benchmarks

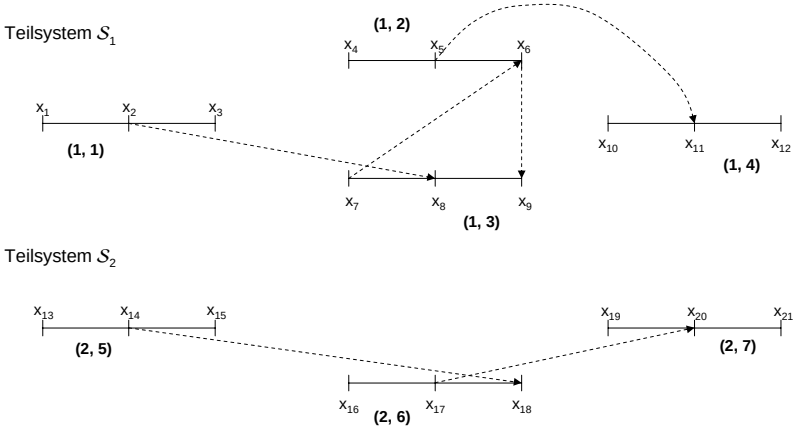


Abbildung 6.1.: Benchmarksystem (1)

Nun wird ein System $\mathcal{S}$ von Pfaden evaluiert, welches aus zwei unabhängigen Teilsystemen $\mathcal{S}_1$ und $\mathcal{S}_2$ gemäß Algorithmus 2 besteht. Das System ist in Abbildung 6.1 dargestellt. Die Pfade werden jeweils mit zwei Koordinaten gekennzeichnet, wobei

die erste Koordinate die Teilsystemnummer und die zweite Koordinate die Pfadnummer angibt. Der Pfad (1, 2) ist also der Pfad mit der Nummer 2 im Teilsystem $\mathcal{S}_1$. Die Aufträge sind durch ihre im ganzen System eindeutige Nummer gekennzeichnet. Da $\mathcal{S}$ insgesamt 21 Aufträge enthält wäre zur vollständigen Darstellung von $<_{\mathfrak{A}}$ zwischen den Aufträgen eine 21×21-Matrix notwendig. Da zwischen den Aufträgen der verschiedenen Teilsystem aber keine Beziehungen gelten, werden hier die Adjazenzmatrizen für $<_{\mathfrak{A}}$ in Einschränkung auf die jeweiligen Teilsysteme dargestellt. Zunächst wird $<_{\mathfrak{A}}$ auf den Aufträgen des Teilsystems $\mathcal{S}_1$ gezeigt:

$<_{\mathfrak{A}}$ auf $\mathcal{S}_1$	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
x_1	0	1	1	0	0	0	0	1	1	0	0	0
x_2	0	0	1	0	0	0	0	1	1	0	0	0
x_3	0	0	0	0	0	0	0	0	0	0	0	0
x_4	0	0	0	0	1	1	0	0	1	0	1	1
x_5	0	0	0	0	0	1	0	0	1	0	1	1
x_6	0	0	0	0	0	0	0	0	1	0	0	0
x_7	0	0	0	0	0	1	0	1	1	0	0	0
x_8	0	0	0	0	0	0	0	0	1	0	0	0
x_9	0	0	0	0	0	0	0	0	0	0	0	0
x_{10}	0	0	0	0	0	0	0	0	0	0	1	1
x_{11}	0	0	0	0	0	0	0	0	0	0	0	1
x_{12}	0	0	0	0	0	0	0	0	0	0	0	0

Die Adjazenzmatrix für $<_{\mathfrak{A}}$ auf den Aufträgen des Teilsystems $\mathcal{S}_2$ lautet:

$<_{\mathfrak{A}}$ auf $\mathcal{S}_2$	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}
x_{13}	0	1	1	0	0	1	0	0	0
x_{14}	0	0	1	0	0	1	0	0	0
x_{15}	0	0	0	0	0	0	0	0	0
x_{16}	0	0	0	0	1	1	0	1	1
x_{17}	0	0	0	0	0	1	0	1	1
x_{18}	0	0	0	0	0	0	0	0	0
x_{19}	0	0	0	0	0	0	0	1	1
x_{20}	0	0	0	0	0	0	0	0	1
x_{21}	0	0	0	0	0	0	0	0	0

Die gewählten Zeiten mit der Einheit Sekunden werden in der Tabelle 6.1 festgehalten.

Die beiden Teilpfadysteme werden in einem verteilten System von fünf Ressourcen evaluiert, wobei jede jeden Pfad ausführen kann. Dies ist eine ausreichende Anzahl von Ressourcen, da in Teilsystem $\mathcal{S}_1$ nur ein Zyklus von Pfaden bezüglich $<$ vorkommt: Die Pfade (1, 2) und (1, 3) liegen nämlich in solch einem Zyklus. Für die Ausführung von $\mathcal{S}_1$ werden also minimal zwei Ressourcen benötigt. Weiterhin können die Pfade (1, 1) und (1, 4) parallel zueinander ausgeführt werden. Die Relation

Ausführungs-zeit in sec	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}
	1	1	1	0,2	4	1	6	1	1	5	1	1,5	1
	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}					
	4	1	5	1	1	0,5	2	3					

Tabelle 6.1.: Ausführungszeiten der Aufträge

$<$ ist auf $\mathcal{S}_2$ irreflexiv, daher können alle Pfade dieses Systems nacheinander auf einer Ressource ausgeführt werden. Dies zeigt also, dass fünf Ressourcen für eine Ausführung der Pfade ausreichend sind.

Die Reihenfolge für die Ausführung der Pfade, die den Ressourcen durch Algorithmus 2 vorgegeben wird, lautet hier für das Teilsystem $\mathcal{S}_1$: (1, 1), (1, 2), (1, 3), (1, 4). Für das Teilsystem $\mathcal{S}_2$ gilt die Reihenfolge: (2, 5), (2, 6), (2, 7).

Für die Ausführung der Pfade wurde die In-Order-Strategie gewählt, was bedeutet, dass alle Pfade in der durch Algorithmus 2 vorgegebenen Reihenfolge ausgeführt werden müssen. Die Out-Of-Order-Strategie führte hier zu keinem anderen Resultat, da der Pfad (1, 1) in den Versuchen immer vor dem Pfad (1, 4) (aufgrund Konstruktion des KHS) allokiert wurde, wodurch eine Ausführung außerhalb der vorgegebenen Reihenfolge nicht stattfand.

6.1.1. Evaluation mit dem einstufigen KHS

Zunächst wird das Benchmarksystem mit dem einstufigen KHS evaluiert. Die Hormonwerte wurden dabei auf allen Ressourcen unter Einhaltung der in Kapitel 5 gegebenen Regeln etwa gleich groß gewählt. Des Weiteren wird bei jeder Übernahme eines Pfades ein Lastsuppressor mit geringer Stärke ausgesendet, um eine möglichst große Auslastung aller Ressourcen zu erreichen. Die Auswertung dieses Versuchs wird in Tabelle 6.2 für jedes Teilsystem, jeden Pfad und jeden Auftrag dargestellt. *Ausführungszeit* und *Latenzen* werden in Sekunden angegeben. *Anteil* gibt den Anteil der Ausführungszeit ohne Latenzen eines Pfades oder Auftrages an seiner Gesamtausführungszeit an. Durch *Ressource* wird die Nummer der ausführenden Ressource gekennzeichnet und in der Spalte *Fertig* ist gekennzeichnet, ob das Teilsystem, der Pfad oder der Auftrag auf der Ressource beendet wurde. Die Angaben über Latenzzeiten und deren Anteil an der Ausführungszeit machen für Teilsysteme aufgrund der parallelen Ausführung auf verschiedenen Ressourcen keinen Sinn und werden daher nicht aufgeführt.

Aus der Tabelle 6.2 kann man erkennen, dass alle Aufträge, Pfade und Teilsysteme vollständig ausgeführt wurden. Das einstufige KHS hat für die Pfade des Teilsystems 1 die Ressourcen 1, 2 und 3 allokiert, und für die Pfade des Teilsystems 2 wurden die Ressourcen 1, 4 und 5 allokiert. Weiterhin sieht man, dass die Ausführungszei-

	Ausführungszeit	Latenzen	Anteil	Ressource	Fertig
Teilsystem $\mathcal{S}_1$	17,5	-	-	1; 2; 3	✓
Pfad (1, 1)	3	0	1,0	1	✓
Pfad (1, 2)	16,2	11	0,321	2	✓
Pfad (1, 3)	8,5	0,5	0,94	1	✓
Pfad (1, 4)	7,5	0	1,0	3	✓
Auftrag x_1	1	0	1,0	1	✓
Auftrag x_2	1	0	1,0	1	✓
Auftrag x_3	1	0	1,0	1	✓
Auftrag x_4	0,2	0	1,0	2	✓
Auftrag x_5	4	0	1,0	2	✓
Auftrag x_6	12	11	0,083	2	✓
Auftrag x_7	6	0	1,0	1	✓
Auftrag x_8	1	0	1,0	1	✓
Auftrag x_9	1,5	0,5	0,666	1	✓
Auftrag x_{10}	5	0	1,0	3	✓
Auftrag x_{11}	1	0	1,0	3	✓
Auftrag x_{12}	1,5	0	1,0	3	✓
Teilsystem $\mathcal{S}_2$	11,5	-	-	1; 4; 5	✓
Pfad (2, 5)	6	0	1,0	1	✓
Pfad (2, 6)	7	0	1,0	5	✓
Pfad (2, 7)	11	5,5	0,5	4	✓
Auftrag x_{13}	1	0	1,0	1	✓
Auftrag x_{14}	4	0	1,0	1	✓
Auftrag x_{15}	1	0	1,0	1	✓
Auftrag x_{16}	5	0	1,0	5	✓
Auftrag x_{17}	1	0	1,0	5	✓
Auftrag x_{18}	1	0	1,0	5	✓
Auftrag x_{19}	0,5	0	1,0	4	✓
Auftrag x_{20}	7,5	5,5	0,266	4	✓
Auftrag x_{21}	3	0	1,0	4	✓
Gesamtausführungszeit	17,5				

Tabelle 6.2.: Ausführungsstatistik für das Benchmarksystem mit einstufigem KHS

ten der Aufträge bis auf kleine Abweichungen den beschriebenen Ausführungszeiten aus Tabelle 6.1 entsprechen. Die Abweichungen kommen einerseits durch den Berechungsaufwand im KHS zustande und andererseits werden sie durch Latenzzeiten verursacht. Bleibt also die Frage, wie diese Latenzzeiten zustande kommen: An der Statistik sieht man, dass die Ressource 1 für Pfade beider Teilsystems genutzt wurde. Hier kann es also zu gegenseitigen Beeinflussungen von Pfaden dieser Teilsysteme kommen. Diesen Effekt kann man im zeitlichen Verlauf der Ausführung des Systems in Abbildung 6.2 erkennen. Hier wird die Aktivität jeder der fünf Ressourcen durch eine Linie dargestellt. Die unterste Linie steht dabei für Ressource 1 und so weiter. Jede Linie kann drei verschiedene Niveaus haben: Wenn eine Linie auf dem niedrigsten Niveau ist, bedeutet das, dass die zugehörige Ressource gerade untätig ist und kein Pfad in der Bearbeitung ist. Eine Linie auf dem mittleren Niveau bedeutet, dass die zugehörige Ressource einen Pfad bearbeitet, aber in einem Wartezustand ist. Eine Linie auf dem höchsten Niveau bedeutet, dass die zugehörige Ressource gerade einen Auftrag ausführt. Die verschiedenen Aktionen einer Ressource sind dabei im Diagramm gekennzeichnet:

NoSuccSched: Die Ressource war nicht in der Lage, einen Pfad für die Ausführung zu finden, der nicht bereits ausgeführt ist.

PSCD: Die Ressource hat einen Pfad zur Ausführung ausgewählt.

JNR: Der als nächstes auszuführende Auftrag des Pfades ist nicht zur Ausführung bereit. Grund: Er wartet auf andere Aufträge.

JRDY: Der als nächstes auszuführende Auftrag des Pfades wird ausgeführt.

JEXE: Der aktuelle Auftrag wurde fertig ausgeführt.

PEXE: Der aktuelle Pfad wurde vollständig ausgeführt, das heißt alle seine Aufträge wurden ausgeführt.

PEXEABRT: Die Ausführung des aktuellen Pfades wurde abgebrochen, weil der Ressource ein weiterer Pfad zugewiesen wurde, der eine vollständige Ausführung des aktuellen Pfades unmöglich macht.

PEXEABRTN: Die Ausführung des aktuellen Pfades wurde abgebrochen, weil er der Ressource entzogen wurde.

IRPC: Der Pfadzähler innerhalb eines Teilsystems wurde um einen Pfad in der durch Algorithmus 2 gegebenen Reihenfolge hochgesetzt.

Zusätzlich wird zu jeder Aktion angegeben, auf welches Teilsystem (T-Task-Nummer), Pfad und Auftrag sie sich bezieht. Bei Aufträgen wurde der Übersichtlichkeit halber

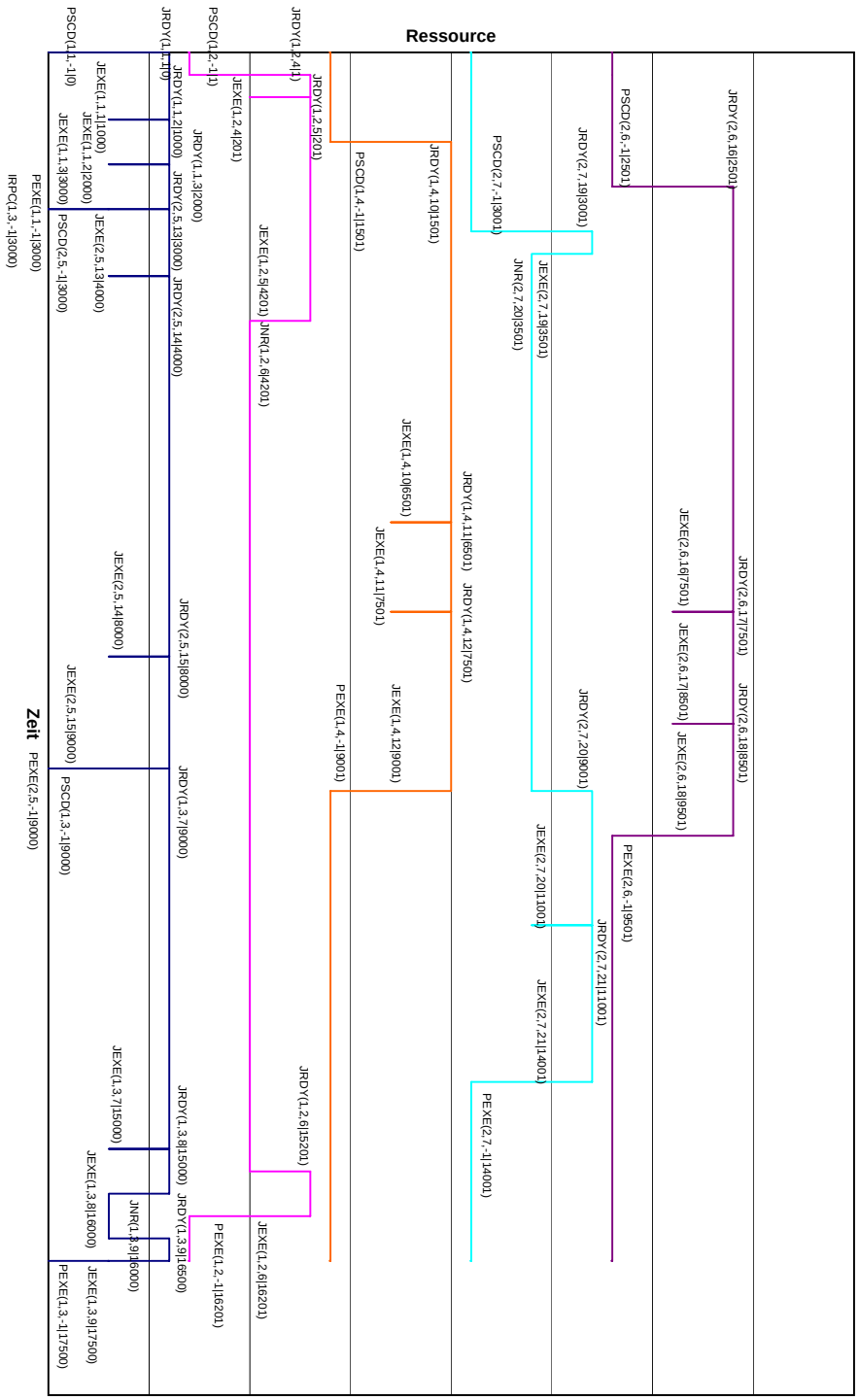


Abbildung 6.2.: Ausführung des Pfadsystems mit Benutzung des einstufigen KHS

auf nur die Nummer angegeben und auf das „x“ verzichtet. Zum Schluß der Kennzeichnung wird noch ein Zeitstempel in Millisekunden angegeben, der aus der Logdatei des Simulators ermittelt wird. Jeder Zeitstempel ist relativ zur global ersten Aktion zu sehen. Eine Darstellung in dieser nichtlinearen Form wurde der Übersichtlichkeit halber gewählt, da andernfalls nicht alle Aktionen in ein gemeinsamen Diagramm passen würden.

An der untersten Linie in Abbildung 6.2 sieht man, dass Ressource 1 die Pfade (1, 1), (2, 5) und (1, 3) in dieser Reihenfolge ausführt. Die Ausführung von Pfad (1, 1) wird sofort gestartet und läuft ohne Verzögerungen ab, was daran liegt, dass seine Aufträge nicht von Aufträgen anderer Pfade abhängig sind. Diesselbe Aussage gilt dann auch für die Ausführung von Pfad (2, 5). Dann, 9 Sekunden nach der ersten Aktion, wird die Ausführung von Pfad (1, 3) gestartet. Der Auftrag x_7 wird hierbei als erstes ausgeführt und benötigt fünf Sekunden. In der Zwischenzeit hat Ressource 2 (zweite Linie) mit der Ausführung von Pfad (1, 2) begonnen. Wenn sie aber mit der Ausführung von Auftrag x_6 beginnen will, kann sie das nicht, weil dieser Auftrag von der Ausführung des Auftrages x_7 abhängt. Deshalb wartet sie und im Diagramm erkennt man, dass sie 200 Millisekunden nach der Ausführung von x_7 mit der Ausführung von x_6 beginnt. Dies ist also eine gegenseitige Beeinflussung von Pfaden verschiedener Teilsysteme, der im zweistufigen KHS, siehe Unterkapitel 6.1.2, nicht mehr auftreten kann. Wenn man Ressource 1 weiter betrachtet, dann sieht man, dass sie nun im Gegenzug mit der Ausführung von Auftrag x_9 auf die Ausführung von Auftrag x_6 warten muss bevor sie die Ausführung des Pfades beenden kann. Ein ganz ähnlicher Effekt, diesmal aber zwischen Pfaden desselben Teilsystems, tritt auf den Ressourcen 4 und 5 bei der Ausführung der Aufträge x_{17} und x_{20} auf.

Aus den Werten in diesem Diagramm kann man auch ablesen, dass die rekursive Formel 4.1.1 beziehungsweise Lemma 4.1.2 gut passt. Sie bezieht lediglich die Zeit für das Senden der Information eines vollständig ausgeführten Auftrages nicht mit ein.

6.1.2. Evaluation mit dem zweistufigen KHS und minimalen Ressourcen

Nun wird das Benchmarksystem mit dem zweistufigen KHS evaluiert. Als zusätzliche Randbedingung wurden die Hormonwerte regelkonform so konfiguriert, dass für jedes Teilsystem höchstens die minimal benötigten Ressourcen durch T-Tasks reserviert werden, und damit auch nur die minimal benötigten Ressourcen für die Pfadausführung verwendet werden. Dies sind, wie oben beschrieben, zwei Ressourcen für das Teilsystem $\mathcal{S}_1$ und eine Ressource für das Teilsystem $\mathcal{S}_2$. Die Hormonwerte für T-Tasks wurden auf allen Ressourcen etwa gleich groß gewählt. Dasselbe gilt für die Hormonwerte der Pfade. Des Weiteren wird bei jeder Übernahme eines Pfades ein Lastsuppressor mit geringer Stärke ausgesendet, um eine möglichst große Auslas-

tung aller Ressourcen zu erreichen. Die Auswertung dieses Versuchs wird in Tabelle 6.3 für jedes Teilsystem, jeden Pfad und jeden Auftrag dargestellt.

Durch das zweistufige KHS wurden die Ressourcen 1 und 3 für die Ausführung des Teilsystems $\mathcal{S}_1$ und die Ressource 5 für die Ausführung des Teilsystems $\mathcal{S}_2$ gewählt, was der Randbedingung der Nutzung möglichst weniger Ressourcen entspricht. Man sieht, dass kaum Latenzen bei der Ausführung von Pfaden des Teilsystems $\mathcal{S}_1$ anfallen, was daran liegt, dass sich jeweils zwei Pfade eine Ressource teilen. Für die Pfade des Teilsystems $\mathcal{S}_2$ gilt, dass dort überhaupt keine Latenzzeiten anfallen. Dies liegt daran, dass sie alle auf genau einer Ressource in der vorgegebenen Reihenfolge ausgeführt werden, und somit gar keine Latenzen anfallen können.

Diese Beobachtungen sind auch gut aus dem Diagramm in Abbildung 6.3 ersichtlich.

Durch die stark sequentielle Ausführung beider Systeme wird dafür jedoch eine viel größere Ausführungszeit benötigt, als mit der Benutzung des einstufigen KHS in Unterkapitel 6.1.1. So benötigt die Ausführung von $\mathcal{S}_1$ mit dem einstufigen KHS 17,5 Sekunden, während sie unter Benutzung des zweistufigen KHS 17,7 Sekunden braucht. Viel deutlicher zeigt sich der Unterschied bei $\mathcal{S}_2$: Unter Benutzung des einstufigen KHS wird für seine Ausführung 11,5 Sekunden benötigt (es werden drei Ressourcen genutzt), während die Ausführung mit dem zweistufigen KHS 18,5 Sekunden braucht (es wird eine Ressource benötigt).

An diesem Versuch sieht man also, dass man durch den Einsatz des zweistufigen KHS die Latenzzeiten durch die gegenseitige Beeinflussung durch Pfade verschiedener Teilsysteme vermeiden kann. Allerdings scheint es sinnvoll, eine größere Anzahl an Ressourcen für die Ausführung zur Verfügung zu stellen, um die Ausführungsparallelität innerhalb von Teilsystem besser ausnutzen zu können.

6.1.3. Evaluation mit dem zweistufigen KHS

Nun wird das Benchmarksystem wieder mit dem zweistufigen KHS evaluiert. Allerdings wird hier die Randbedingung der minimalen Ressourcennutzung gelockert, und zwar können nun für das Teilsystem $\mathcal{S}_1$ maximal drei Ressourcen und für das Teilsystem maximal zwei Ressourcen allokiert werden. Die Werte für die verschiedenen Hormone auf den Ressourcen sind ansonsten gleich zu denen aus Unterkapitel 6.1.2.

Die Statistik zu diesem Versuch ist in Tabelle 6.4 zu sehen. Tatsächlich werden für das Teilsystem $\mathcal{S}_1$ drei Ressourcen allokiert, nämlich die Ressourcen 1, 2 und 3. Für Teilsystem $\mathcal{S}_2$ wurden durch das zweistufige KHS die Ressourcen 4 und 5 allokiert.

In der Statisik fällt auf, dass die gemessenen Werte der einzelnen Pfade und Aufträge identisch zu denen aus dem letzten Versuch sind, siehe Tabelle 6.3. Trotzdem ist die Ausführungszeit von Teilsystem $\mathcal{S}_1$ mit 11,5 Sekunden 6,2 Sekunden schneller als im letzten Versuch, während die Ausführungszeit von $\mathcal{S}_2$ ebenfalls mit 11,5 Sekunden ganze 7 Sekunden schneller als im letzten Versuch ist. Der Grund für diesen

	Ausführungszeit	Latenzen	Anteil	Ressource	Fertig
Teilsystem $\mathcal{S}_1$	17,7	-	-	1; 3	✓
Pfad (1, 1)	3	0	1,0	1	✓
Pfad (1, 2)	9,7	4,5	0,536	3	✓
Pfad (1, 3)	8,5	0,5	0,94	1	✓
Pfad (1, 4)	7,5	0	1,0	3	✓
Auftrag x_1	1	0	1,0	1	✓
Auftrag x_2	1	0	1,0	1	✓
Auftrag x_3	1	0	1,0	1	✓
Auftrag x_4	0,2	0	1,0	3	✓
Auftrag x_5	4	0	1,0	3	✓
Auftrag x_6	5,5	4,5	0,182	3	✓
Auftrag x_7	6	0	1,0	1	✓
Auftrag x_8	1	0	1,0	1	✓
Auftrag x_9	1,5	0,5	0,666	1	✓
Auftrag x_{10}	5	0	1,0	3	✓
Auftrag x_{11}	1	0	1,0	3	✓
Auftrag x_{12}	1,5	0	1,0	3	✓
Teilsystem $\mathcal{S}_2$	18,5	-	-	5	✓
Pfad (2, 5)	6	0	1,0	5	✓
Pfad (2, 6)	7	0	1,0	5	✓
Pfad (2, 7)	5,5	0	1,0	5	✓
Auftrag x_{13}	1	0	1,0	5	✓
Auftrag x_{14}	4	0	1,0	5	✓
Auftrag x_{15}	1	0	1,0	5	✓
Auftrag x_{16}	5	0	1,0	5	✓
Auftrag x_{17}	1	0	1,0	5	✓
Auftrag x_{18}	1	0	1,0	5	✓
Auftrag x_{19}	0,5	0	1,0	5	✓
Auftrag x_{20}	2	0	1,0	5	✓
Auftrag x_{21}	3	0	1,0	5	✓
Gesamtausführungszeit	18,5				

Tabelle 6.3.: Ausführungsstatistik für das Benchmarksystem mit zweistufigem KHS und minimalen Ressourcen

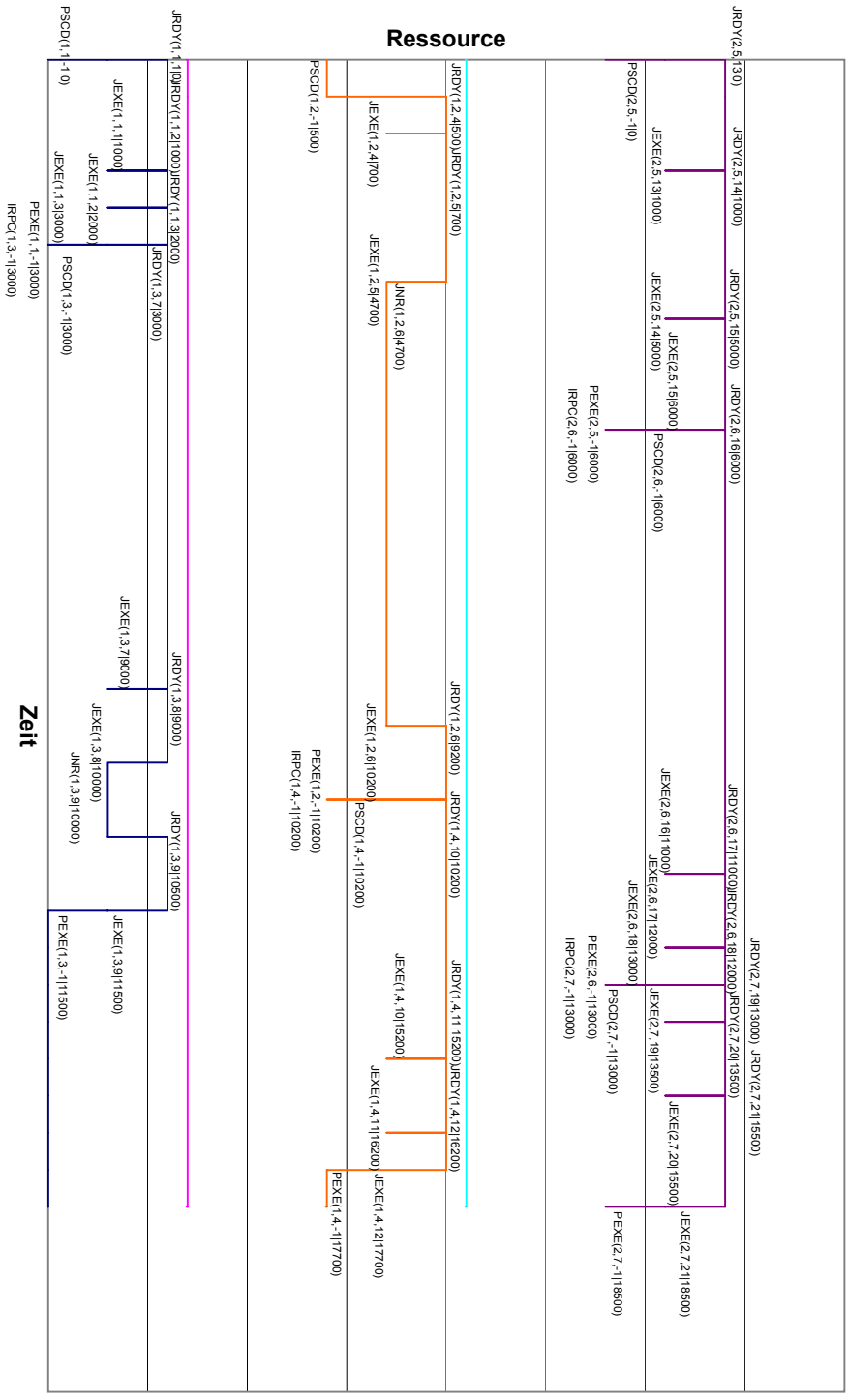


Abbildung 6.3.: Ausführung des Pfadsystems mit Benutzung des zweistufigen KHS und einer minimalen Anzahl an Ressourcen

Effekt liegt darin, dass nun pro Teilsystem je eine Ressource mehr für die Ausführung zur Verfügung steht, wie man im Diagramm in Abbildung 6.4 erkennen kann. Dadurch können mehr Pfade eines Teilsystems parallel ausgeführt werden.

In dieser Evaluation werden die Pfade (1, 1) und (1, 3) auf Ressource 3 ausgeführt, Pfad (1, 2) wird auf Ressource 2 ausgeführt, und Pfad (1, 4) wird auf Ressource 3 ausgeführt. Pfad (1, 1) wird vollständig ohne Latenzzeiten ausgeführt, und dann wird mit der Abarbeitung von Pfad (1, 3) begonnen. In der Zwischenzeit wurde die Abarbeitung von Pfad (1, 2) gestartet. Er kann bis zum Auftrag x_6 ausgeführt werden, muss dann aber auf die Ausführung von Auftrag x_7 von Pfad (1, 3) auf Ressource 1 warten. Weil damit aber Auftrag x_5 bereits ausgeführt wurde, kann Pfad (1, 4) auf Ressource 3 vollständig ohne Latenzzeiten ausgeführt werden. Die um 6,2 Sekunden schnellere Ausführung des Teilsystems in diesem Versuch gegenüber dem Versuch in Unterkapitel 6.1.2 kann also durch die parallele Abarbeitung von Pfad (1, 4) erklärt werden.

Die Ausführung des Teilsystems $\mathcal{S}_2$ ist ein gutes Beispiel für ein gutes Timing bei einer Pfadausführung. Pfad (2, 5) wird von Ressource 5 und Pfad (2, 6) wird von Ressource 4 ausgeführt. Die Ausführung von Auftrag x_{18} des Pfades (2, 6) kann erst nach der Ausführung des Auftrages x_{14} des Pfades (2, 5) geschehen. Bis die Ressource 4 aber mit der Ausführung des Pfades (2, 6) zu Auftrag x_{18} gelangt, ist Auftrag x_{14} auf Ressource 5 bereits ausgeführt, so dass hier keine Latenzzeiten entstehen.

Dasselbe Prinzip gilt auch für die Aufträge x_{20} und x_{21} von Pfad (2, 7), wieder auf Ressource 5, die auf die Ausführung von Auftrag x_{17} von Pfad (2, 6) warten. Bis Ressource 5 zur Ausführung von x_{20} und x_{21} gelangt, wurde x_{17} auf Ressource 4 bereits vollständig ausgeführt, so dass hier keine Latenzzeiten entstehen.

Am Start der Ausführung der Pfade von $\mathcal{S}_1$ kann man die sequentielle Arbeitsweise des KHS bei der Vergabe erkennen: Zunächst wird Pfad (1, 1) (0 Sekunden), dann Pfad (1, 2) (0,5 Sekunden), dann Pfad (1, 3) und zum Schluss Pfad (1, 4) (1,5 Sekunden) vergeben. Selbiges gilt für die Pfade des Teilsystems $\mathcal{S}_2$, nur fällt dieser Effekt wegen ihrer sequentiellen Ausführung nicht weiter auf.

Fazit: In diesem Versuch sieht man einen Vorteil des zweistufigen KHS gegenüber dem einstufigen KHS deutlich. Es gibt keine gegenseitige Beeinflussung von Pfaden verschiedener Systeme, und durch eine großzügige Ressourcenvorgabe kann die Ausführungszeit eines Teilsystems minimiert werden. Dahingegen ist das einstufige KHS flexibler bei der Ressourcenvergabe, da keine Vorgaben über die Anzahl der benötigten Ressourcen mitgegeben werden müssen.

	Ausführungszeit	Latenzen	Anteil	Ressource	Fertig
Teilsystem $\mathcal{S}_1$	11,5	-	-	1; 2; 3	✓
Pfad (1, 1)	3	0	1,0	1	✓
Pfad (1, 2)	9,7	4,5	0,536	2	✓
Pfad (1, 3)	8,5	0,5	0,94	1	✓
Pfad (1, 4)	7,5	0	1,0	3	✓
Auftrag x_1	1	0	1,0	1	✓
Auftrag x_2	1	0	1,0	1	✓
Auftrag x_3	1	0	1,0	1	✓
Auftrag x_4	0,2	0	1,0	2	✓
Auftrag x_5	4	0	1,0	2	✓
Auftrag x_6	5,5	4,5	0,182	2	✓
Auftrag x_7	6	0	1,0	1	✓
Auftrag x_8	1	0	1,0	1	✓
Auftrag x_9	1,5	0,5	0,666	1	✓
Auftrag x_{10}	5	0	1,0	3	✓
Auftrag x_{11}	1	0	1,0	3	✓
Auftrag x_{12}	1,5	0	1,0	3	✓
Teilsystem $\mathcal{S}_2$	11,5	-	-	4; 5	✓
Pfad (2, 5)	6	0	1,0	5	✓
Pfad (2, 6)	7	0	1,0	4	✓
Pfad (2, 7)	5,5	0	1,0	5	✓
Auftrag x_{13}	1	0	1,0	5	✓
Auftrag x_{14}	4	0	1,0	5	✓
Auftrag x_{15}	1	0	1,0	5	✓
Auftrag x_{16}	5	0	1,0	4	✓
Auftrag x_{17}	1	0	1,0	4	✓
Auftrag x_{18}	1	0	1,0	4	✓
Auftrag x_{19}	0,5	0	1,0	5	✓
Auftrag x_{20}	2	0	1,0	5	✓
Auftrag x_{21}	3	0	1,0	5	✓
Gesamtausführungszeit	11,5				

Tabelle 6.4.: Ausführungsstatistik für das Benchmarksystem mit zweistufigem KHS; drei Ressourcen für die Ausführung $\mathcal{S}_1$ und zwei Ressourcen für die Ausführung von $\mathcal{S}_2$

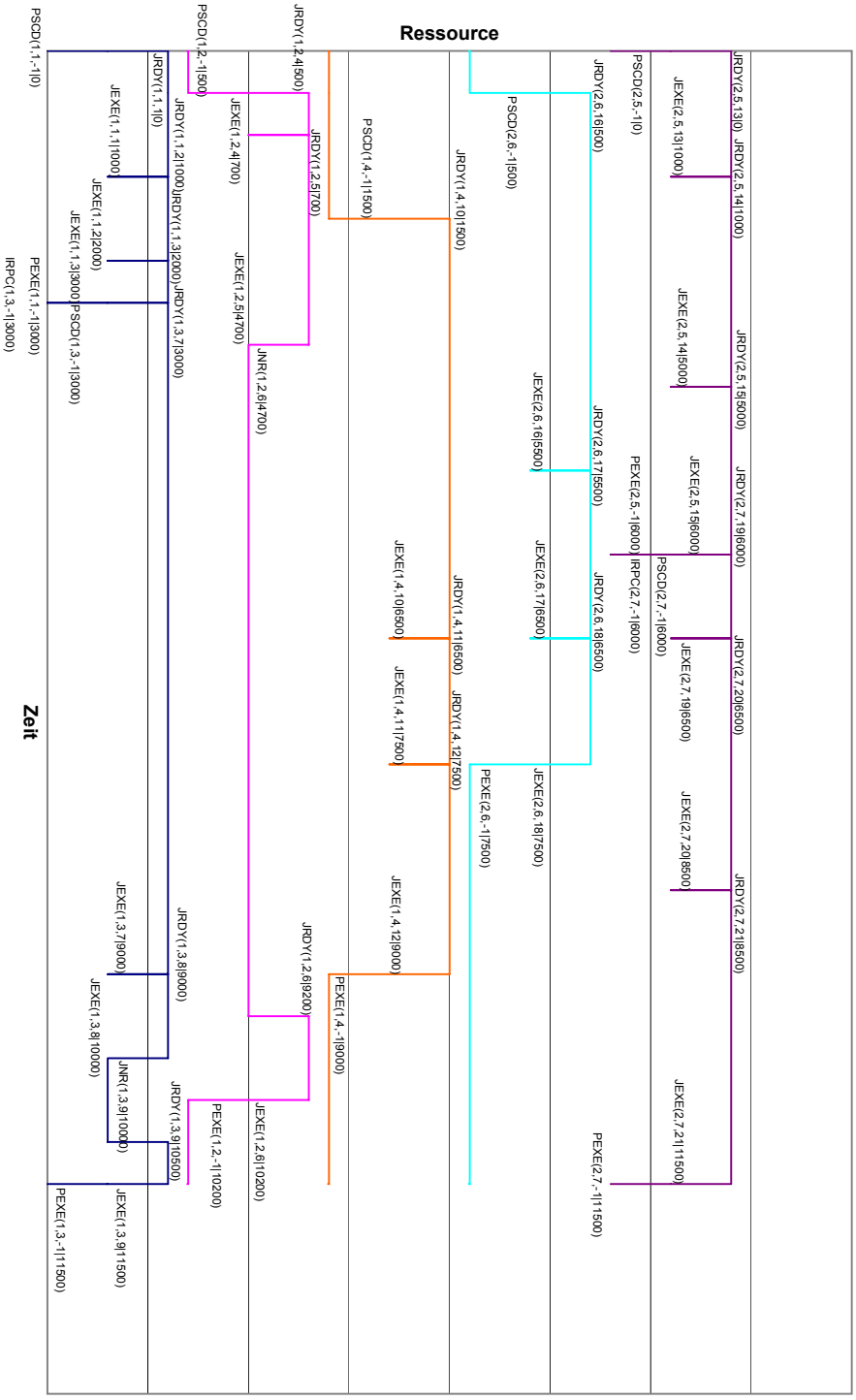


Abbildung 6.4.: Ausführung des Pfadsystems mit Benutzung des zweistufigen KHS und mehreren Ressourcen

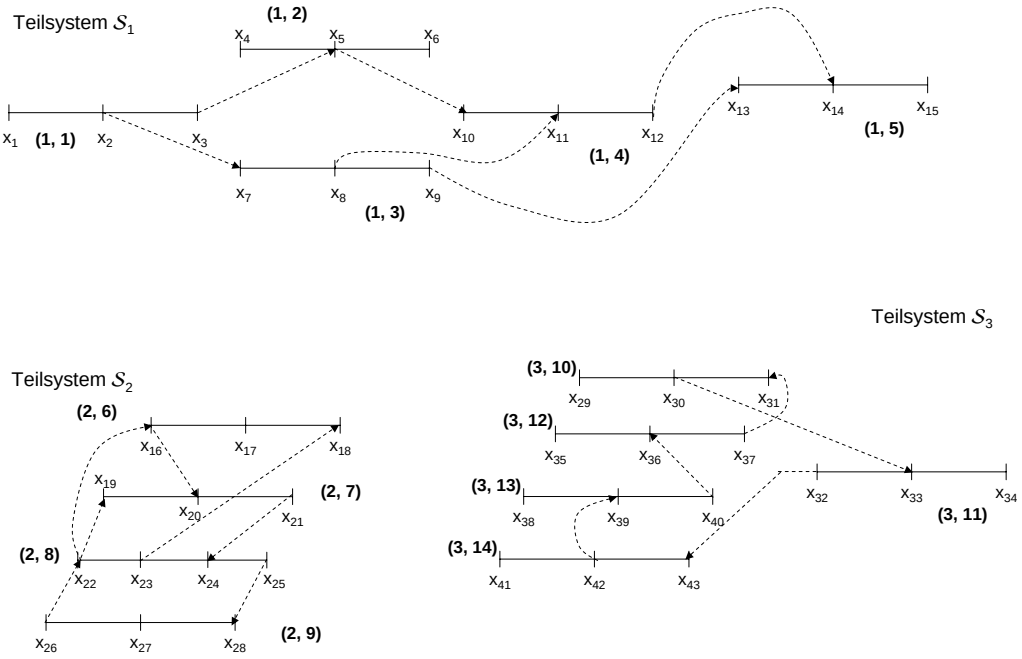


Abbildung 6.5.: Benchmarksystem (2)

6.2. Evaluation des zweiten Benchmarks

Nun wird ein System $\mathcal{S}$ von Pfaden evaluiert, welches aus drei unabhängigen Teilsystemen $\mathcal{S}_1$, $\mathcal{S}_2$ und $\mathcal{S}_3$ besteht. Das System ist in Abbildung 6.1 dargestellt. Die Pfade werden jeweils wieder mit zwei Koordinaten gekennzeichnet, wobei die erste Koordinate die Teilsystemnummer und die zweite Koordinate die Pfadnummer angibt. Da $\mathcal{S}$ insgesamt 43 Aufträge enthält wäre zur vollständigen Darstellung von $<_{\mathfrak{A}}$ zwischen den Aufträgen eine 43×43-Matrix notwendig. Da zwischen den Aufträgen der verschiedenen Teilsysteme aber keine Beziehungen gelten, werden hier die Adjazenzmatrizen für $<_{\mathfrak{A}}$ in Einschränkung auf die jeweiligen Teilsysteme dargestellt. Zunächst wird $<_{\mathfrak{A}}$ auf den Aufträgen des Teilsystems $\mathcal{S}_1$ gezeigt, siehe Tabelle 6.5. Die Adjazenzmatrix zu $<_{\mathfrak{A}}$ bezüglich der Aufträge des Teilsystems $\mathcal{S}_2$ wird in Tabelle 6.6 gezeigt, und die Adjazenzmatrix zu $<_{\mathfrak{A}}$ bezüglich der Aufträge des Teilsystems $\mathcal{S}_3$ wird in Tabelle 6.7 gezeigt. Die Ausführungsdauer der einzelnen Aufträge wird dann in Tabelle 6.8 aufgeführt.

Die drei Teilpfadysteme werden in einem verteilten System von zwölf Ressourcen evaluiert, wobei jede jeden Pfad ausführen kann.

Für das Teilsystem $\mathcal{S}_1$ gilt, dass es sequentiell auf einer Ressource ausführbar ist, da $<$ auf den Pfaden irreflexiv ist. Die durch Algorithmus 2 festgelegte Ordnung auf den Pfaden von $\mathcal{S}_1$ ist (1, 1), (1, 2), (1, 3), (1, 4) und (1, 5). Hierbei gilt aber, dass die beiden Pfade (1, 2) und (1, 3) parallel ausführbar sind, da sie nicht in $<$ (und damit

$<_{\mathfrak{A}}$ auf $\mathcal{S}_1$	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1
x_2	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1
x_3	0	0	0	0	1	1	0	0	0	1	1	1	0	1	1
x_4	0	0	0	0	1	1	0	0	0	1	1	1	0	1	1
x_5	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1
x_6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_7	0	0	0	0	0	0	0	1	1	0	1	1	1	1	1
x_8	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1
x_9	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
x_{10}	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1
x_{11}	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
x_{12}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
x_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
x_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
x_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabelle 6.5.: Relation $<_{\mathfrak{A}}$ auf dem Teilsystem $\mathcal{S}_1$

$<_{\mathfrak{A}}$ auf $\mathcal{S}_2$	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	x_{27}	x_{28}
x_{16}	0	1	1	0	1	1	0	0	1	1	0	0	1
x_{17}	0	0	1	0	0	0	0	0	0	0	0	0	0
x_{18}	0	0	0	0	0	0	0	0	0	0	0	0	0
x_{19}	0	0	0	0	1	1	0	0	1	1	0	0	1
x_{20}	0	0	0	0	0	1	0	0	1	1	0	0	1
x_{21}	0	0	0	0	0	0	0	0	1	1	0	0	1
x_{22}	1	1	1	1	1	1	0	1	1	1	0	0	1
x_{23}	0	0	1	0	0	0	0	0	1	1	0	0	1
x_{24}	0	0	0	0	0	0	0	0	0	1	0	0	1
x_{25}	0	0	0	0	0	0	0	0	0	0	0	0	1
x_{26}	1	1	1	1	1	1	1	1	1	1	0	1	1
x_{27}	0	0	0	0	0	0	0	0	0	0	0	0	1
x_{28}	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabelle 6.6.: Relation $<_{\mathfrak{A}}$ auf dem Teilsystem $\mathcal{S}_2$

$<_{\mathfrak{A}}$ auf $\mathcal{S}_3$	x_{29}	x_{30}	x_{31}	x_{32}	x_{33}	x_{34}	x_{35}	x_{36}	x_{37}	x_{38}	x_{39}	x_{40}	x_{41}	x_{42}	x_{43}
x_{29}	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0
x_{30}	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0
x_{31}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_{32}	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1
x_{33}	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
x_{34}	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
x_{35}	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0
x_{36}	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
x_{37}	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
x_{38}	0	0	1	0	0	0	0	1	1	0	1	1	0	0	0
x_{39}	0	0	1	0	0	0	0	1	1	0	0	1	0	0	0
x_{40}	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0
x_{41}	0	0	1	0	0	0	0	1	1	0	1	1	0	1	1
x_{42}	0	0	1	0	0	0	0	1	1	0	1	1	0	0	1
x_{43}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabelle 6.7.: Relation $<_{\mathfrak{A}}$ auf dem Teilsystem $\mathcal{S}_3$

Ausführungs-zeit in sec	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}
	1	1	0,5	0,2	4	1	6	1	2	5	1	1,5	2
	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}
	1,5	3	5	1	1	0,5	2	3	4	3,5	1,5	1	2
	x_{27}	x_{28}	x_{29}	x_{30}	x_{31}	x_{32}	x_{33}	x_{34}	x_{35}	x_{36}	x_{37}	x_{38}	x_{39}
	1,5	2	4	3,5	1	2,5	3	2	2	2	6	2	3,5
	x_{40}	x_{41}	x_{42}	x_{43}									
	1,5	2	2,5	0,2									

Tabelle 6.8.: Ausführungszeiten der Aufträge

auch nicht in $<_D$) in Relation stehen. Hier ist also Potential zur Parallelisierung vorhanden.

Im Teilsystem $\mathcal{S}_2$ liegen alle vier Pfade in einem einzigen Zyklus bezüglich $<$. Sie müssen jeweils auf einer exklusiven Ressource ausgeführt werden. Deshalb macht eine Angabe der Ausführungsreihenfolge hier keinen Sinn.

Die fünf Pfade des Teilsystems $\mathcal{S}_3$ liegen auch alle in einem einzigen Zyklus bezüglich $<$. Hier kann allerdings eine Optimierung gemäß Satz 3.2.17 vorgenommen werden (in Übereinstimmung mit Satz 3.2.8): So sollen die Pfade in der Ordnung (3, 14), (3, 13), (3, 12) und (3, 10) auf einer Ressource ausgeführt werden, während der Pfad (3, 11) exklusiv auf einer anderen Ressource ausgeführt werden soll. Auf diese Weise kann man die Zahl der benötigten Ressourcen von fünf auf zwei herabsetzen, obwohl man dadurch die Vorteile der parallelen Ausführung nicht mehr voll nutzen kann.

Diese Betrachtung zeigt, dass die Anzahl der vorhanden Ressourcen ausreicht, um alle Pfade auszuführen. Die Hormonwerte sind wieder so gewählt, dass jede Ressource jeden Pfad ausführen kann. Bei der Übernahme eines Pfades wird jedoch ein Lastsuppressor ausgesendet, so dass eine Ressource nicht beliebig viele Pfade übernehmen kann.

In den Tabellen 6.9 und 6.10, die eine Statisik über die Ausführung der Teilsysteme dieses Benchmarksystems enthalten, werden die Angaben zu den Aufträgen aus Platzgründen nicht dargestellt. Die Details kann man sich aber einfach aus den Angaben über die zugehörigen Pfade und den Diagrammen in den Abbildungen 6.6 bzw. 6.7 und 6.8 bzw. 6.9klar machen.

6.2.1. Evaluation mit dem einstufigen KHS

Das Benchmarksystem wird zunächst mit dem einstufigen KHS mit einer In-Order-Ausführung evaluiert. Die Resultate werden in Tabelle 6.9 und in den Abbildungen 6.6 und 6.7 dargestellt. Die Aufteilung der Abbildung auf zwei Seiten wurde der Übersichtlichkeit halber vorgenommen. Es fällt auf, dass alle Ressourcen ausgenutzt werden. Die Pfade des Teilsystems $\mathcal{S}_1$ werden auf den Ressourcen 1, 5, 7 und 9, die Pfade des Teilsystems $\mathcal{S}_2$ auf den Ressourcen 2, 3, 6 und 10 und die Pfade des Teilsystems $\mathcal{S}_3$ auf den Ressourcen 1,4, 8, 11 und 12 ausgeführt. Dies bedeutet, dass nur die Ressource 1 für Pfade verschiedener Teilsysteme genutzt wird, die anderen Ressourcen stehen für ihre Pfade exklusiv zur Verfügung. Die Ausführung des Teilsystems $\mathcal{S}_1$ läuft in 23,5 Sekunden ab. Es sind dabei keine Latenzzeiten durch Pfade anderer Teilsysteme zu verzeichnen, weil der Pfad (1, 1) auf der Ressource 1 als erstes ausgeführt wird. Der hohe Grad der Parallelisierung (4 Ressourcen) kann hier wegen der stark sequentiellen Struktur der Pfade von $\mathcal{S}_1$ nicht voll ausgenutzt werden, was man insbesondere an den starken Latenzzeiten bei der Ausführung der Pfade (1, 4) (Ausführungszeit: 16 Sekunden, Latenzzeit: 8,5 Sekunden) und (1, 5) (Ausführungszeit: 21,5 Sekunden, Latenzzeit: 15 Sekunden) sieht, vergleiche Tabelle

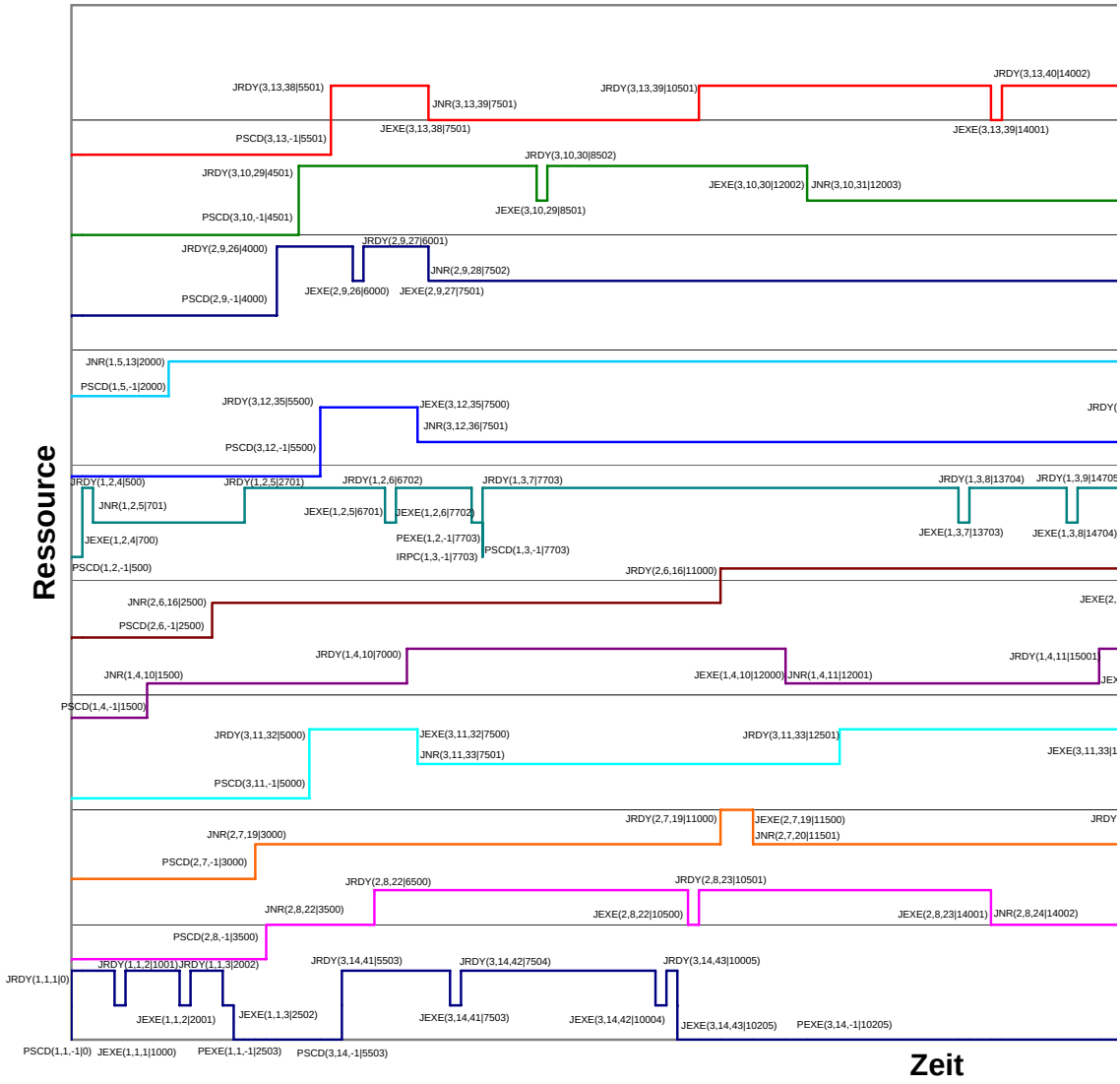


Abbildung 6.6.: Ausführung mit einstufigem KHS - Teil 1

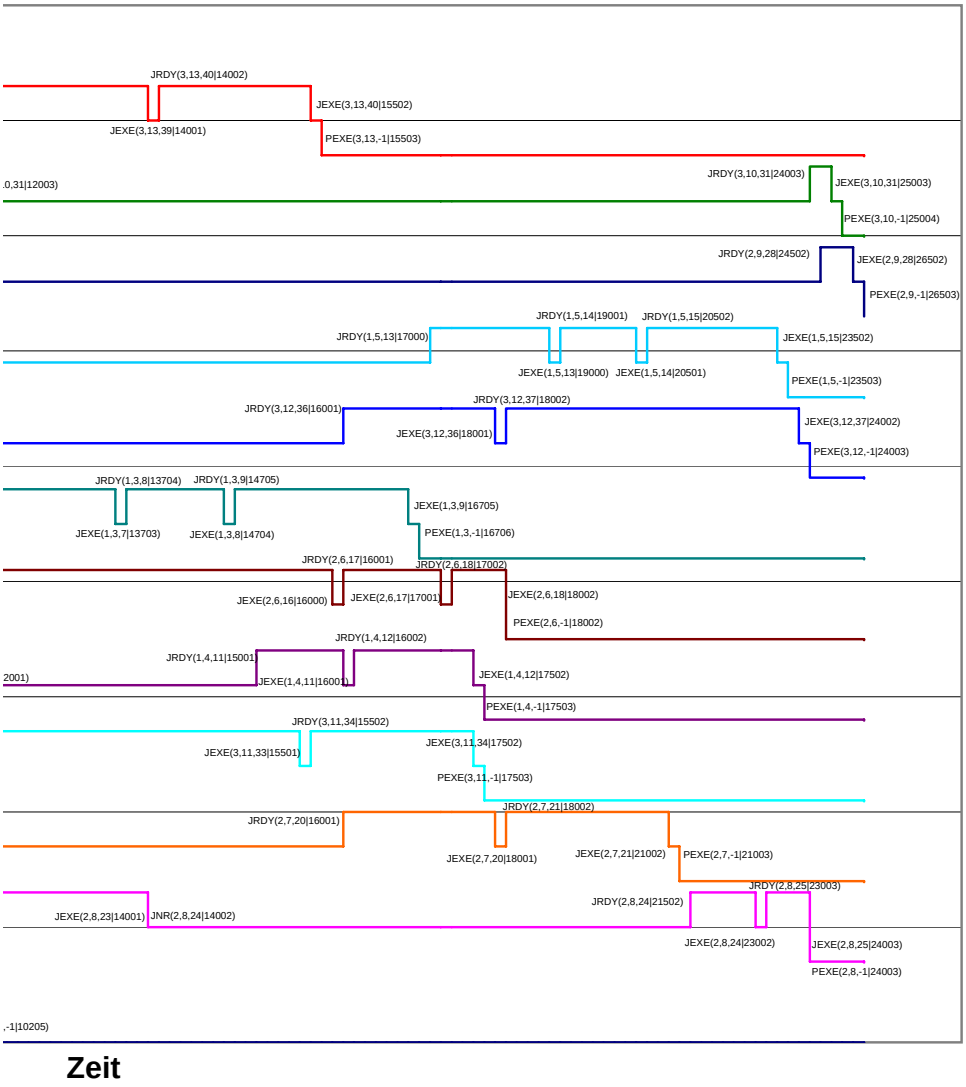


Abbildung 6.7.: Ausführung mit einstufigem KHS - Teil 2

	Ausführungszeit	Latenzen	Anteil	Ressource	Fertig
Teilsystem $\mathcal{S}_1$	23,5	-	-	1; 5; 7; 9	✓
Pfad (1, 1)	2,5	0	1,0	1	✓
Pfad (1, 2)	7,2	2	0,722	7	✓
Pfad (1, 3)	9	0	1,0	7	✓
Pfad (1, 4)	16	8,5	0,469	5	✓
Pfad (1, 5)	21,5	15	0,302	9	✓
Teilsystem $\mathcal{S}_2$	24	-	-	2; 3; 6; 10	✓
Pfad (2, 6)	15,5	8,5	0,452	6	✓
Pfad (2, 7)	18	12,5	0,306	3	✓
Pfad (2, 8)	20,5	10,5	0,488	2	✓
Pfad (2, 9)	22,5	17	0,245	10	✓
Teilsystem $\mathcal{S}_3$	20,5	-	-	1; 4; 8; 11; 12	✓
Pfad (3, 10)	20,5	12	0,415	11	✓
Pfad (3, 11)	12,5	5	0,6	4	✓
Pfad (3, 12)	18,5	8,5	0,541	8	✓
Pfad (3, 13)	10	3	0,7	12	✓
Pfad (3, 14)	4,7	0	1,0	1	✓
Gesamtausführungszeit	26,5				

Tabelle 6.9.: Ausführungsstatistik für das Benchmarksystem mit einstufigem KHS

6.9.

Die Ausführung des Teilsystems $\mathcal{S}_2$ läuft in 24 Sekunden ab, auf Ressourcen, die exklusiv für Pfade dieses Systems zur Verfügung stehen. Durch die Struktur der Relationen $<_{\mathfrak{A}}$ und $<$ ist hier keine Optimierung möglich, so dass jeder Pfad tatsächlich eine eigene Ressource zur Ausführung benötigt. Trotzdem treten aufgrund der starken Abhängigkeiten hohe Latenzzeiten auf, siehe Tabelle 6.9, die aber wegen dem eben Gesagten nicht verkleinerbar sind.

Das Teilsystem $\mathcal{S}_3$ wird innerhalb von 20,5 Sekunden auf fünf Ressourcen ausgeführt. Die Möglichkeit, es stark sequentiell auf nur zwei Ressourcen ausführen zu lassen, wird dabei durch das KHS nicht genutzt. Die auftretenden Latenzzeiten, siehe Tabelle 6.9, liegen ebenfalls an den starken Abhängigkeiten zwischen den Aufträgen der Pfade. Die Ausführung dieses Teilsystems wird sogar geringfügig verzögert, da Ressource 1 zunächst einen Pfad von $\mathcal{S}_1$ ausführt. Dies fällt bei der Gesamtausführungszeit jedoch nicht stark ins Gewicht, da diese durch die Ausführungszeit von $\mathcal{S}_2$ bestimmt wird.

Auch in dieser Evaluation kann man die Arbeitsweise des KHS gut erkennen. Je größer die Nummer eines Pfades ist, umso später startet seine Ausführung, vergleiche

dazu die Pfade (1, 1) (Startzeit: 0 Sekunden) und (3, 5) (Startzeit: 5,5 Sekunden). Daran sieht man, dass die Pfade sequentiell auf den Ressourcen durch das KHS vergeben werden. Dieser Effekt tritt bei diesem Benchmark unter Benutzung des zweistufigen KHS, siehe Unterkapitel 6.2.2, wesentlich schwächer auf, da hier die Pfade verschiedener Teilsysteme zeitlich parallel vergeben werden.

Weiterhin fällt auf, dass die Gesamtausführungszeit laut Tabelle 6.9 mit 26,5 Sekunden länger ist als die Gesamtausführungszeit der einzelnen Teilsysteme. Der Grund dafür liegt genau an der im letzten Absatz beschriebenen sequentiellen Verteilung der Pfade durch das KHS, wodurch die Ausführung der Teilsysteme mit einer geringen zeitlichen Versetzung relativ zueinander erfolgt. Dies liegt an den Berechnungszeiten des KHS für die Vergabe der T-Tasks und Pfade, die sich schließlich auf etwa $2,5$ Sekunden kumulieren. Diese Berechnungszeiten hängen ausschließlich vom KHS und nicht von der Anwendung ab. Die Ausführungszeiten der Pfade in dieser Evaluation wurden im Sekundenbereich gewählt, dass die Berechnungszeiten auffallen. Wenn die Ausführungszeiten länger wären, so würden die Berechungszeiten nicht oder kaum ins Gewicht fallen.

Eine Out-Of-Order Evaluation bringt bei der großen Ressourcenvielfalt keine anderen Ergebnisse, da Pfad (1, 2) immer zuerst allokiert wird und eine spätere Allokation von Pfad (1, 3) nicht zu seiner Unterbrechung und damit zu keinem erneuten Scheduling führt.

6.2.2. Evaluation mit dem zweistufigen KHS

Nun wird das Benchmarksystem wieder mit dem zweistufigen KHS und einer In-Order-Ausführung evaluiert. Hier wird das KHS von vornherein so konfiguriert, dass jedem Teilsystem in der ersten Stufe der Zuweisung höchstens vier Ressourcen exklusiv zugewiesen werden. Die Resultate werden in Tabelle 6.10 und in den Abbildungen 6.8 und 6.9 dargestellt. Die Aufteilung der Abbildung auf zwei Seiten wurde der wieder Übersichtlichkeit halber vorgenommen.

Die Pfade des Teilsystems $\mathcal{S}_1$ werden auf den Ressourcen 2, 7, 8 und 9, die Pfade des Teilsystems $\mathcal{S}_2$ auf den Ressourcen 1, 6, 10 und 11 und die Pfade des Teilsystems $\mathcal{S}_3$ auf den Ressourcen 3, 4, 5, 11 und 12 ausgeführt. Die Ausführung der Teilsysteme $\mathcal{S}_1$ und $\mathcal{S}_2$ läuft vom Timing her genauso ab wie in der Evaluation zum einstufigen KHS. Einzig die Ausführung von $\mathcal{S}_3$ ist hier anders: Zunächst wird der Pfad (3, 10) auf der Ressource 5 allokiert und seine Ausführung startet. Dann wird aber der Pfad (3, 14) ebenfalls auf dieser Ressource allokiert. Dieser muss aber nach der oben beschriebenen Ordnung vor dem Pfad (3, 10) ausgeführt werden. Deshalb wird die Ausführung von Pfad (3, 10) abgebrochen und ein neues Schedule durchgeführt, bei dem Pfad (3, 14) zur Ausführung bestimmt wird. Nach dessen Ausführung wird dann der Pfad (3, 10) vollständig ausgeführt. Dadurch wird die Ausführungszeit des Teilsystems auf 23,7 Sekunden verlängert.

Die Gesamtausführungszeit aller Teilsysteme liegt in dieser Evaluation bei 24 Se-

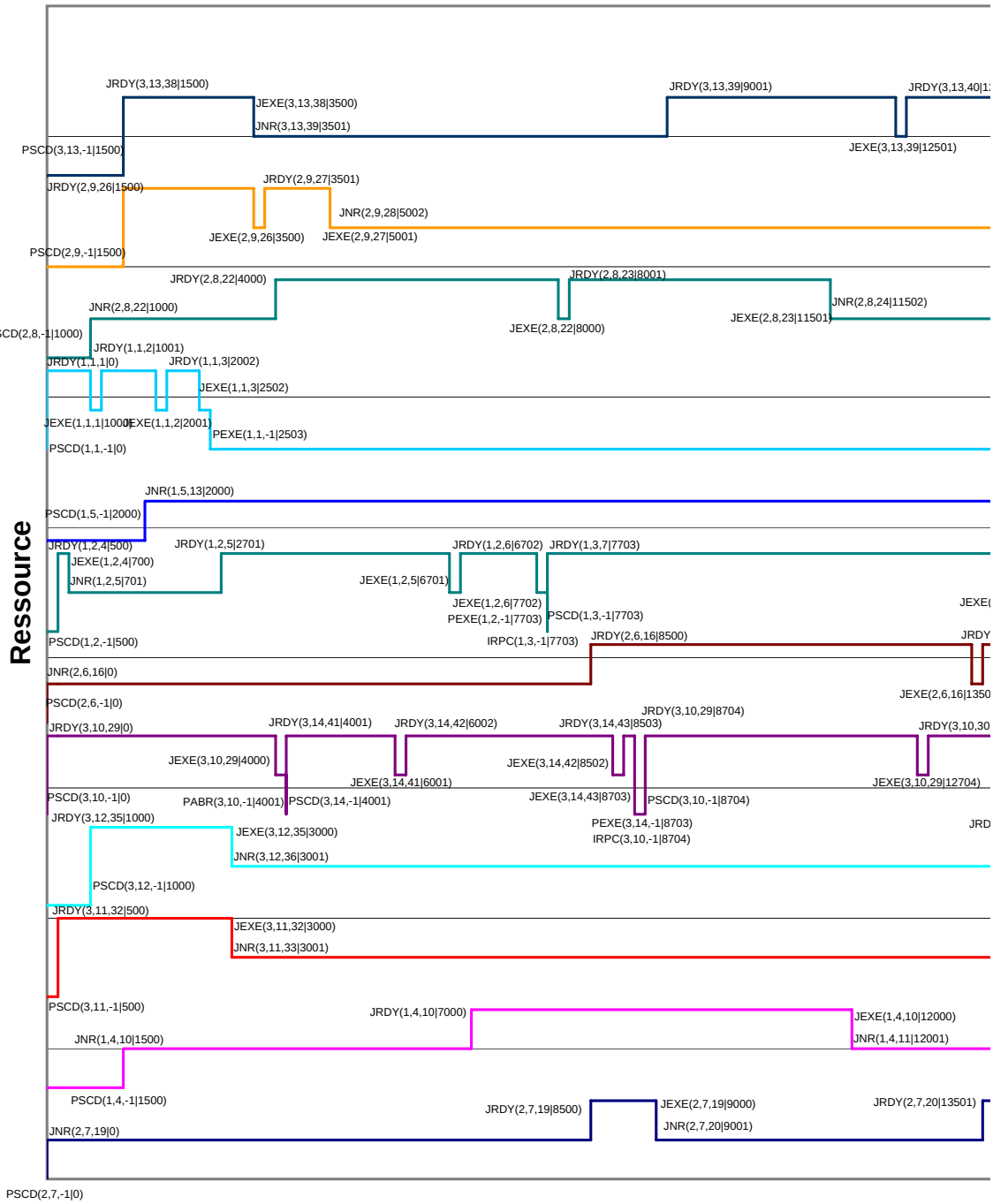


Abbildung 6.8.: Ausführung mit zweistufigem KHS - Teil 1

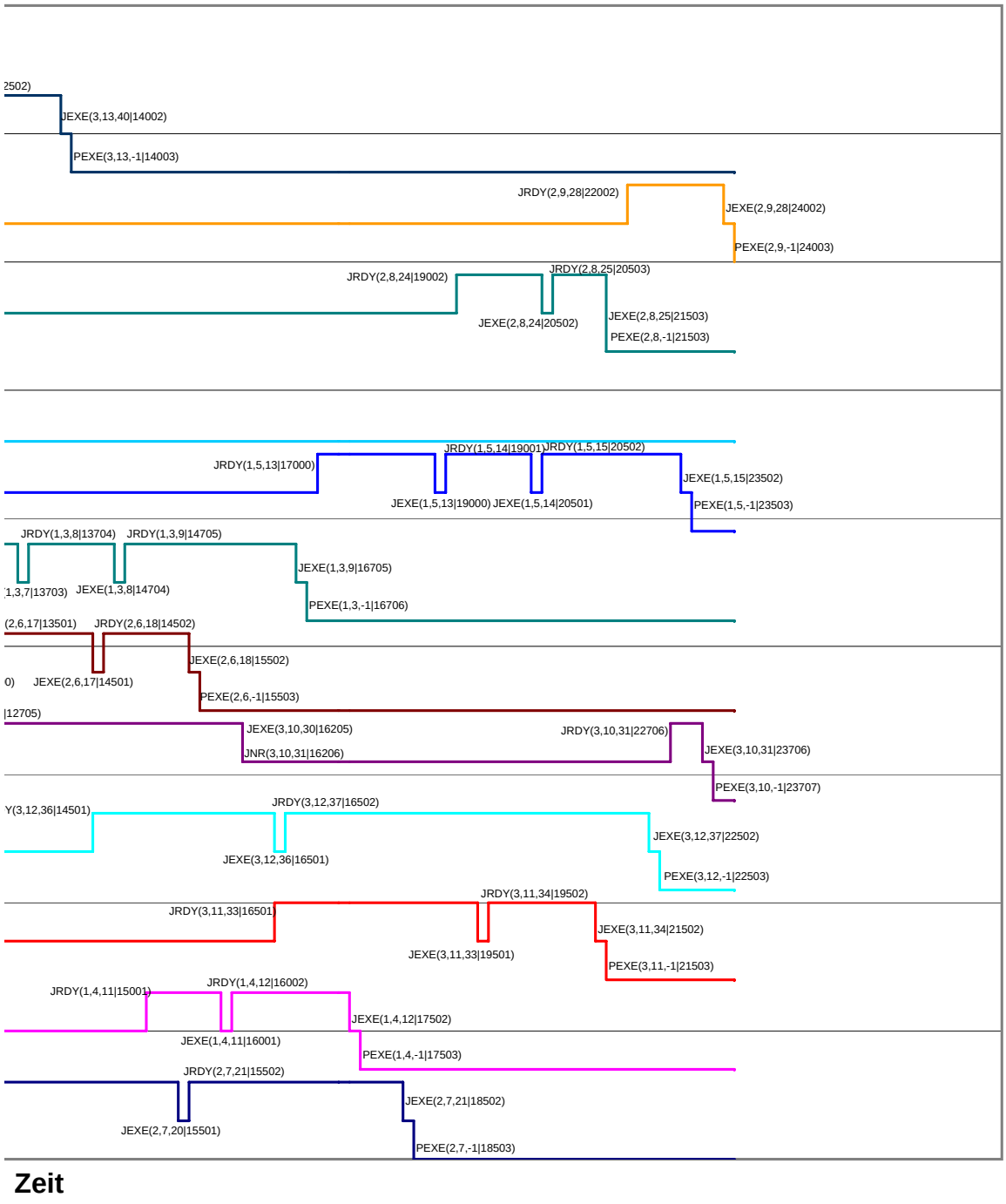


Abbildung 6.9.: Ausführung mit zweistufigem KHS - Teil 2

	Ausführungszeit	Latenzen	Anteil	Ressource	Fertig
Teilsystem $\mathcal{S}_1$	23,5	-	-	2; 7; 8; 9	✓
Pfad (1, 1)	2,5	0	1,0	9	✓
Pfad (1, 2)	7,2	2	0,722	7	✓
Pfad (1, 3)	9	0	1,0	7	✓
Pfad (1, 4)	16	8,5	0,469	2	✓
Pfad (1, 5)	21,5	15	0,302	8	✓
Teilsystem $\mathcal{S}_2$	24	-	-	1; 6; 10; 11	✓
Pfad (2, 6)	15,5	8,5	0,452	6	✓
Pfad (2, 7)	18	13	0,297	1	✓
Pfad (2, 8)	20,5	10,5	0,488	10	✓
Pfad (2, 9)	22,5	17	0,245	11	✓
Teilsystem $\mathcal{S}_3$	23,7	-	-	3; 4; 5; 12	✓
Pfad (3, 10)	4	0	1,0	5	–
Pfad (3, 10)	15	6,5	0,567	5	✓
Pfad (3, 11)	21	13,5	0,357	3	✓
Pfad (3, 12)	21,5	11,5	0,461	4	✓
Pfad (3, 13)	12,5	5,5	0,56	12	✓
Pfad (3, 14)	4,7	0	1,0	5	✓
Gesamtausführungszeit	24				

Tabelle 6.10.: Ausführungsstatistik für das Benchmarksystem mit zweistufigem KHS

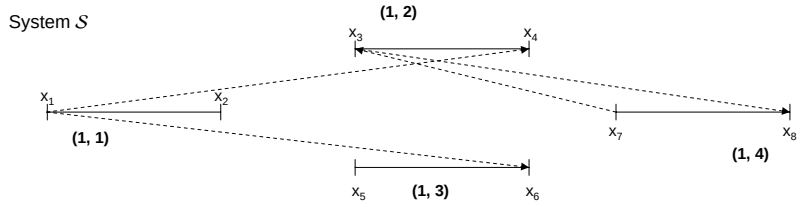


Abbildung 6.10.: Benchmarksystem (3)

kunden. Dieser Wert ist immer noch geringer als in der vorigen Evaluation, was erstens daran liegt, dass durch das zweistufige KHS die Verteilung der Pfade verschiedener Teilsystem zeitlich parallel erfolgt, und somit die Startzeiten der Pfade enger beieinander liegen können. Zweitens fällt der Abbruch von Pfad (3, 10) aufgrund der längeren Ausführungszeit des Teilsystems $\mathcal{S}_2$ nicht ins Gewicht.

Auch hier bringt eine Out-Of-Order Evaluation keine anderen Ergebnisse, da Pfad (1, 2) immer zuerst allokiert wird und eine spätere Allokation von Pfad (1, 3) nicht zu seiner Unterbrechung und damit zu keinem erneuten Scheduling führt.

Fazit: An dieser Evaluation erkennt man die Unterschiede der beiden Arten des KHS in der Zeitdauer der Allokation der Pfade. Wenn man die Zeitdauer für die Allokation der T-Tasks jedoch hinzurechnet, kann sich dieser zeitliche Unterschied wieder relativieren, siehe Kapitel 8.

Weiterhin sieht man, dass die Eigenschaft des zweistufigen KHS der Allokation von exklusiven Ressourcen bei einer hinreichend großen Anzahl an vorhandenen Ressourcen auch durch das einstufige KHS realisiert wird. Unter solchen Randbedingungen liegt der Schluß nahe, das einstufige KHS zu verwenden, schon um einem Anwender die Konfiguration der T-Tasks zu ersparen.

6.3. Evaluation der In-Order und Out-Of-Order Ausführung

In dieser Evaluation sollen die Vor- und Nachteile der Out-Of-Order Ausführung gezeigt werden. Dazu betrachte man das in Abbildung 6.10 dargestellte System $\mathcal{S}$ von vier Pfaden. Es ist einfach zu sehen, dass die Pfade (1, 2) und (1, 3) parallel (und sequentiell zu Pfad (1, 1)) ausgeführt werden können, während (1, 2) und (1, 4) in einem Zyklus bezüglich $<$ liegen und auf zwei verschiedenen Ressourcen ausgeführt werden müssen. Dies bedeutet, dass zwei Ressourcen zur Ausführung von $\mathcal{S}$ genügen. Die Relation $<_{\mathfrak{A}}$ ist in Tabelle 6.11 dargestellt und die Ausführungszeiten der Aufträge werden in Tabelle 6.12 gezeigt.

Es werden zwei Evaluationen durchgeführt, eine mit In-Order Ausführung und eine mit Out-Of-Order Ausführung mit dem einstufigen KHS. Die Benutzung des

$<_{\mathfrak{A}}$ auf $\mathcal{S}$	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
x_1	0	1	0	1	0	1	0	0
x_2	0	0	0	0	0	0	0	0
x_3	0	0	0	1	0	0	0	1
x_4	0	0	0	0	0	0	0	0
x_5	0	0	0	0	0	1	0	0
x_6	0	0	0	0	0	0	0	0
x_7	0	0	1	1	0	0	0	1
x_8	0	0	0	0	0	0	0	0

Tabelle 6.11.: Relation $<_{\mathfrak{A}}$ auf dem System $\mathcal{S}$

Ausführungs-zeit in sec	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
	3	1	2	4	4	1	7	1

Tabelle 6.12.: Ausführungszeiten der Aufträge

zweistufigen KHS macht hier keinen Sinn, weil nur Pfade eines Teilsystems ausgeführt werden sollen. Die Ausführungsreihenfolge sei gemäß Algorithmus 2: (1, 1), (1, 2), (1, 3), (1, 4).

Tabelle 6.13 und Abbildung 6.11 zeigen die Ausführung des Benchmarksystems In-Order, während Tabelle 6.14 und Abbildung 6.12 seine Ausführung Out-Of-Order zeigen. In beiden Evaluationen werden die Pfade (1, 1), (1, 2) und (1, 3) von Ressource 1 übernommen, während Pfad (1, 4) von Ressource 2 übernommen wird. In beiden Evaluationen startet Ressource 1 mit der Ausführung von Pfad (1, 1). Bei der In-Order Ausführung wird mit Pfad (1, 2) weitergemacht. Da Auftrag x_3 aber auf die Ausführung des relativ langen Auftrages x_7 von Pfad (1, 4) auf Ressource

	Ausführungszeit	Latenzen	Anteil	Ressource	Fertig
System $\mathcal{S}$	20	-	-	1; 2	✓
Pfad (1, 1)	4	0	1,0	1	✓
Pfad (1, 2)	11	5	0,545	1	✓
Pfad (1, 3)	5	0	1,0	1	✓
Pfad (1, 4)	11	3	0,727	2	✓
Gesamtaus-führungszeit	20				

Tabelle 6.13.: Ausführungsstatistik für das Benchmarksystem mit einstufigem KHS und In-Order Ausführung

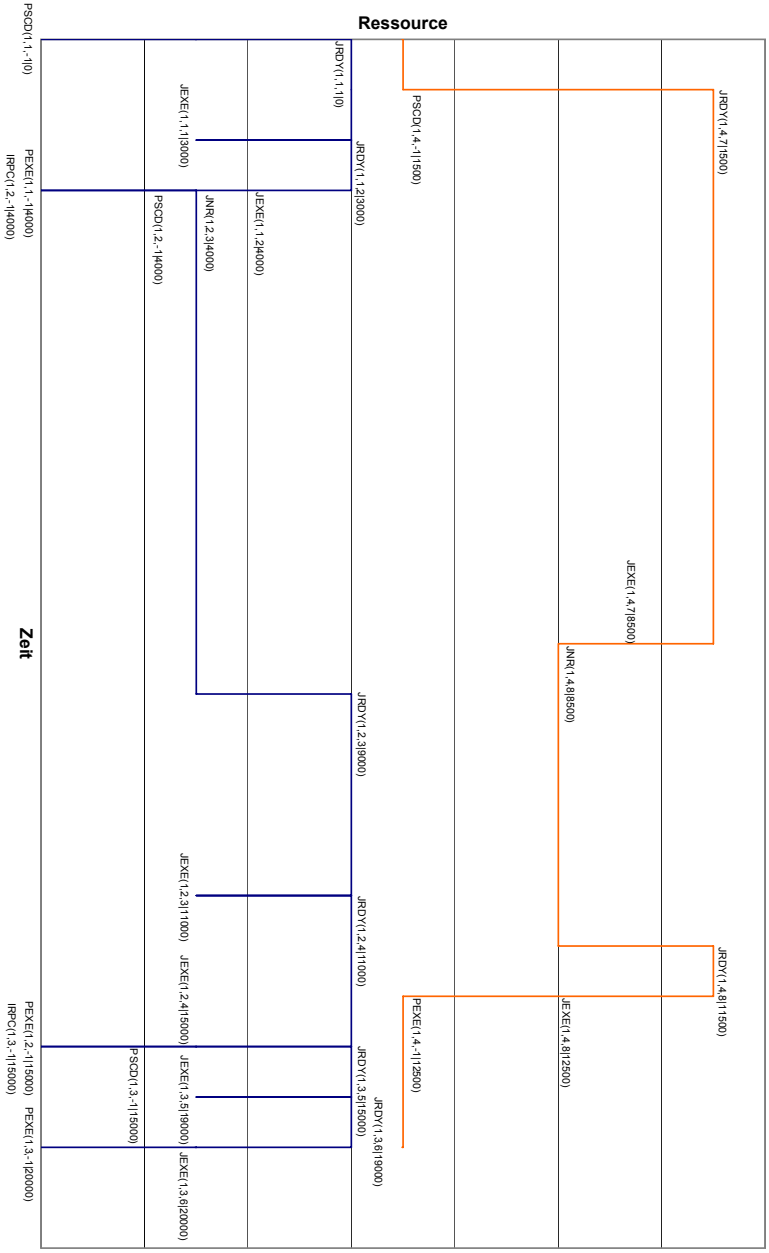


Abbildung 6.11.: Ausführung des Benchmarksystems mit einstufigen KHS In-Order

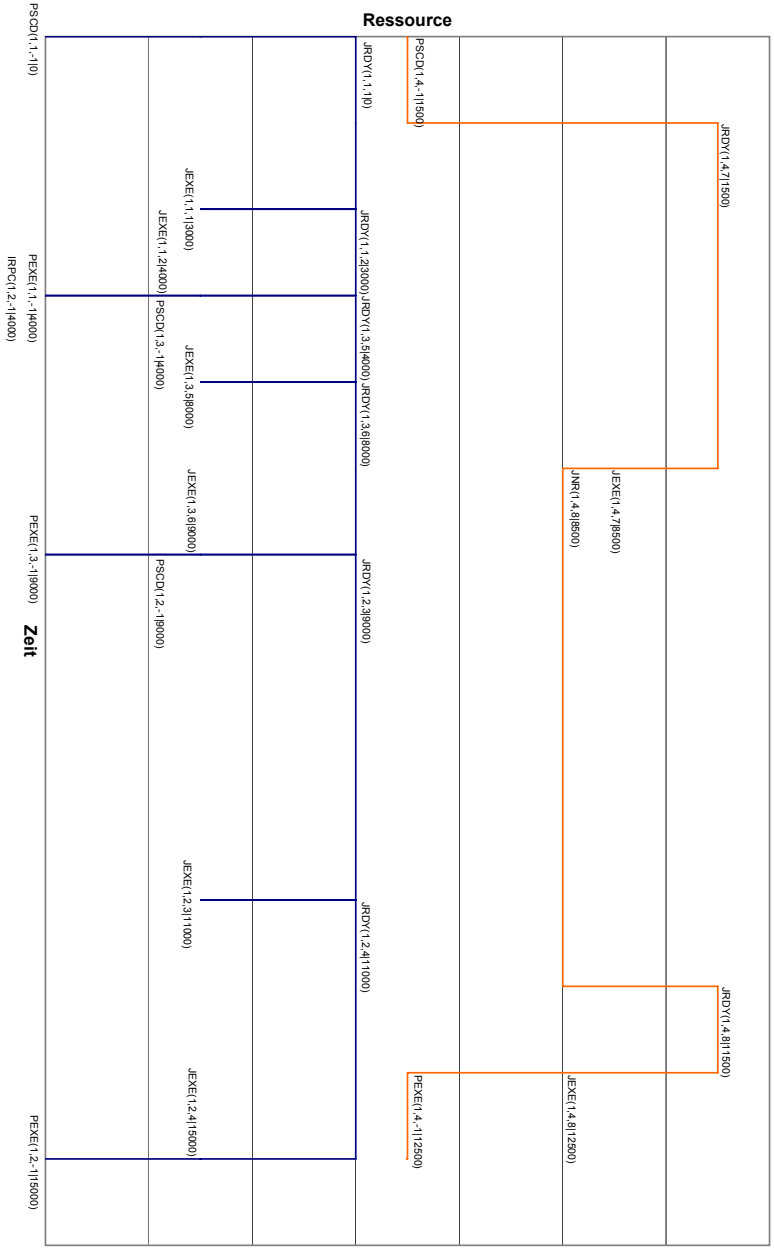


Abbildung 6.12.: Ausführung des Benchmarksystems mit einstufigen KHS Out-Of-Order

	Ausführungszeit	Latenzen	Anteil	Ressource	Fertig
System $\mathcal{S}$	15	-	-	1; 2	✓
Pfad (1, 1)	4	0	1,0	1	✓
Pfad (1, 2)	6	0	1,0	1	✓
Pfad (1, 3)	5	0	1,0	1	✓
Pfad (1, 4)	11	3	0,727	2	✓
Gesamtausführungszeit	15				

Tabelle 6.14.: Ausführungsstatistik für das Benchmarksystem mit einstufigem KHS und Out-Of-Order Ausführung

2 wartet, verzögert sich seine und die Ausführungszeit von Pfad (1, 3), so dass die Gesamtausführungszeit schließlich 20 Sekunden beträgt.

Bei der Out-Of-Order Ausführung wird auf Ressource 1 nach der Ausführug von Pfad (1, 1) mit der Ausführung von Pfad (1, 3) begonnen, weil dafür weniger nichtausgeführte Aufträge gezählt werden. Das ist hier ein großer Vorteil, denn während Ressource 2 mit der relativ langen Ausführung von Auftrag x_7 (7 Sekunden) beschäftigt ist (in denen Pfad (1, 2) nicht gestartet werden könnte), kann auf Ressource 1 der Pfad (1, 3) komplett ausgeführt werden. Nach seiner Fertigstellung wird Pfad (1, 2) ausgeführt.

Hier liegt also ein Beispiel vor, bei dem eine Out-Of-Order Ausführung zeitliche Einsparungen gegenüber der In-Order Ausführung zeigt. Das dies nicht immer der Fall sein muss, zeigen die folgenden beiden Beispiele einer In-Order und einer Out-Of-Order Ausführung des Teilsystems $\mathcal{S}_1$ von Benchmark 2 in Abbildung 6.5 in Unterkapitel 6.2, in denen die Ausführung des Pfades (1, 3) vor der Ausführung des Pfades (1, 2) erzwungen wird (im Unterschied zur Evaluation des Benchmarks in Unterkapitel 6.2). Dabei wurde das zweistufige KHS gewählt, um eventuelle Einflüsse von anderen Teilsystemen zu eliminieren.

Die Tabellen 6.15 und 6.16 zeigen eine In-Order und eine Out-Of-Order Ausführung des Teilsystems $\mathcal{S}_1$. Dabei ist deutlich zu sehen, dass die In-Order Ausführung eine deutlich kleinere Ausführungszeit (23,5 Sekunden) erzielt als die Out-Of-Order Ausführung (28,5 Sekunden). In diesem Beispiel liegt der Grund in den komplexen Abhängigkeiten der Relation $<_{\mathfrak{A}}$ zwischen den Aufträgen der Pfade (1, 2) und (1, 3), die durch den Entscheidungsalgorithmus zur Out-Of-Order Ausführung in Unterkapitel 5.2.3 nicht ausreichend erfasst werden können.

Beispiel 6.3.1. *Zum Schluß sollen noch an einem Beispiel die Rekursionsformeln zur Berechnung der Ausführbereitschaftszeit von Aufträgen aus Kapitel 4 gezeigt werden. Man beachte, das die maximale Abtastrate bei den Evaluationen auf $t_A = 0,5$ s eingestellt ist. Dazu betrachte man das Benchmarksystem (3) in Abbildung 6.10 zusammen mit der Out-Of-Order Evaluation, die in Abbildung 6.12 dargestellt ist.*

	Ausführungszeit	Latenzen	Anteil	Ressource	Fertig
$\mathcal{S}_1$	23,5	-	-	2; 10; 12	✓
Pfad (1, 1)	2,5	0	1,0	12	✓
Pfad (1, 2)	5,2	0	1,0	12	✓
Pfad (1, 3)	9	0	1,0	12	✓
Pfad (1, 4)	16	8,5	0,469	10	✓
Pfad (1, 5)	21,5	15	0,302	2	✓
Gesamtausführungszeit	23,5				

Tabelle 6.15.: Ausführungsstatistik für das Benchmarksystem (2) mit zweistufigem KHS mit In-Order Ausführung

	Ausführungszeit	Latenzen	Anteil	Ressource	Fertig
$\mathcal{S}_1$	28,5	-	-	2; 7; 12	✓
Pfad (1, 1)	2,5	0	1,0	12	✓
Pfad (1, 2)	5,2	0	1,0	12	✓
Pfad (1, 3)	9	0	1,0	12	✓
Pfad (1, 4)	22	14,5	0,341	2	✓
Pfad (1, 5)	24,5	18	0,265	7	✓
Gesamtausführungszeit	28,5				

Tabelle 6.16.: Ausführungsstatistik für das Benchmarksystem (2) mit zweistufigem KHS mit Out-Of-Order Ausführung

Die Rekursionsformeln werden an Pfad $(1, 4)$ *demonstriert. Alle Zeitangaben in diesem Beispiel sind relativ zum Start der Evaluation zu interpretieren. Der erste Auftrag von Pfad* $(1, 4)$ *ist* x_7 *und es gilt* $\mathcal{V}(x_7) = \emptyset$*. Nach der Rekursion in Gleichung* (4.1.1)*, oberer Zweig, ist* x_7 *also spätestens ausführbereit, sobald der Pfad einer Ressource zugeteilt wurde plus der Zeit der maximalen Abtastrate. Dies ist zum Zeitpunkt 1,5 s + 0,5 s = 2 s der Fall. Tatsächlich startet seine Ausführung jedoch bereits zum Zeitpunkt 1,5 s. Diese Abweichung zum Wert der Rekursion liegt darin begründet, dass die Ressource die maximale Abtastrate nicht ausgeschöpft hat. Dies ist auch in Abbildung 6.12 deutlich zu erkennen: Die Ausführung von* x_7 *beginnt ohne Zeitverlust nach der Zuteilung des Pfades zur Ressource 2.*

Nun betrachte man den zweiten und letzten Auftrag x_8 *dieses Pfades. Laut Evaluation ist* x_8 *zum Zeitpunkt 11,5 s ausführbereit. Es gilt* $\mathcal{V}(x_8) = \{x_3, x_7\}$*, in der Rekursion muss daher der untere Zweig betachtet werden. In die Maximumsbildung gehen dort die Zeiten ein, zu denen* x_3 *und* x_7 *ausgeführt sind.* x_3 *ist zum Zeitpunkt 11 s ausgeführt und* x_7 *ist zum Zeitpunkt 8,5 s ausgeführt. Nach der Rekursion ist* x_8 *also zum Zeitpunkt* $\max\{11;\, 8,5\} + 0,5$ *s = 11,5 s ausführbereit. Tatsächlich erkennt Ressource 2 den Auftrag* x_8 *zum Zeitpunkt 11,5 s als ausführbereit und startet ihn. An Abbildung 6.12 kann man erkennen, dass diesmal die maximale Abtastrate von Ressource 2 voll ausgeschöpft wurde.*

Dieses Beispiel zeigt also, dass die Rekursionen aus Kapitel 4 eine obere Schranke für die Ausführungszeiten von Aufträgen berechnen.

Fazit: In diesem Unterkapitel wird ein Beispiel vorgestellt, in man den Vorteil der Zeiteinsparung der Out-Of-Order Ausführung gegenüber der In-Order Ausführung erkennt. Weiterhin wird ein Gegenbeispiel vorgestellt, bei dem eine Out-Of-Order Ausführung noch mehr Zeit für die Ausführung benötigt. Dies zeigt, dass die implementierte Out-Of-Order Strategie in einfachen Fällen funktionieren kann, aber das sie für einen zuverlässigen Einsatz noch verbessert werden muss. Eine Idee hierbei wäre, dass in die Entscheidung, ob ein Pfad vorgezogen werden soll, auch noch die Anzahl der Pfade mit eingeht, von denen seine Aufträge abhängig sind.

7. Stand der Forschung

Da in dieser Arbeit Schwerpunkte gelegt werden, nämlich auf Pfade und ihre Ausführbarkeit und auf ihre Verteilung in einem verteilten System unter Einhaltung von Self-X-Eigenschaften, wird der Stand der Technik in den Forschungsbereichen beider Schwerpunkte betrachtet.

7.1. Pfade und ihre Ausführbarkeit

In [Mat89, CBMT96] wird eine ähnliche Definition zu der Definition der hier genutzten Relation $<_{\mathfrak{A}}$ eingeführt. Sie wird dort allerdings dazu benutzt, um Eigenschaften von synchroner und asynchroner Nachrichtenkommunikation zu studieren und auf diesen Eigenschaften sogenannte Kommunikationsklassen aufzubauen. Auf diesen werden dann Hierarchien nachgewiesen.

Auch in [Lam78] wird eine ähnliche Definition herangezogen, um sogenannte logische Uhren zu beschreiben, und damit Ereignisse (die Aufträge der vorliegenden Arbeit) verschiedener verteilter Prozesse total zu ordnen, unter Einbezug einer Zufallskomponente.

Die vorliegende Arbeit unterscheidet sich von den genannten Arbeiten insofern, als das hier zunächst ein sehr allgemeines Ausführbarkeitskriterium gefunden wird. Auf einem System ausführbarer Pfade wird dann eine zusätzliche Relation $<$ eingeführt, um darauf einen Ablaufplan zu erstellen. Ein weiterer Unterschied ist, dass sich diese Arbeit mit der Manipulation der elementaren Relation $<_{\mathfrak{A}}$ zum Zwecke der Ressourceneinsparung befasst, was in den anderen Arbeiten nicht vorkommt. Weitere Grundsteine für die vorliegende Arbeit werden in [PRB06, BPR06, BRP06] gelegt.

In [RN03] wird das weite Feld des Planens im Rahmen der Forschung über Künstliche Intelligenz aufgezeigt. Durch das Partial Order Planning (POP) werden Pläne von Tasks (hier Pfade) für zunächst einen Agenten (hier Ressource) erstellt. Dabei werden zusätzlich zu einer Ordnungsrelation (analog zur Relation $<_{\mathfrak{A}}$ in der vorliegenden Arbeit) semantische Eigenschaften des Plans mit Hilfe von Variablen und der Formulierung von Prädikaten berücksichtigt. Weiterentwickelte Varianten des POP eignen sich für die Planerstellung unter Einbezug mehrerer Agenten. Die vorliegende Arbeit unterscheidet sich jedoch stark von POP: Da hier ein Ausführungsplan unabhängig von der jeweiligen Domäne zum Einsatz in einem selbstorganisierenden System erstellt wird, wird die Semantik der Pfade nicht berücksichtigt. Weiterhin nutzen POP-Algorithmen Heuristiken, um ihre Pläne zu erstellen, und versuchen

eine starke Parallelität bei der Ausführung auf einer Ressource zu gewährleisten. In der hier vorgestellten Arbeit hingegen werden die Pfade formal analysiert um einen Ausführungsplan zu erstellen. Außerdem wird auf eine möglichst starke Sequentialisierung der Pfade Wert gelegt, um eine untere Schranke der Anzahl der für ihre Ausführung benötigten Ressourcen zu berechnen. Die Analyse- und Planungsalgorithmen können daher für die Ausführungsplanberechnung in einem selbstorganisierenden System mit einer beschränkten Anzahl von Ressourcen eingesetzt werden, und es kann vorab festgestellt werden, ob die Ressourcenanzahl für die Pfadausführung ausreicht. Auch liegen in dieser Arbeit genaue Abschätzungen über den Zeitaufwand zur Erstellung eines Planes vor, wodurch sich die Analyse- und Planungsalgorithmen für den Einsatz in Echtzeitsystemen mit harten Zeitschranken eignen. Ein weiterer Unterschied ist, dass bei POP ein Plan genau festlegt, auf welcher Ressource ein Pfad ausgeführt wird, während dies in der vorliegenden Arbeit dynamisch zur Laufzeit durch das KHS geleistet wird.

7.2. Organic Computing

Selbstorganisation wird schon seit vielen Jahren erforscht. Schon in [Jet89] und [Whi95] werden die grundlegenden Prinzipien von selbstorganisierenden Systemen untersucht, wie beispielsweise emergentes Verhalten und Fortpflanzung. Wenn man Selbstorganisation in Informatiksystemen betrachtet, so können diverse Projekte und Initiativen genannt werden.

Im DFG-Schwerpunktprogramm 1083 „Intelligente Softwareagenten und betriebswirtschaftliche Anwendungsszenarien“ [DFG06a] wird untersucht wie Agententechnologien für große Anwendungssysteme in realitätsnahen betriebswirtschaftlichen Anwendungsszenarien entwickelt und getestet werden können. Dabei werden unter anderem Prinzipien aus der Selbstorganisation genutzt. Eine Art Nachfolger stellt der DFG-Sonderforschungsbereich 637 „Self-control of Logistic Processes“ [DFG09c] dar, in dem autonome Systeme in der Logistik untersucht werden. Im DFG-Schwerpunktprogramm 1125 „RoboCup“ [DFG06b] werden ebenfalls Grundlagenforschungen durchgeführt, die Selbstorganisation betreffen. Das Design und der Betrieb großer Sensornetzwerke unter Nutzung von Selbstorganisationsprinzipien ist auch der Forschungsschwerpunkt des DFG-Graduiertenkollegs 1194 [DFG09b]. Im Jahr 2010 wurde das DFG-Schwerpunktprogramm/Transregio 89 „Invasives Rechnen (InvasIC)“ [DFG10] eingerichtet. Die Forschung basiert auf der Idee, dass parallele Programme in die Lage versetzt werden, in einer Phase (Invasion) ressourcengewahr Berechnungen auf eine Menge aktuell verfügbarer Ressourcen zu verteilen. Nach der parallelen Abarbeitung der Berechungen werden diese in einer weiteren Phase (Rückzug) wieder freigegeben. Auch hier werden Prinzipien der Selbstorganisation genutzt.

Das *Autonomic Computing Projekt* von IBM [IBM03, KC03] beschäftigt sich mit

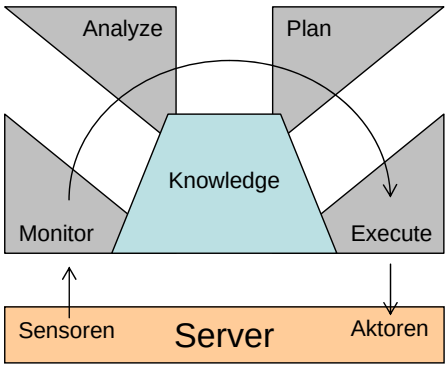


Abbildung 7.1.: Der MAPE-Zyklus

Selbstorganisation von IT-Servern in Netzwerken. Darin wurden mehrere Selbst-X-Eigenschaften wie wie Selbstoptimierung, Selbststabilisierung, Selbstkonfiguration, Selbstschutz und Selbstheilung untersucht. Für die Realisierung dieser Eigenschaften wurde der MAPE-Zyklus definiert, der aus den Teilen *M*onitor, *A*nalyze, *P*lan and *E*xecute besteht, siehe Abbildung 7.1. Der MAPE-Zyklus wird im Hintergrund parallel zu den normalen Server-Aktivitäten ausgeführt und besitzt Analogien zum autonomen Nervensystem.

Die deutsche *Organic Computing Initiative* wurde im Jahre 2003 gegründet. Ihr Hauptanliegen ist die Verbesserung der Kontrollierbarkeit komplexer Systeme, indem Prinzipien genutzt werden, die man auch in biologischen Lebewesen finden kann [VDE03, Sch05]. Erfolgreiche Organisationsprinzipien aus der Biologie werden hierbei für eingebettete Rechnersysteme angepasst. Das aktuell laufende DFG-Schwerpunktprogramm 1183 „Organic Computing“ [DFG07] wurde daher zur Vertiefung der Forschung auf diesem Gebiet eingerichtet. Die Entwicklungsphasen des Schwerpunktprogrammes werden in Abbildung 7.2 dargestellt. In der ersten Phase wurden Grundlagenforschungen über die wichtigsten Prinzipien des Organic Computing, wie zum Beispiel Selbstorganisation und Emergenz, durchgeführt. Darauf aufbauend wurden in der zweiten Phase von den teilnehmenden Projektgruppen Basistechnologien für den Einsatz in einem technischen Umfeld entwickelt, die in der momentan laufenden dritten Phase in Demonstratoren wie sich selbst steuernden Fahrzeugen oder in der Fabrikautomation implementiert und getestet werden. Viele Projekte darin erforschen die Verbesserung der Zuverlässigkeit von eingebetteten Systemen. Einige dieser Projekte sollen im folgenden vorgestellt werden:

Das ASoC-Projekt [LHR+05] strebt eine selbstorganisierende fehlerresistente System-on-Chip-Architektur (SoC) an. Es schlägt vor, einen Teil der auf heutigen Chips großzügig zur Verfügung stehenden Transistorkapazität zu nutzen, um Eigenschaften des Organic Computing zu realisieren. Ziel ist es, die Fehlertoleranz, Leistungsfähigkeit, Energieeffizienz, Wartbarkeit und Anpassungsfähigkeit von SoCs zu steigern.

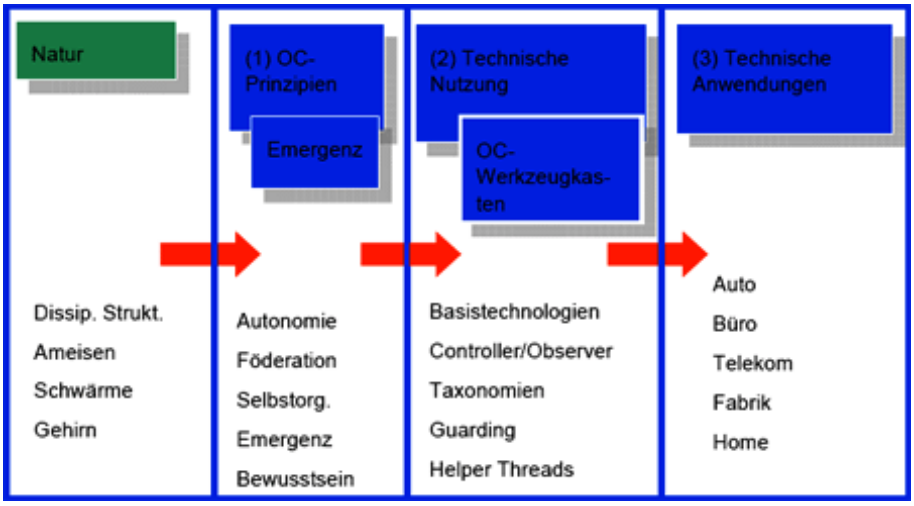


Abbildung 7.2.: Phasen des SPP 1183; Quelle: [DFG07]

Abbildung 7.3 zeigt das Grundprinzip von ASOC. Das auf einem Chip integrierte System ist in zwei Ebenen aufgeteilt. Die funktionale Ebene erfüllt, wie bei traditionellen SoCs, die eigentlichen Aufgaben des Systems. Sie enthält funktionale Elemente wie Prozessorkerne, Speicher, Ein- und Ausgabeschnittstellen, Netzwerkanbindungen etc. Darüber gibt es die Autonomic Ebene, die mit ihren Autonomic Elementen (AEs) die organischen Eigenschaften zur Verfügung stellt. Die AEs überwachen die Einheiten der funktionalen Ebene, indem sie eine Regelschleife ähnlich des oben erwähnten MAPE-Zyklus realisieren. Die Monitor-Komponente eines AE erfasst die Zustandsinformationen eines funktionalen Elementes, die ein Evaluator zusammen mit den Informationen anderer AE nutzt, um notwendige Aktionen anzustossen. Diese Aktionen beeinflussen das funktionale Element über den Aktuator, wodurch sich der Regelkreis schließt. Als zusätzliche Sicherheit überwachen sich die einzelnen AEs gegenseitig, da auch auf dieser Ebene Fehler auftreten können.

Im DoDOrg-Projekt [JKB+06] werden biologische Prinzipien untersucht, um eine neue selbstorganisierende und robuste Prozessorarchitektur zu entwickeln. Das grundlegende Konzept dieses Projektes ist das KHS, welches in Kapitel 5 vorgestellt wird. Die verschiedenen Teile des Projektes werden in Abbildung 7.4 dargestellt. Auf Hardwareebene (Cell Level) werden verschiedene sogenannte Organic Processing Elemente (OPE) entwickelt, die einerseits die regulären Aufgaben eines Prozessors bewältigen können und andererseits eine Unterstützung für die Verarbeitung der Hormone des KHS bieten. Auf Softwareebene (Organ Level) wird das KHS in der Organic Middleware implementiert und untersucht, um so eine Taskverteilung auf den heterogenen OPEs zu ermöglichen. Dazwischen ist das Ultra Low Power Management angesiedelt, durch das eine Optimierung des Energieverbrauchs angestrebt werden soll. Es nimmt über das KHS Einfluss auf Arbeitsweise der OPEs. Auf

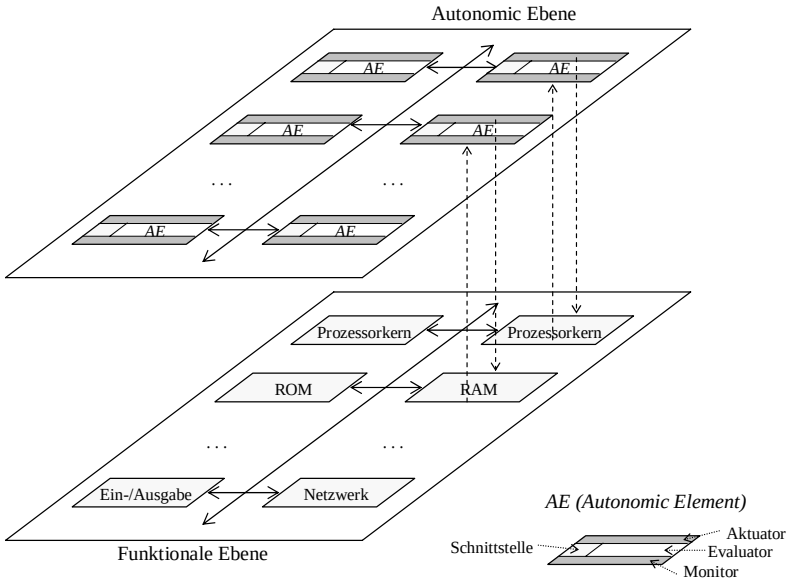


Abbildung 7.3.: Architektur des ASoC-Projektes; Quelle: [BU07]

oberster Ebene (Brain Level) agiert das Organic Robotic Control System, durch das eine Roboteranwendung überwacht und gesteuert werden soll.

Die „Observer/Controller-Architektur“ [RMB+06] ist ein oft vorkommendes Prinzip, um verlässliche selbstorganisierende Systeme zu entwerfen und bauen. Dabei wird ein System beobachtet, und es wird analysiert, ob darin beispielsweise emergentes Verhalten auftritt, wie zum Beispiel Staus oder Verkehrsballungen im Projekt „Organic Traffic Control“ [PRT+08]. Eine Beobachtungsinstanz entdeckt hierbei die eben beschriebenen emergenten Effekte, während eine Kontrollinstanz versucht, die unerwünschten emergenten Effekte wie beispielsweise Staus zu unterdrücken oder zu vermeiden und positive emergente Effekte wie einen kontinuierlichen Verkehrsfluss zu fördern. Für die Realisierung solcher Observer/Controller-Systeme werden heutzutage Learning Classifier Systeme genutzt.

Im Projekt „Organic Fault Tolerant Control Architecture for Robotic Applications“ [MLA+06] wird der aktuelle Zustand von mobilen Robotern von sogenannten organischen Kontrolleinheiten überwacht, die zusätzlich zu den Basiskontrolleinheiten implementiert wurden. Die organischen Kontrolleinheiten wenden Lernstrategien wie Online-Lernen an, um Fehler zu entdecken und ihnen gegenzuwirken. Ein Beipiel dafür ist die Modifikation der Steuerung eines Roboterarmes, um den Ausfall von anderen Roboterarmen zu kompensieren.

Das Projekt „Organic Self-organizing Bus-based Communication Systems“, siehe [WZT06], verfolgt einen dezentralen Ansatz zur Selbstorganisation der Parameter eines Kommunikationsbusses, wie beispielsweise Datenraten und Prioritäten. Dabei werden Modelle aus der Spieltheorie genutzt.

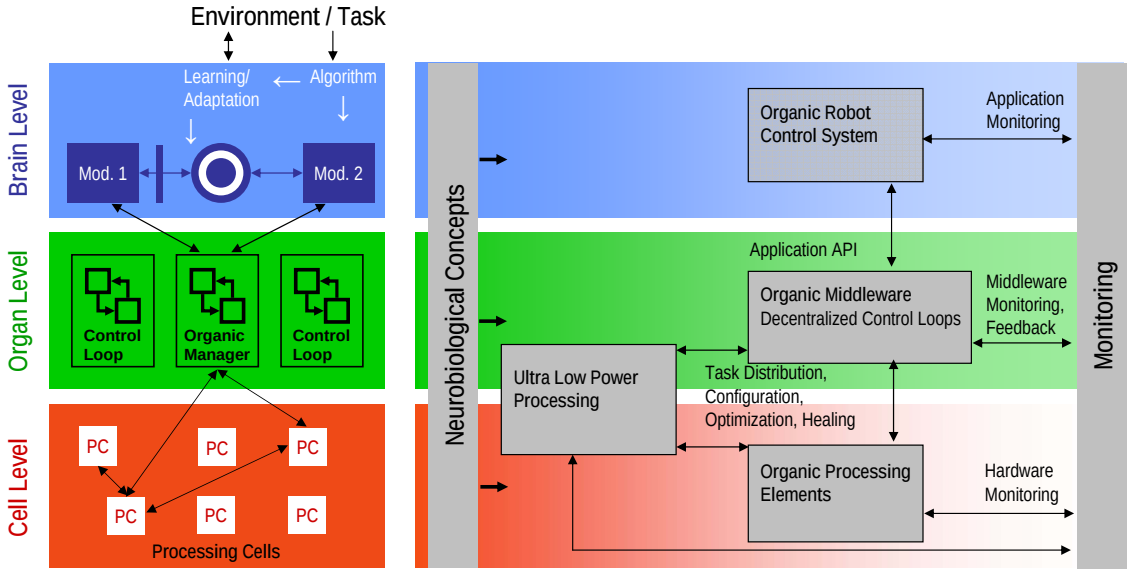


Abbildung 7.4.: Gliederung des DoDOrg-Projektes

Das Projekt „Organic Computing Middleware for Ubiquitous Environments“, siehe [TPSU07], untersucht eine Middleware-Architektur, die einen zweistufigen Ansatz für Selbstorganisation, Selbstoptimierung und Selbstheilung nutzt. Durch einen reflexbasierten Mechanismus werden schnelle Reaktionen auf Änderungen in der Umgebung des ubiquitären Systems ermöglicht, während ein langsamerer Planungsalgorithmus für das Erreichen von langfristigen Zielen genutzt wird. Ähnlich zum DoDOrg-Projekt werden dabei künstliche Hormone für die Selbstorganisation eingesetzt. Die künstlichen Hormone werden an Systemnachrichten angehängt und kennzeichnen beispielweise die Auslastung des Systems.

Ein weiteres Projekt, das sich mit selbstorganisierenden SoCs, ist das CAR-SoC-Projekt [KMUU06]. Hier wird die Entwicklung und Bewertung eines selbstorganisierenden, vernetzbaren SoC fokussiert mit dem Ziel, das CAR-SoC-System im Automobilbereich zu evaluieren. Die Gesamtarchitektur des CAR-SoC-Systems wird in Abbildung 7.5 dargestellt. Ein CAR-SoC-System besteht aus mehreren CAR-SoC-Chips, die über Schnittstellen miteinander verbunden sind. Über der Hardware liegt die System-Software (CAROS, [KUMU08]), in der ein Regelkreis zur Realisierung der lokalen Selbstorganisation implementiert ist. Dieser Regelkreis besteht im wesentlichen aus den Komponenten lokales Monitoring, lokaler Autonomic Manager und lokaler Ressourcen-Manager. Zusätzlich beinhaltet er noch eine Kommunikationseinheit und einen Security-Manager. Das lokale Monitoring beobachtet verschiedene lokale Systemparameter (Energieverbrauch, Prozessorauslastung usw.) und gibt die Zustandsinformationen an den lokalen Autonomic Manager weiter. Darauf aufbauend trifft der lokale Autonomic Manager Optimierungs- und Rekonfigura-

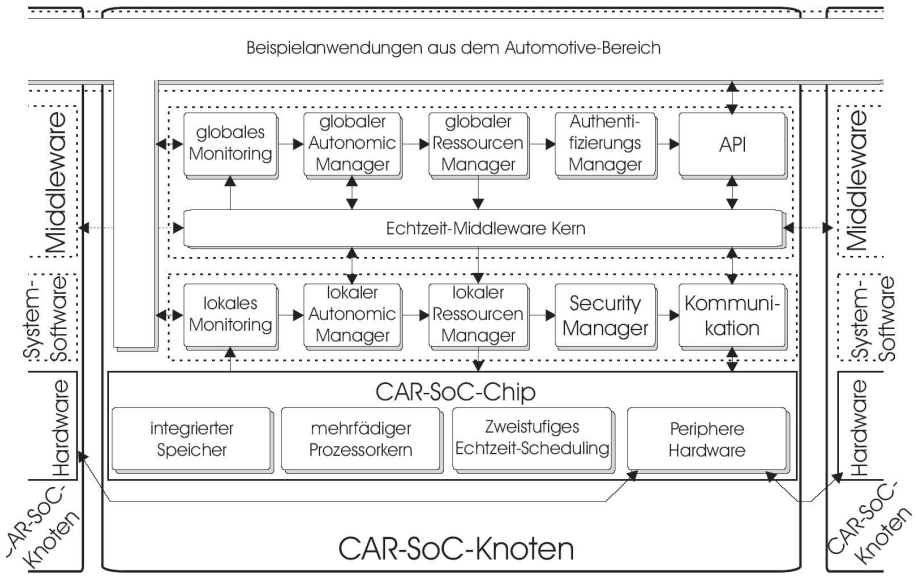


Abbildung 7.5.: Die CAR-SoC-Architektur; Quelle: [BU07]

tionsentscheidungen und lässt sie durch den lokalen Ressourcen-Manager ausführen. Die Kommunikationseinheit verwaltet die peripheren Schnittstellen, während durch den Security-Manager eine Datenverschlüsselung realisiert wird (angestossen durch den Autonomic Manger). Die CARISMA-Middleware sorgt dann auf logischer Ebene für eine Verbindung der verschiedenen CAR-SoC-Chips [NB08], und sorgt für die knotenübergreifende Selbstorganisation. Dies wird durch die Überlagerung eines globalen Regelkreises ermöglicht, welcher auf den lokalen Regelkreisen aufsetzt und der knotenübergreifende Parameter (Lastverteilung, globaler Energieverbrauch usw.) beobachtet. Der globale Autonomic Manager nutzt diese Informationen zusammen mit denen der lokalen Autonomic Manager, um Selbstorganisation auf globaler Ebene zu ermöglichen. Um den globalen Autonomic Manager gegen Ausfall abzusichern, ist er dezentral realisiert. Auf dieser Ebene werden agentenbasierte Prinzipien und Auktionsmechanismen erforscht um Selbstorganisation zu erreichen.

Die DFG-Forschergruppe 1085 „OC-Trust“ [DFG09a] befasst sich mit der Vertrauenswürdigkeit von Organic Computing Systemen. Sie untersucht verschiedene Strategien um das Vertrauen in verteilte selbstorganisierende Dienste und Rechenknoten zu evaluieren, indem beispielsweise Knoten in der direkten Umgebung oder in einer weiteren Distanz abgefragt werden [SMB+09].

Auf internationaler Ebene beschäftigt sich eine Taskforce der IEEE Computational Intelligence Society (IEEE CIS ETTC OCTF) [IEE09] mit Organic Computing. Auch andere internationale Forschungsprogramme wie etwa [EU09, CSI09] befassen sich mit Selbstorganisation für Rechnersysteme.

7.3. Organische Verteilung von Pfaden

Alle im Folgenden vorgestellten Arbeiten sind Heuristiken [CSLR90], also Algorithmen, die versuchen mit eingeschränkter Zeit und eingeschränktem Wissen eine möglichst gute Lösung des Job-Shop Scheduling-Problems [Mey92, Hen02, Bru04] zu finden. Beim Job-Shop Scheduling-Problem ist ein Schedule zur Verteilung von Jobs auf Maschinen gesucht, welches eine gegebene Zielfunktion minimiert. Die Zielfunktion im Rahmen dieser Arbeit ist die Minimierung der Ressourcen (hier: Maschinen), die zur Ausführung der Pfade benötigt werden. Andere Optimierungsziele finden sich jedoch in [Pin01, Bru04] und Mehrzieloptimierungen werden in [Yu85] beschrieben. Das Job-Shop Scheduling-Problem ist $\mathcal{NP}$-hart, und damit nach dem aktuellen Stand der Wissenschaft schwierig zu lösen. Daher sind heuristische Ansätze wie in dieser Arbeit adäquate Lösungen, um dieses Problem zu beherrschen. Ein weiterer Vorteil des Einsatzes des KHS ist, dass zusätzlich die dynamischen organischen Eigenschaften zum Tragen kommen können.

Die organische Verteilung der Pfade in dieser Arbeit baut wesentlich auf die Arbeiten in [BPR08, BPR07] auf. Hier werden die Grundlagen für das in Kapitel 5.1 vorgestellte KHS gelegt, zusammen mit detaillierten Abschätzugen über die WCET der Self-X-Eigenschaften und über die benötigte Kommunikationsbandbreite für das Broadcasten und Multicasten der Hormone. In [BRP08] wird bereits eine fortführende Arbeit vorgestellt, in der die hergeleiteten Echtzeitschranken erfolgreich evaluiert werden, während in [RBW08] ein generisches Qualitätsmass für die Güte der Taskverteilung in einem Prozessorgitter entwickelt und evaluiert wird. Die Arbeit in [BR09] untersucht die Übernahmehäufigkeiten von Tasks unter Benutzung des KHS abhängig von den gewählten Hormonwerten im Detail. Neuste Arbeiten untersuchen eine prototypische Implementierung des KHS in Middleware, wodurch Evaluationen der Echtzeitschranken und des Hormondatenaufkommens ohne den Einsatz von Simulationswerkzeugen ermöglicht werden. Die Kommunikation zwischen den einzelnen Ressourcen wird dabei über klassisches Ethernet (UDP), WLAN und in Zukunft auch per CAN-Bus realisiert. Einzelne Teile der vorliegenden Arbeit, insbesondere Überlegungen zum einstufigen und zweistufigen KHS wurden in [PB10] veröffentlicht.

Arbeiten zur Simulation des biologischen Hormonsystems wurden schon seit Mitte des 20. Jahrhunderts in [DE57, Far04, FSJ+01] vorgestellt.

Ein Ansatz eines digitalen Hormonmodells wurde von Shen in [She03, SCW02a, SCW02b, SWGC04] vorgestellt. Das Modell vereint Eigenschaften von stochastischem Schließen, dynamischen Netzwerkkonfigurationen und Selbstorganisation zur Kontrolle verteilter Roboterschwärme.

In [HS93, HS92] wird das Problem der Taskallokation in einem verteilten System von Prozessoren durch eine Analogie zur Physik angegangen, indem jede Task als Teilchen angesehen wird, auf das Kräfte wirken. Die Tasks migrieren zwischen den Prozessoren bis keine Kräfte mehr wirken. Dieser Ansatz ist zwar auch selbstor-

ganisierend und dezentral, aber im Gegensatz zu den in dieser Arbeit vorgestellten Ansätzen werden keine lokalen Regeln für die Übernahme von Tasks vorgestellt, und es können keine harten Zeitschranken eingehalten werden.

In [BMCB05] wird ein Schedulingalgorithmus vorgestellt, um Tasks in einem Prozessorgrid zu verteilen. Er wurde in der Xavantes Grid Middleware implementiert und ordnet die Tasks in Gruppen an. Dafür werden jedoch mehrere zentrale Instanzen benutzt, etwa ein *Group Manager*, ein *Process Manager* und ein *Activity Manager*. Diese Instanzen stellen eine mögliche Schwäche dar, da bei Ausfall ihre Funktionalität nicht mehr gewährleistet werden kann.

Ein weiter Ansatz wird in [RG00] gezeigt. Die Autoren untersuchen zwei Algorithmen zur Taskverteilung, Fast Critical Path (FCP) und Fast Load Balancing (FLB). Hierbei wurde FCP so konstruiert, dass Zeitbedingungen eingehalten werden können, während durch FLB gewährleistet wird, dass auch tatsächlich jeder Prozessor verwendet wird. Der Ausfall von Prozessoren wird jedoch in keinem der beiden Algorithmen berücksichtigt.

In [RSAU91] wird ein Ansatz zum Load-Balancing für die Taskallokation auf zwei Prozessoren vorgestellt. Eine Task wird dabei auf einem Prozessor mit einer Wahrscheinlichkeit ausgeführt, die umgekehrt proportional zur Länge seiner Taskwarteschlange ist. Obwohl dieser Algorithmus verteilt auf den Prozessoren ausgeführt wird, deckt er Selbst-X- und Echtzeiteigenschaften nicht ab.

Das Kanban-Prinzip [Wil08] wurde in der Toyota Motor Company entwickelt und ist ein Verfahren zur Optimierung des Produktionsablaufs bei Fließfertigung, bei dem keine zentrale Instanz für die Planung der einzelnen Fertigungsschritte vorhanden ist. Eine Station in der Fertigung fordert dabei das benötigte Material an, wenn der Vorrat zur Neige geht. Dadurch werden separate (und somit dezentrale) Regelkreise auf der Ebene der Fertigungsstationen realisiert, die es ermöglichen Lagergrößen zu optimieren. In jüngster Zeit wurde die Kanban-Idee auch für die IT-Entwicklung adaptiert [Lad08].

Weitere Ansätze für Load-Balancing werden in [Bec95, DDLM95, FPS03, FPS06, XL94] vorgestellt. Keiner von ihnen deckt jedoch das volle Spektrum von Selbst-X und Echtzeiteigenschaften ab.

In [Esc07] wird ein Konzept für eine selbstverteilende parallele Architektur, auf der ein dezentrales Scheduling möglich ist, vorgestellt, optimiert und evaluiert. Sie deckt jedoch ebenfalls das volle Selbst-X Spektrum nicht ab.

8. Zusammenfassung und Ausblick

Die vorliegende Arbeit ist im Wesentlichen in zwei Teile gegliedert: Im ersten Teil wird das Konzept der Pfade eingeführt, die aus Aufträgen bestehen, welche kausale Abhängigkeiten untereinander aufweisen. Diese Relation wird mit $<_{\mathfrak{A}}$ gekennzeichnet. Daraufhin werden Analysemethoden entwickelt, um die Pfade eines Systems auf Ausführbarkeit zu untersuchen. Wenn sich ein System als ausführbar herausstellt, so wird untersucht, in welcher Reihenfolge man seine Pfade ausführen kann, und wieviele Ressourcen dafür benötigt werden. Dazu wird die Relation $<$ auf den Pfaden eingeführt, die die kausalen Abhängigkeiten von Aufträgen auf Pfade erweitert. Wenn $<$ auf den Pfaden des Systems irreflexiv ist, dann ist es möglich, dass sie sequentiell auf einer einzigen Ressource ausgeführt werden können. Für den anderen Fall werden Methoden entwickelt, um die „kritischen“ Pfade, also solche, die bezüglich $<$ nicht irreflexiv sind, zu isolieren und durch Repräsentantenpfade zu erstetzen. Dadurch kann man auf dem verbleibenden System wieder eine Ordnung für die Ausführung auf den Pfaden definieren. Damit ist klar, dass die Anzahl der zur Ausführung benötigten Ressourcen in erster Linie durch die Größe der darin vorkommenden Zyklen von Pfaden bestimmt wird. Deshalb wird im ersten Teil der Arbeit weiterhin noch untersucht, wie man die Ressourcenzahl der zur Ausführung von solchen Pfaden optimieren beziehungsweise reduzieren kann: Zunächst erkennt man, dass man Zyklen von Pfaden isoliert vom restlichen System betrachten kann. Es wird das Konzept der Anordnung von Pfaden eingeführt, wodurch es möglich ist, $<_{\mathfrak{A}}$ auf den Zyklen und somit auf dem ganzen System zu verstärken, um eine Einsparung von Ressourcen bei der Ausführung zu erzielen. Diese Anordnungen können zunächst auf den Zyklen durchgeführt werden; falls die Pfade des Zyklus dann noch ausführbar sind, so sind es auch die Pfade des gesamten Systems. Diese isolierte Betrachtung von Zyklen kann erhebliche Rechenzeiteinsparungen nach sich ziehen. Zwei Strategien zum Finden von ausführbaren Anordnungen auf Zyklen ergänzen diese sehr theoretischen Überlegungen. Die erste Strategie befasst sich mit dem Aufbau einer Bibliothek, durch die typische Zyklen zusammen mit Schemata zur Anordnung von den darin enthaltenen Zyklen charakterisiert werden sollen. Die zweite Strategie schlägt vor, auf Zyklen zufällige Anordnungen zu wählen, und diese auf Ausführbarkeit zu testen, wenn durch die Bibliothek keine Hilfe gefunden werden kann.

Für jeden der verwendeten Algorithmen wird eine Auswandsabschätzung für die

Laufzeit angegeben. Dies zeigt, dass die Analyseverfahren und Algorithmen unter Echtzeitbedingungen eingesetzt werden können.

Zum Ende des ersten Teils werden zwei Rekursionen angegeben, durch die die Ausführungszeit der Pfade unter Idealbedingungen berechnet werden können, wenn diese zur Ausführung auf Ressourcen verteilt wurden.

Dieser Problemstellung widmet sich der zweite Teil dieser Arbeit. Es wird ein Ansatz zur Verteilung von Pfaden auf Ressourcen entwickelt, der die in Kapitel 1.1 vorgestellten organischen Eigenschaften respektiert. Dieser beruht auf dem in anderen Arbeiten entwickelten künstlichen Hormonsystem. Darauf basierend wird ein einstufiges und ein zweistufiges KHS konzipiert und weiter entwickelt, durch das die Pfade auf Ressourcen verteilt werden können. Das einstufige KHS ist dabei in seiner Arbeitsweise dem ursprünglichen KHS sehr ähnlich, während das zweistufige KHS zunächst exklusiv Ressourcen für einzelne Teilsysteme reserviert, und dann erst die Verteilung der Pfade vornimmt. Beide KHS-Typen werden um zusätzliche Eigenschaften, wie die sichere Begrenzung der Übernahme von T-Tasks und Pfaden und einem Mechanismus zu ihrer verteilten Terminierung erweitert. Am Ende des zweiten Teils werden die beiden KHS-Typen evaluiert und ihre Leistungsfähigkeit demonstriert. Es zeigt sich, dass der erste Ansatz für einen Nutzer einfacher zu handhaben ist, da die benötigten Parameter sehr leicht berechnet werden können. Beim zweiten Ansatz ist die exklusive Ressourcenreservierung hervorzuheben, durch die eine störungsfreie Ausführung der Pfade von Teilsystemen gewährleistet wird.

Fazit: Durch die in dieser Arbeit gewonnenen Erkenntnisse ist nun möglich, mit echtzeitfähigen Algorithmen die Ausführbarkeit von Pfaden zu untersuchen und den Ressourcenaufwand für ihre Ausführung zu optimieren. Weiterhin werden zwei verschiedene Ansätze des künstlichen Hormonsystems zur Allokation von Pfaden in einem verteilten System bereit gestellt, die ihre Stärken unter jeweils verschiedenen Randbedingungen voll entfalten und somit ein breites Anwendungsfeld abdecken. Für den Rechenzeitaufwand beider Ansätze können Schranken angegeben werden, was sie für den Einsatz in Echtzeitsystemen qualifiziert.

8.1. Mögliche Erweiterungen dieser Arbeit

Die in dieser Arbeit gewonnenen Erkenntnisse können in vielfältige Richtungen ausgebaut werden:

- Die in Kapitel 3.2.4 vorgestellten Zyklen in der Bibliothek können um weitere typische Zyklen und den zugehörigen Anordnungsschemata erweitert werden. Weiterhin können Anordnungsschemata wie im ersten Beispiel in der Bibliothek für ganze Klassen von Zyklen abstrahiert werden, so dass man nicht mehr an konkrete Zyklen gebunden ist.

- Zusätzlich zu den Evaluationen dieser Arbeit können Evaluationen durchgeführt werden, um die in Kapitel 5 aufgestellten Gleichungen über die maximalen Zeitschranken bei Selbstkonfiguration und Selbstheilung der beiden KHS-Typen zu untermauern und zu verfeinern. Dies kann mit den in dieser Arbeit entwickelten Simulationswerkzeugen problemlos erfolgen, und wurde hier aus Platz- und Zeitgründen aber nicht getan.

- Es können Evaluationen durchgeführt werden, die eine dauerhafte Veränderung der Eignungswerte von Ressourcen durch das Ausführen von Pfaden berücksichtigen. Als Beispiel stelle man sich den Energievorrat einer Ressource vor: Durch eine Pfadausführung wird er verkleinert, und das sollte sich im Eigungswert für die Pfade niederschlagen.

- Ein großes zusätzliches Forschungsfeld eröffnet sich, wenn Randbedingungen verschärft werden. So sollten Untersuchungen über die Ausführbarkeit von Pfadsystemen durchgeführt werden, wenn die Anzahl der zur Verfügung stehenden Ressourcen beschränkt ist, und insbesondere wenn es Ressourcen mit exklusiven Eigenschaften gibt. Dies kann zu Engpässen bei der Ausführung führen, oder gar zur Nichtausführbarkeit ganzer Teilsysteme, wenn beispielsweise eine solche Ressource bereist für ein Teilsystem allokiert ist, und daher für ein anderes Teilsystem nicht zur Verfügung steht. Deshalb wäre es ein wichtiges Ziel, Aussagen über die minimale Anzahl von benötigten Ressourcen zur Ausführung unter solchen Randbedingungen zu finden.

- In der vorliegenden Arbeit ist die Anzahl der Ressourcen insoweit stabil, dass zur Laufzeit keine neuen Ressourcen hinzukommen können. Sobald man dies zulässt, entsteht das Problem der verteilten Zustandshaltung. Dann muss den neu hinzugekommen Ressourcen mitgeteilt werden, welche Pfade bereits ausgeführt worden sind, auch um eine fälschliche Terminierung zu verhindern.

  Dazu muss eine Ressource zunächst erkennen, dass sie neu zu einem bestehenden System zugeschaltet wurde. Dies kann durch das Lauschen auf Hormone erfolgen; wenn die Ressource selbst innerhalb eines definierten Zeitraums keine Hormone sendet, so kann sie davon ausgehen, dass sie neu zum System dazu gekommen ist. Dann muss sie sich über den aktuellen Stand der Bearbeitung der Pfade informieren. Dies kann dadurch erfolgen, in dem sie den aktuellen Zustand einer bestimmten Ressource anfordert, beispielsweise derjenigen mit der kleinsten Identifikationsnummer (diese Information erhält sie durch das Lauschen auf die Hormone). Wenn dieses Problem gelöst ist, dann können auch Ressourcen, die ausgefallen waren, wieder ins System integriert werden.

- Die Möglichkeit der kaskadierten Ausführung von Pfadsystemen kann untersucht werden: In einer Anwendung kann es passieren, dass ein System aus sovielen Pfaden besteht, dass diese nicht mehr unmittelbar und sofort den

Ressourcen zugewiesen werden können. Für diesen Fall kann untersucht werden, ob es möglich ist, sukkzessive Teilsysteme des Systems abzuarbeiten, für deren Ausführung genügend Ressourcen vorhanden sind. Dazu muss das KHS um die Möglichkeit zur Ermittlung der Anzahl vorhandener Ressourcen erweitert werden, was aber durch das Aussenden eines Lebenszeichen-Hormons einer jeden Ressource realisiert werden kann. Weiterhin muss die verteilte Zustandshaltung dem KHS ermöglichen zu erkennen, ob Pfade bereits ausgeführt sind oder nicht, um eine fälschliche Terminierung von Pfaden, die in der Kaskade erst später ausgeführt werden sollen, zu verhindern.

- Schließlich kann auch das Problem einer unzuverlässigen Kommunikation untersucht werden. Wenn durch einen Sendefehler die Verteilung von Hormonen gestört wird, kann es passieren, dass das KHS auf einigen Ressourcen einen Pfad für abgebrochen hält, während er auf anderen Ressourcen als in Arbeit registriert wird. Dies kann zu mehrfacher Übernahme von Pfaden und damit zu einer fälschlicher Mehrfachausführung, also einer Semantikänderung im System, führen.

  Die resultierenden Effekte sollten systematisch erfasst werden, um Lösungsstrategien zu entwickeln, beziehungsweise um Kriterien angeben zu können, unter denen ein verteiltes System noch die Semantik des Pfadsystems einhält.

A. Anhang

A.1. Definition der transitiven Hülle einer Relation

Gegeben sei eine nichtleere Menge M mit der zweistelligen Relation R. Die transitive Hülle R^+ von R ist dann wie folgt definiert:

Definition A.1.1. *Für je zwei Elemente x und y aus M gilt:*

$$x\,R^+\,y :\Longleftrightarrow \exists n \geq 0\ \exists x_1, \ldots, x_n \in M : x\,R\,x_1\,R\,x_2\,R \ldots\ R\,x_n\,R\,y$$

Anschaulich gesehen bedeutet diese Definition, dass nun zwei Elemente x und y in der R^+-Relation stehen, wenn es einen Weg bezüglich der R-Relation gibt, der von x zu y führt.

Eine Bemerkung zur Notation: Im Folgenden wird für je zwei Elemente $x,\ y \in M$ und die Relation R synonym genutzt (Diese Notation gelte auch für alle anderen Relationen):

$$xRy :\Longleftrightarrow (x,\ y) \in R$$

Zur Charakterisierung der transitiven Hülle einer Relation werden hier noch zwei interessante Eigenschaften erwähnt. Man kann sich leicht klarmachen, dass R^+ die kleinste transitive Relation ist, die R enthält:

Beweis. R^+ ist transitiv aufgrund Definition. Außerdem ist R^+auch minimal, denn sonst gäbe es eine kleinere transitive Relation R^0, die R enthält, mit $R^0 \subset R^+$, d.h. es gäbe ein Paar $x,\ y \in M$ mit $(x,\ y) \in R^+$, aber $(x,\ y) \notin R^0$. Da das Paar aber in R^+ liegt gibt es ein $n \geq 0$ und $x_1, \ldots, x_n \in M$ mit $x\,R\,x_1\,R\,x_2\,R \ldots\ R\,x_n\,R\,y$. Da R in R^0 liegt, muss nun aufgrund der Transitivität von R^0 gelten $(x,\ y) \in R^0$, was ein Widerspruch zur Voraussetzung ist. Also ist die Annahme falsch und R^+ ist die kleinste transtive Relation, die R enthält. □

Ebenso einfach macht man sich klar, dass folgende Identität für R^+ gilt:

$$R^+ = \bigcap \{N \mid N \subseteq M \times M \text{ und } N \text{ ist transitiv und } R \subseteq N\}$$

Beweis.
„⊆“: Wenn $(x,\, y) \in R^+$ gilt, dann gibt es ein $n \geq 0$ und $x_1, \ldots, x_n \in M$ mit $x\,R\,x_1\,R\,x_2\,R \ldots\ R\,x_n\,R\,y$. Diese Folge liegt dann paarweise in jedem N, welches R enthält. Also liegt $(x,\, y)$ in jedem transitiven N, welches R enthält und somit im

Schnitt aller dieser Mengen n.
„$\supseteq$“: R^+ ist nach obiger Aussage transitiv und enthält R. Also nimmt R^+ an dem Schnitt teil, daher ist jedes Element des Schnittes auch ein Element aus R^+. □

A.2. Der Algorithmus von Warshall

Der Algorithmus von Warshall berechnet die transitive Hülle einer zweistelligen Relation auf einer Menge, die durch eine quatratische $k \times k$-Matrix repräsentiert wird. Die Vorgehensweise wird in Algorithmus 10 beschrieben. Sie basiert auf der Idee, dass in der Matrix Einträge mit einer 1 gesucht werden. Wenn ein solcher vorliegt, dann bedeutet das, dass die zugehörigen Elemente in Relation R stehen, es gilt also etwa $a_i \, R \, a_l$ $(1 \leq_l^i \leq k)$. Dann wird für jedes Element a_j $(1 \leq j \leq k)$ der der Relation zugrunde liegenden Menge untersucht, ob bereits a_i oder a_l dazu in Relation stehen. Wenn dem so ist wird, wird $a_i \, R \, a_j$ in der Matrix gesetzt. Dies realisiert die Transitivität der entstehenden Relation. Eine exakte Beschreibung des Algorithmus mit Korrektheitsbeweis findet sich in [War62].

Algorithmus 10 : Der Algorithmus von Warshall

Eingabe : Binäre $k \times k$-Matrix A, die eine zweistellige Relation beschreibt
Ausgabe : A, welche nun genau die transitive Hülle der Relation enthält

```
1  Beginn
2      für (l = 1; l ≤ k; l++) tue
3          für (i = 1; i ≤ k; i++) tue
4              wenn (A[i, l] == 1) dann
5                  für (j = 1; j ≤ k; j++) tue
6                      A[i, j] = A[l, j] ∨ A[i, j]
7                  Ende
8              Ende
9          Ende
10     Ende
11 Ende
```

Der Aufwand des Algorithmus ist nun leicht zu berechnen. In Zeile 4 werden maximal k^2 Vergleiche durchgeführt wegen der Doppelschleife, während in Zeile 6 maximal k^3 arithmetische Operationen benötigt werden.

A.3. Symmetrie der Matrix Ψ

Gegeben sei eine Matrix $A \in \mathbb{R}^{n \times n}$. Dann gilt für die Einträge der Matrizen A und A^T $a_{ij} = (a^T)_{ji}$. Somit gilt für die Matrix Ψ mit $\psi_{ij} = a_{ij} * (a^T)_{ij}$:

$$\psi_{ij} = a_{ij} * (a^T)_{ij} = (a^T)_{ji} * a_{ji} = \psi_{ji}$$

Dies zeigt die Symmetrie von Ψ.

A.4. Berechnung der direkten Vorgängermenge von Aufträgen

In diesem Unterkapitel wird vorausgesetzt, dass das System $\mathcal{S}$ von Pfaden ausführbar ist. Zunächst muss man die Vorgängermengen von Aufträgen kennen, um dann die Mengen relativ maximaler Aufträge zu berechnen. Das ist einfach, wenn man Kenntnis von der Adjazenzmatrix $\mathfrak{A}$ besitzt, denn man muss nur die zu einem Auftrag gehörige Spalte auslesen.

Algorithmus 11 : Berechnung der Vorgängermengen von Aufträgen

Eingabe : Binäre $k \times k$-Matrix $\mathfrak{A}$, die die Relation $<_{\mathfrak{A}}$ auf der Menge der Aufträge $\mathfrak{A}$ beschreibt
Ausgabe : Menge der Vorgängermengen aller Aufträge aus $\mathfrak{A}$
Beginn
  $V = \emptyset$;
  für $(j = 1;\ j \leq k;\ j++)$ **tue**
    $V(x_j) = \emptyset$; //Initialisieren der Vorgängermenge von x_j
    //System ist ausführbar, also muss der Fall $i = j$ nicht
    //untersucht werden
    für $(i = 1;\ i \leq k,\ i \neq j;\ i++)$ **tue**
      wenn $(\mathfrak{A}[i, j] == 1)$ **dann**
        $V(x_j) := V(x_j) \cup \{x_i\}$;
      Ende
    Ende
    $V := V \cup \{V(x_j)\}$;
  Ende
Ende

Es ist klar, dass die Mengen $V(x_j)$ $(1 \leq j \leq k)$ nur Aufträge x_i enthalten mit $x_i <_{\mathfrak{A}} x_j$. Sie enthalten auch alle Aufträge mit dieser Eigenschaft, denn für jeden Auftrag mit dieser Eigenschaft ist in der Matrix $\mathfrak{A}$ nach Definition an der Stelle $\mathfrak{A}[i, j]$ eine Eins gesetzt. Dies zeigt die Korrektheit des Algorithmus.

Nun zum Aufwand: Unter der Voraussetzung, dass das Initialisieren von Mengen die Dauer einer arithmetischen Operation benötigt, werden von diesem Algorithmus $k+1$ arithmetische Operationen, $k(k-1)+k = k^2$ Hinzufügeoperationen zu Listen und $k(k-1)$ Vergleichsoperationen benötigt. Mit den in den vorangegangenen Kapiteln eingeführten Zeitgrößen wird also maximal ein Zeitaufwand von $t_M n_0(k+1) + t_Z n_0 k^2 + t_V n_0 k(k-1)$ für die Ausführung des Algorithmus benötigt. Die Terminierung ist wegen der beiden Schleifenbedingungen klar.

Wenn nun die direkte Vorgängermenge eines Auftrages x berechnet werden soll, so muss man seine Vorgängermenge untersuchen. Jeder Auftrag aus der Vorgängermenge von x ist Kandidat ein Auftrag aus der direkten Vorgängermenge von x zu sein. Man wähle einen Auftrag x' aus der Vorgängermenge und untersuche nun in der Matrix $\mathfrak{A}$, ob es einen anderen Auftrag x'' aus der Vorgängermenge von x gibt mit $x' <_{\mathfrak{A}} x''$? Wenn dies der Fall ist, so ist x' nach Definition kein Auftrag aus der direkten Vorgängermenge von x. Anderenfalls gilt $x' \in \max V(x)$. Entsprechend gilt der folgende Algorithmus 12, der die direkten Vorgängermengen von jedem Auftrag aus $\mathfrak{A}$ findet. Dabei bedeutet $\mathfrak{A}\left[[x_j^l], [x_j^{l'}]\right]$ den Eintrag in Matrix $\mathfrak{A}$ in der Zeile von Auftrag x_j^l und in der Spalte von Auftrag $x_j^{l'}$.
Wegen dem oben Gesagten ist klar, dass jeder Auftrag, der zu $\max V(x_j)$ bei der Ausführung des Algorithmus hinzugefügt wird, auch tatsächlich ein Auftrag aus der direkten Vorgängermenge von x_j ist. Es werden aber auch sicher alle solche Aufträge gefunden, denn sie sind in der Vorgängermenge von x_j (siehe Algorithmus 11) und somit werden sie von diesem Algorithmus untersucht. Wegen ihrer Maximalität zu x_j ist die Eigenschaft in Zeile 9 des Algorithmus für sie aber nie erfüllt, was bedeutet, dass solch ein Auftrag richtig erkannt wird. Dies zeigt die Korrektheit des Algorithmus, die Terminierung ist klar wegen der Schleifenbedingungen und der endlichen Kardinalität der darin vorkommenden Mengen.

Nun wieder zum Aufwand des Algorithmus: Die innerste Schleife (Zeile 8) wird höchstens $k-2$ Mal durchlaufen, da in der Menge $V(x_j)$ wegen der Ausführbarkeit von $\mathcal{S}$ höchstens $k-1$ Aufträge enthalten sind und der Auftrag x_j^l nicht untersucht wird. In der Schleife werden also höchstens $k-2$ Vergleiche und höchstens eine arithmetische Operation beim Setzen der Variablen *isMax* durchgeführt.

Die mittlere Schleife (Zeile 6) wird nach dem eben Festgestellten höchstens $k-1$ Mal durchlaufen. Dabei werden eine arithmetische Operation (Zeile 7), ein Vergleich (Zeile 14), die innerste Schleife und maximal eine Hinzufügeoperation zu Listen (Zeile 15) ausgeführt.

Die äußerste Schleife (Zeile 4) wird k Mal durchlaufen. Dabei fallen eine arithmetische Operationen (Zeile 5), eine Hinzufügeoperation zu Listen (Zeile 18) und die mittlere Schleife an.

In den Zeilen 2 und 3 fallen einmalig zwei arithmetische Operationen an.

Mit den in den vorangegangenen Kapiteln eingeführten Zeitgrößen wird also maximal ein Zeitaufwand von $t_M n_0(2k^2-k+2)+t_Z n_0 k^2+t_V n_0 k(k-1)^2$ zur Ausführung

Algorithmus 12 : Berechnung der direkten Vorgängermenge von Aufträgen

Eingabe : Binäre $k \times k$-Matrix $\mathfrak{A}$, die die Relation $<_{\mathfrak{A}}$ auf der Menge der Aufträge $\mathfrak{A}$ beschreibt, und die Menge V aus Algorithmus 11

Ausgabe : Menge $V_{\max}$ der direkten Vorgängermengen aller Aufträge aus $\mathfrak{A}$

```
1  Beginn
2      V_max = ∅;
3      boolean isMax;
4      für alle V(x_j) ∈ V tue
5          max V(x_j) = ∅;
6          für alle x_j^l ∈ V(x_j) tue
7              isMax = true;
8              für alle x_j^l' ∈ V(x_j) mit l ≠ l' tue
9                  wenn (𝔄 [[x_j^l], [x_j^l']] == 1) dann
10                     isMax=false;
11                     break; //Abbruch, da kein Auftrag aus direkter
                           Vorgängermenge
12                 Ende
13             Ende
14             wenn isMax dann
15                 max V(x_j) := max V(x_j) ∪ {x_j^l};
16             Ende
17         Ende
18         V_max := V_max ∪ {max V(x_j)};
19     Ende
20 Ende
```

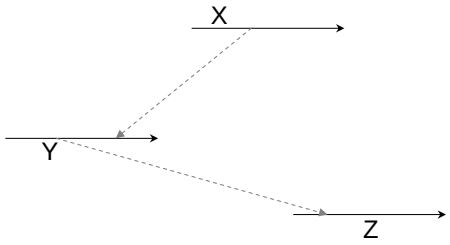


Abbildung A.1.: System von Pfaden mit Relationen $<_{\mathfrak{A}}$

des Algorithmus benötigt.

A.5. Eigenschaften von Einschränkungen der Pfad-Relationen

In den Definitionen 2.3.4 beziehungsweise 3.2.12 werden die Relationen $<$ und $<_D$ auf einem Pfadsystem $\mathcal{S}$ beschrieben, die die Abhängigkeiten von Pfaden untereinander beschreiben. Dabei besitzt $<$ durch seine Transitivität den „weiteren Horizont“. In den Definitionen 3.2.2 beziehungsweise 3.2.13 werden die Einschränkungen dieser Relationen auf ein Teilsystem $\mathcal{S}^E$ des ursprünglichen Systems $\mathcal{S}$ beschrieben.

Hier soll nun eine weitere Möglichkeit aufgezeigt werden, die Relation $<_D$ zu berechnen: Wenn man $<_D$ auf ein Teilsystem $\mathcal{S}^E$ einschränken möchte, so besagt Definition 3.2.13, dass man $<_D$ auf dem kompletten System $\mathcal{S}$ betrachtet und dann nur noch die Pfade von $\mathcal{S}^E$ und $<_D$ auf ihnen betrachtet. Äquivalent dazu ist, dass man zunächst die Pfade von $\mathcal{S}^E$ betrachtet und dann erst $<_D$ aufstellt, siehe folgendes kommutierendes Diagramm:

$$\begin{array}{ccc} \text{Pfadsystem mit } <_{\mathfrak{A}} & \xrightarrow{\text{Einschränkung auf Pfade von } \mathcal{S}^E} & \text{Pfade von } \mathcal{S}^E \text{ mit } <_{\mathfrak{A}} \\ \Big\downarrow \text{Aufstellen von } <_D & & \Big\downarrow \text{Aufstellen von } <_D \\ \text{Pfadsystem mit } <_D & \xrightarrow[\text{Einschränkung auf } \mathcal{S}^E]{} & \mathcal{S}^E \text{ mit } <_D \end{array}$$

Man kann diese Äquivalenz leicht nachrechnen, die informativ gesprochen an der festen Vorgabe von $<_{\mathfrak{A}}$ liegt.

Bei der Relation $<$ sieht dies anders aus: Laut Definition 3.2.2 wird die Einschränkung von $<$ auf ein Teilsystem $\mathcal{S}^E$ berechnet, indem man zunächst $<$ auf $\mathcal{S}$ betrachtet, und dies dann auf die Pfade von $\mathcal{S}^E$ einschränkt. Der andere Weg, zunächst die Pfade von $\mathcal{S}^E$ zu betrachten und dann erst $<$ darauf zu berechnen, führt im Allgemeinen nicht zum selben Ergebnis wie Abbildung A.1 zeigt. Sie zeigt die drei Pfade X, Y und Z und die „Hintereinander“-Relation zwischen Aufträgen dieser Pfade. Wenn man nun $<$ auf dem System $\{X, Y, Z\}$ betrachtet, so gelten folgende

Beziehungen: $X < Y$, $Y < Z$ und $X < Z$. Betrachtet man nun die Einschränkung auf $\mathcal{S}^E := \{X, Z\}$, so gilt nach Definition 3.2.2 $X < Z$.

Wenn man den anderen Weg gehen würde, so müsste man die Pfade X und Z betrachten. Zwischen ihnen besteht durch $<_{\mathfrak{A}}$ keine Beziehung, und daher würde sich auch beim Berechnen von $<$ keinerlei Beziehung ergeben.

Das bedeutet, dass man zur Berechnung der Einschränkung von $<$ auf ein Teilsystem von Pfaden im Allgemeinen nicht die Wahl hat, welchen Weg man zu ihrer Gewinnung gehen will. Dieses einfach Beispiel zeigt somit, dass man für eine Anwendung im Kontext dieser Arbeit Definition 3.2.2 für die Einschränkung von $<$ auf ein Teilsystem von Pfaden benutzen muss. Dies liegt daran, dass die transitiven Abhängigkeiten zwischen Pfaden auch betrachtet werden müssen, da man durch eine Einschränkung einen Teil der Pfade zeitweise „ausblendet“, später aber wieder berücksichtigen muss.

Eine Ausnahme ist allerdings, dass auf Zyklen bezüglich $<$ beide Berechnungswege gegangen werden können, wie Lemma 3.2.2 zeigt. Da man dann aber $<$ bereits auf $\mathcal{S}$ kennt, ist diese Aussage von geringer Bedeutung.

A.6. Modellierung durch Petrinetze

Es ist möglich, die Pfade eines Systems $\mathcal{S}$ mit Petrinetzen [Bau96] zu modellieren und damit ihre Ausführbarkeit zu prüfen. Dabei werden die Aufträge als Transitionen und die kausalen Abhängigkeiten als Vor- beziehungsweise Nachstellen modelliert.

Definition A.6.1 (Modellierung als Petrinetz)**.** *Sei $\mathcal{S}$ ein System von Pfaden. Man führe aus:*

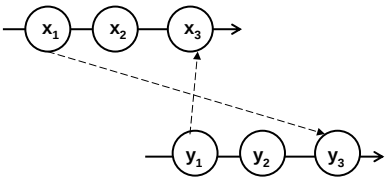
1. *Füge für den ersten Auftrag des Pfades eine Startstelle mit einer Marke ein, die zu einer Transition führt. Dann füge hinter die Transition eine Nachstelle ein. Falls ein weiterer Auftrag in diesem Pfad existiert, dann wird die Stelle mit einer weiteren Transition mit Nachstelle verbunden, die für diesen nächsten Auftrag steht. Führe den letzten Schritt sukkzessive für alle Aufträge des Pfades durch.*

2. *Führe Punkt* 1 *für alle Pfade des Systems $\mathcal{S}$ durch.*

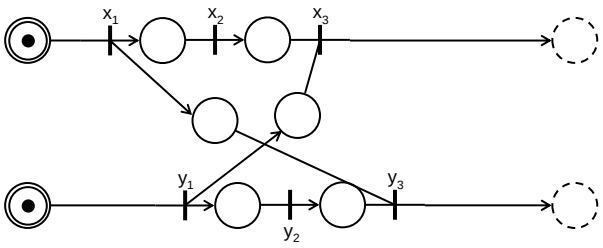
3. *Füge für alle Aufträge x und y aus $\mathcal{S}_{\mathfrak{A}}$, welche auf verschiedenen Pfaden liegen und für die $x <_{\mathfrak{A}} y$ gilt hinter der x zugehörigen Transition eine Nachstelle ein, die zu der zu y gehörigen Transition führt.*

Für alle Transitionen gilt, dass sie das Gewicht 1 *haben.*

Beispiel A.6.1. *Abbildung A.2 (a) zeigt ein Pfadsystem $\mathcal{S}$ mit zwei Pfaden, die jeweils aus drei Aufträgen bestehen, welches in Abbildung A.2 (b) gemäß Definition A.6.1 als Petrinetz modelliert wurde.*



(a) Pfadsystem S



(b) Pfadsystem S modelliert als Petrinetz

Abbildung A.2.: Pfadsystem (a), modelliert als Petrinetz (b)

Nun muss man noch einen speziellen Typ von Petrinetzen betrachten, die sogenannten Kausalnetze, wie sie etwa in [Wol] definiert sind. Die dort genannte Definition wird hier zitiert.

Definition A.6.2 (Kausalnetz nach [Wol])**.** *Ein Tripel* (B, E, G) *heißt Kausalnetz, falls* B *und* E *disjunkte Mengen sind,* $G \subseteq (B \times E) \cup (E \times B)$ *und folgende Eigenschaften gelten:*

1. *Die transitive Hülle* G^+ *von* G *ist irreflexiv.*
2. *Jedes Element* b *aus* $B \cup E$ *hat nur endlich viele Vorgänger, das heißt die Menge* $\{b' \mid [b', b] \in G^+\}$ *ist endlich.*
3. *Für jedes Element* $b \in B$ *gibt es höchstens ein* $e_1 \in E$ *mit* $[e_1, b] \in G$ *und höchstens ein* $e_2 \in E$ *mit* $[b, e_2] \in G$*.*

Bemerkung A.6.1. In dieser Definition bezeichnet B die Stellen eines Petrinetzes und E bezeichnet seine Transitionen. G ist schließlich eine Teilmenge der Kreuzproduktmengen von Stellen und Transitionen beziehungsweise von Transitionen und Stellen.

Der folgende Satz charakterisiert, wie man mit einem Petrinetz die Ausführbarkeit eines Pfadsystems prüfen kann.

Satz A.6.1. *Sei $\mathcal{S}$ ein System von Pfaden mit der Relation $<_{\mathfrak{A}}$ und P das daraus nach Definition A.6.1 gewonnene Petrinetz. Dann sind die folgenden Aussagen äquivalent:*

1. *$\mathcal{S}$ ist ausführbar.*

2. *P ist ein Kausalnetz, bei dessen Ausführung jede Transition genau einmal feuert.*

Beweis.
„1. $\Rightarrow$ 2.“: $\mathcal{S}$ ist also ausführbar. Annahme G^+ aus Definition A.6.2 wäre nicht irreflexiv. Wegen der Konstruktion von P nach Definition A.6.1 müsste es einen Zyklus in den Aufträgen von $\mathcal{S}$ geben, Widerspruch. Auch hat jedes Element b aus $B \cup E$ nur endlich viele Vorgänger, denn erstens ist die Menge der Aufträge in $\mathcal{S}$ und somit der Transitionen und Stellen in P endlich und zweitens ist, wie eben festgestellt, G^+ irreflexiv. Schließlich gilt nach Definition A.6.1, dass es für jedes Element $b \in B$ höchstens ein $e_1 \in E$ mit $[e_1, b] \in G$ und höchstens ein $e_2 \in E$ mit $[b, e_2] \in G$ gibt. Also ist P ein Kausalnetz. Weiter ist klar, dass jede Transition in P bei der Ausführung mindestens einmal feuert. Denn sonst gäbe es eine Transition, die nicht feuert. Sie könnte das nicht, weil eine ihrer Vorstellen keine Marke enthält. Da G^+ irreflexiv ist, also keine Zyklen enthält, müsste schließlich eine Vorstelle der ersten Transition eines Pfades keine Marke enthalten, im Widerspruch zur Voraussetzung. Also feuert jede Transition mindestens einmal. Jede Transition feuert aber auch höchstens einmal. Denn falls eine Transition zweimal feuern würde, würde das bedeuten, dass ihre Vorstellen wieder Marken enthalten. Diese Vorstellen müssten dann von anderen Transitionen gefüllt worden sein. Wieder wegen der Irreflexivität von G^+ und aufgrund Konstruktion von P müsste dann eine der Startmarken wieder gefüllt sein, was ein Widerspruch zur Konstruktion von P ist.
„2. $\Rightarrow$ 1.“: Klar, da P ein Kausalnetz ist, und somit G^+ irreflexiv ist. Also muss $<_{\mathfrak{A}}$ auf den Aufträgen von $\mathcal{S}$ irreflexiv sein. Alternativ: Jede Transition feuert einmal, das bedeutet, dass der zugehörige Auftrag ausgeführt werden kann. Dies gilt für alle Transitionen, also gilt die Behauptung. □

Bemerkung A.6.2. Satz A.6.1 klassifiziert also das Petrinetzmodell eines ausführbaren Systems von Pfaden als Kausalnetz. Weiter zeigt er, dass man ein solches System von Pfaden auch in einer datenflussähnlichen Art und Weise ausgeführen kann, in der jeder Auftrag genau dann ausgeführt wird, wenn alle Aufträge, auf die er wartet, bereits ausgeführt wurden. Der Grund liegt darin, dass jede Transition genau einmal feuert und daher jeder Auftrag genau einmal ausgeführt wird, und somit nicht unkontrolliert mehrere Male ausgeführt werden kann.

A.7. Technische Details zur Implementierung und Auswertung

A.7.1. Eingabeformat

```
1  Hormone Simulator
2  File Format Version 1.6
3
4  // Defining a processing element with possible tasks:
5  ProcessingElement = GridX GridY
6  PossibleTask = TTaskNumber PathNumber EagerValue Suppressor [Accelerator [
       LoadSuppressor [OfferAccelerator [OfferRate [PresetValue]]]]
7  ...
8
9  // Defining relations between tasks
10 Task = TTaskNumber PathNumber [ColorR ColorG ColorB]
11 RelatedTask = TTaskNumber PathNumber RelationValue
12 ...
13
14 // Defining sequences of paths in total order
15 TTask = TTasknumber [#MaxOccurrence] Pathnumber1  Pathnumber2...
16 ...
17
18 // Defining the jobs of a path
19 Path = TTasknumber Pathnumber Jobnumber time energy Jobnumber time energy...
20 ...
21
22 // Defining adjazenz matrix on jobs per row
23 Adjazenzmatrix = Dimension lsJob grtJob RELATION lsJob grtJob RELATION...
24 Adjazenzmatrix = Dimension lsJob grtJob RELATION lsJob grtJob RELATION...
25 ...
26
27 // Defining the parallel executable paths relative to an anchor path
28 ParallelPath = AnchorPath Pathnumber Pathnumber Pathnumber...
29 ...
30
31 // Defining the paths in a cycle of a sequence
32 Cycle = TTaskNumber CycleIdentifier Pathnumber Pathnumber Pathnumber...
33 ...
```

Listing A.1: Eingabeformat für das KHS

Listing A.1 zeigt das Eingabeformat für den Simulator für das KHS in der für diese Arbeit erweiterten Version. Ein Benutzer legt hierbei für jede Ressource (*ProcessingElement*) deren Koordinaten in einem zweidimensionalen Gitter fest (*GridX, GridY*). Danach wird für die Ressource definiert, welche Tasks (*PossibleTask*), also T-Tasks oder Pfade, auf ihr durch Angabe der eindeutigen T-Task-Nummer und der Pfadnummer ausgeführt werden können. Dabei gilt, dass eine T-Task immer die Pfadnummer Null hat, und dass eine Task mit Pfadnummer Null einer T-Task entspricht. Dann werden Eignungswerte, Suppressoren und Acceleratoren definiert, die optional um Lastsuppressoren und andere Hormontypen ergänzt werden können.

Durch die Angaben in den Zeilen 9 und 10 können Farbwerte für die Ausgabe des Simulators und Verwandschaftsgrade zwischen den T-Tasks und Pfaden festgelegt werden.

Eine T-Task (*TTask*) wird dann durch die Angabe ihrer Nummer, der maximalen Zahl ihrer Allokationen (optional, siehe Kapitel 5.2.2) im verteilten System und der Angabe der zum Teilsystem der T-Task gehörenden Pfade in der durch Algorithmus 2 gegebenen Ordnung definiert. Wenn keine maximale Anzahl der Allokationen einer T-Task angegeben wird, so wird diese Anzahl durch die Anzahl der Pfade im größten Zyklus des Teilsystems beschränkt.

Ein Pfad (*Path*) wird durch die Angabe des zugehörigen Teilsystems beziehungsweise der zugehörigen T-Task (*TTasknumber*), der Angabe der Pfadnummer (*Pathnumber*) und der Angabe der Aufträge des Pfades (*Jobnumber*) in ihrer Ordnung definiert. Für jeden Auftrag wird dabei noch seine Ausführungszeit (*time*) und die für seine Ausführung benötigte Energie (*energy*) angegeben.

Die Relation $<_{\mathfrak{A}}$ wird durch die Angabe der *Adjazenzmatrix* definiert. Für jede Zeile wird dabei ein Auftrag $x \in \mathcal{S}_{\mathfrak{A}}$ (*lsJob*) festgehalten und für alle Aufträge $y \in \mathcal{S}_{\mathfrak{A}}$ (*grtJob*) festgelegt, ob $x <_{\mathfrak{A}} y$ gilt (*RELATION = 1*) oder nicht (*RELATION = 0*). Durch die Angabe der Dimension wird sichergestellt, dass jede Zeile und Spalte genau gleichviel Einträge enthält.

Durch *ParallelPath* werden die relativ zu einem festen Pfad (*AnchorPath*) parallel ausführbaren Pfade gemäß Algorithmus 3 angegeben.

Schließlich werden durch *Cycle* die Pfade eines Zyklus angegeben. Aus Implementierungsgründen muss dafür noch das zugehörige Teilsystem und eine Zyklusnummer benannt werden.

A.7.2. Evaluation durch Logdateiauswertung

```
1  Hormone Simulator Log File for Configuration "test35_1.9TwoCycle - 7SeqPfde" using
       the two-stage KHS
2  Tuesday, December 15, 2009, 11:43:56
3
4  11:43:56.724: Hormone loop period set to 500 milliseconds
5  11:43:56.726: ECCoordinates are 1/1
6  11:43:56.726: ECCoordinates are 1/2
7  11:43:56.726: ECCoordinates are 2/1
8  11:43:56.726: ECCoordinates are 2/2
9  11:43:56.726: ECCoordinates are 3/1
10 11:44:00.727: EC 3/1, TIMESTAMP 8, PathSched scheduled Sequence 2 and Path 5
11 11:44:00.727: EC 1/1, TIMESTAMP 8, PathSched scheduled Sequence 1 and Path 1
12 11:44:00.727: EC 3/1, TIMESTAMP 8, JobRdy Sequence 2 Path 5 Job 13 starts execution
13 11:44:00.727: EC 1/1, TIMESTAMP 9, JobRdy Sequence 1 Path 1 Job 1 starts execution
14 11:44:01.227: EC 2/1, TIMESTAMP 9, PathSched scheduled Sequence 1 and Path 2
15 11:44:01.227: EC 2/1, TIMESTAMP 9, JobRdy Sequence 1 Path 2 Job 4 starts execution
16 11:44:01.727: EC 3/1, TIMESTAMP 10, JobExec Sequence 2 Path 5 Job 13 was executed
17 11:44:01.727: EC 3/1, TIMESTAMP 11, JobRdy Sequence 2 Path 5 Job 14 starts
       execution
18 11:44:02.227: EC 2/1, TIMESTAMP 11, JobExec Sequence 1 Path 2 Job 4 was executed
19 11:44:02.227: EC 2/1, TIMESTAMP 11, JobRdy Sequence 1 Path 2 Job 5 starts execution
20 11:44:03.227: EC 2/1, TIMESTAMP 13, JobExec Sequence 1 Path 2 Job 5 was executed
21 ...
```

Listing A.2: Auschnitt einer Logdatei

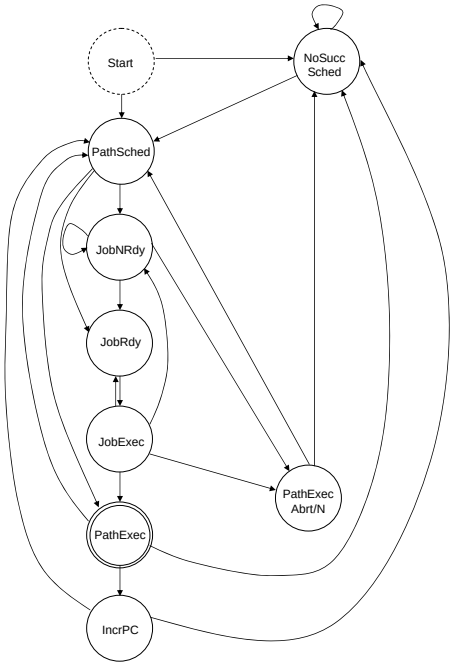


Abbildung A.3.: Endlicher Automat zum Parsen von Logdateien

Listing A.2 zeigt einen Ausschnitt einer Logdatei. Da die Evaluationen auf einem Simulator durchgeführt werden, kann dieser die Logdateiausgaben aller Ressourcen in eine einzige Datei schreiben. Die für die Auswertung essentiellen Angaben fangen hier ab Zeile 10 an: Zunächst wird die aktuelle Zeit angegeben, gefolgt von den Koordinaten der Ressource (*EC 3/1*). Die Angabe der TIMESTAMP kennzeichnet die Anzahl der durchlaufenen Hormonzyklen der Ressource. Nun folgt eine Angabe über die gerade ausgeführte Aktion der Ressource (wie in Kapitel 6 beschrieben) und welche T-Tasks, Pfade und Aufträge dabei involviert waren.

Aus den Zeitangaben und den Aktionen werden dann durch ein Auswertungsprogramm Statistiken und Diagramme wie in Kapitel 6 berechnet und erzeugt. Da diese Verarbeitung verhältnismäßig einfach ist, wird hier nicht weiter darauf eingegangen. Die einzige Herausforderung ist, anhand der Logdatei zu erkennen, ob die Abarbeitung der Pfade erfolgreich war. Daher wurde der endliche Automat aus Abbildung A.3 in das Auswertungsprogramm implementiert. Die Kanten in der Abbildung stellen genau die zulässigen Zustandsübergange bei der Pfadausführung dar und werden deshalb nicht weiter beschriftet. Wenn zusätzlich der letzte erreichte Zustand *PathExec* ist, so wurde der Pfad korrekt abgearbeitet. Dieser Finalzustand wurde in der Abbildung durch einen Doppelkreis gekennzeichnet.

Literaturverzeichnis

[Bau96] Baumgarten, Bernd: *Petri-Netze - Grundlagen und Anwendungen*. Heidelberg: Spektrum Akademischer Verlag, 1996. – ISBN 3–8274–0175–5

[Bec95] Becker, Wolfgang: *Dynamische adaptive Lastbalancierung für große, heterogen konkurrierende Anwendungen*, Universität Stuttgart, Fakultät Informatik, Germany, Dissertation, Dezember 1995. `http://www.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIS-1995-01&engl=`. – 134 S.

[BIT10a] BITKOM: *Eingebettete Systeme - Anwendungsbeispiele, Zahlen und Trends*. Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V., 2010 `http://www.bitkom.org/de/themen/54926_62539.aspx`

[BIT10b] BITKOM: *Eingebettete Systeme: Hidden Champions der Industrie*. Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V., 2010 `http://www.bitkom.org/files/documents/BITKOM-Presseinfo_Embedded_Systems_19_2_2010.pdf`

[BMCB05] Bittencourt, L. F. ; Madeira, E. R. M. ; Cicerre, F. R. L. ; Buzato, L. E.: A Path Clustering Heuristic for Scheduling Task Graphs onto a Grid. In: *3rd International Workshop on Middleware for Grid Computing (MGC05)* (2005)

[BPR06] Brinkschulte, Uwe ; Pacher, Mathias ; Renteln, Alexander von: Selbst-organisierende Middleware für eingebettete Echtzeitsysteme. In: *doIT Forschungstag 2006, Mannheim, dPunkt Verlag* (2006)

[BPR07] Brinkschulte, Uwe ; Pacher, Mathias ; Renteln, Alexander von: Towards an Artificial Hormone System for Self-organizing Real-Time Task Allocation. In: *SEUS*, 2007, S. 339–347

[BPR08] Brinkschulte, Uwe ; Pacher, Mathias ; Renteln, Alexander von: An Artificial Hormone System for Self-Organizing Real-Time Task Allocation in Organic Middleware, Springer Verlag, 2008

[BR09] Brinkschulte, Uwe ; Renteln, Alexander von: Analyzing the Behavior of an Artificial Hormone System for Task Allocation. In: *The 6th International Conference on Autonomic and Trusted Computing (ATC-09)* (2009), S. 47–61

[BRP06] Brinkschulte, Uwe ; Renteln, Alexander von ; Pacher, Mathias: A Scheduling Strategy for Real-Time Dependable Organic Middleware. In: *Conference and Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS VI), Samos, Greece* (2006), July

[BRP08] Brinkschulte, Uwe ; Renteln, Alexander von ; Pacher, Mathias: Measuring the Quality of an Artificial Hormone System Based Task Mapping. In: *Autonomics*, 2008

[Bru04] Brucker, Peter: *Scheduling Algorithms*. Springer Verlag, 2004

[BU07] Brinkschulte, Uwe ; Ungerer, Theo: *Mikrocontroller und Mikroprozessoren*. Springer Verlag, 2007

[BW05] Brinkschulte, Uwe ; Wörn, Heinz: *Echtzeitsysteme*. Springer Verlag, 2005

[CBMT96] Charron-Bost, Bernadette ; Mattern, Friedemann ; Tel, Gerard: Synchronous, Asynchronous, and Causally Ordered Communication. In: *Distributed Computing* 9 (1996), Nr. 4, S. 173–191

[CSI09] CSIRO: Centre for Complex Systems. (2009). `http://www.dar.csiro.au/css/`

[CSLR90] Cormen, Thomas H. ; Stein, Clifford ; Leiserson, Charles E. ; Rivest, Robert L.: *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990

[DDLM95] Decker, T. ; Diekmann, R. ; Lüling, R. ; Monien, B.: Universelles dynamisches Task-Mapping. (1995), S. 122–131

[DE57] Danziger, Lewis ; Elmergreen, George: Mathematical models of endocrine systems. In: *Bulletin of Mathematical Biophysics* (1957)

[DFG06a] DFG Schwerpunktprogramm 1083: Intelligente Softwareagenten und betriebswirtschaftliche Anwendungsszenarien. (2006). `http://www.realagents.org/`

[DFG06b] DFG Schwerpunktprogramm 1125: RoboCup. (2006). `http://www.informatik.uni-bremen.de/spprobocup/`

[DFG07] DFG Schwerpunktprogramm 1183: Organic Computing. (2007). http://www.aifb.uni-karlsruhe.de/Forschungsgruppen/EffAlg/projekte/oc/inhalte

[DFG09a] DFG Forschergruppe 1085: OC-Trust. (2009). http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/projects/oc-trust/

[DFG09b] DFG Graduiertenkolleg 1194: Selbstorganisierende Aktor-Sensor-Netzwerke. (2009). http://www.grk1194.uni-karlsruhe.de/index.php

[DFG09c] DFG Sonderforschungsbereich 637: Selbststeuerung logistischer Prozesse. (2009). http://www.sfb637.uni-bremen.de/

[DFG10] DFG Sonderforschungsbereich/Transregio 89: Invasives Rechnen. (2010). http://www.invasic.de/en/

[Esc07] Eschmann, Frank: *Dezentrale Ablaufplanung für selbstverteilende parallele Systeme*, Johann Wolfgang Goethe-Universität Frankfurt, Diss., 2007

[EU09] EU: Program Future Emerging Technolgies FET - Complex systems. (2009). http://cordis.europa.eu/ist/fet/co.htm

[Far04] Farhy, Leon: Modeling of oscillations of endocrine networks with feedback. In: *Methods in Enzymology* (2004)

[FPS03] Finke, Jorge ; Passino, Kevin M. ; Sparks, Andrew: Cooperative control via task load balancing for networked uninhabited autonomous vehicles. 1 (2003), S. 31 – 36

[FPS06] Finke, Jorge ; Passino, Kevin M. ; Sparks, Andrew: Stable task load balancing strategies for Cooperative control of networked autonomous air vehicles. 14 (2006), S. 789– 803

[FSJ+01] Farhy, Leon ; Straume, Martin ; Johnson, Michael ; Kovatchev, Boris ; Veldhuis, Johannes: A construct of interactive control of the GH axis in the male. In: *American Journal of Physiology - Regulatory, Integrative Comparative Physiology* (2001)

[Hen02] Henning, André: *Praktische Job-Shop Scheduling-Probleme*, Friedrich-Schiller-Universität Jena, Diss., 2002

[Heu98] Heuser, Harro: *Lehrbuch der Analysis Teil 1*. 12. Auflage. Vieweg+Teubner, 1998. – ISBN 3–519–02233–8

[HS92] Heiss, Hans-Ulrich ; Schmitz, Michael: Distributed Load Balancing Using a Physical Analogy / Universität Karlsruhe (TH). 1992. – Forschungsbericht

[HS93] Heiss, Hans-Ulrich ; Schmitz, Michael: Decentralized Dynamic Load Balancing: The Particles Approach. In: *Proc. 8th Int. Symp. on Computer and Information Sciences*. Istanbul, Turkey, November 1993

[IBM03] IBM: Autonomic Computing. (2003). `http://www.research.ibm.com/autonomic/`

[IEE09] IEEE: Organic Computing Task Force. (2009). `http://www.organic-computing.org/ieeetaskforce/`

[Jet89] Jetschke, G.: *Mathematik der Selbstorganisation*. Harry Deutsch Verlag, Frankfurt, 1989

[JKB+06] J.Becker ; K.Brändle ; Brinkschulte, U. ; Henkel, J. ; Karl, W. ; Köster, T. ; Wenz, M. ; Wörn, H.: Digital On-Demand Computing Organism for Real-Time Systems. In: *Workshop on Parallel Systems and Algorithms (PASA), ARCS 2006*. Frankfurt, Germany, March 2006

[KC03] Kephart, Jeffrey O. ; Chess, David M.: The Vision of Autonomic Computing. In: *IEEE Computer* (2003), Januar, S. 41–50

[KMUU06] Kluge, Florian ; Mische, Jörg ; Uhrig, Sascha ; Ungerer, Theo: CAR-SoC - Towards and Autonomic SoC Node. In: *Second International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES 2006)*. L'Aquila, Italy, July 2006

[KUMU08] Kluge, Florian ; Uhrig, Sascha ; Mische, Jörg ; Ungerer, Theo: A Two-Layered Management Architecture for Building Adaptive Real-time Systems. In: *6th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2008)*. Capri, Italy, October 2008

[KWB03] Kleppe, Anneke G. ; Warmer, Jos B. ; Bast, Wim: *MDA explained: the model driven architecture : practice and promise*. Addison-Wesley Professional, 2003. – 192 S.

[Lad08] Ladas, Corey: *Scrumban - Essays on Kanban Systems for Lean Software Development*. Modus Cooperandi Press, 2008

[Lam78] Lamport, Leslie: Time, Clocks and the Ordering of Events in a Distributed System. In: *Communications of the ACM 21, 7 (July 1978), 558-565* (1978)

[Las61] Lasser, Daniel J.: Topological ordering of a list of randomly-numbered elements of a network. In: *Commun. ACM* 4 (1961), Nr. 4, 167–168. http://doi.acm.org/10.1145/355578.366314. – ISSN 0001–0782

[LHR+05] Lipsa, G. ; Herkersdorf, A. ; Rosenstiel, W. ; Bringmann, O. ; Stechele, W.: Towards a Framework and a Design Methodology for Autonomic SoC. In: *2nd IEEE International Conference on Autonomic Computing*. Seattle, USA, 2005

[Mat89] Mattern, Friedemann: *Verteilte Basisalgorithmen*. Springer-Verlag Berlin Heidelberg, 1989. – 285 S.

[Mey92] Meyer, Wolfgang: *Geometrische Methoden zur Lösung von Job-Shop Problemen und deren Verallgemeinerungen*, Universität Osnabrück, Diss., 1992

[MLA+06] Mösch, F. ; Litza, M. ; Auf, A. El S. ; Maehle, E. ; Grosspietsch, K.-E. ; Brockmann, W.: ORCA - Towards an Organic Robotic Control. In: *Self-Organizing Systems, 1st International Workshop (IWSOS 2006) and 3rd International Workshop on New Trends in Network Architectures and Services (EuroNGI 2006) Proceedings, LNCS 4124, ISSN 0302-9743, Springer, Berlin, Heidelberg*, 2006, S. 251–253

[NB08] Nickschas, Manuel ; Brinkschulte, Uwe: Guiding Organic Management in a Service-Oriented Real-Time Middleware Architecture. In: *6th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2008)*. Capri, Italy, October 2008

[PB10] Pacher, Mathias ; Brinkschulte, Uwe: Real-Time Distribution of Time-Dependant Tasks in Heterogeneous Environments. In: *First IEEE Workshop on Self-Organizing Real-Time Systems (SORT 2010)* (2010), S. 8

[Pin01] Pinedo, Michael: *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 2001. – 582 S.

[PRB06] Pacher, Mathias ; Renteln, Alexander von ; Brinkschulte, Uwe: Towards an Organic Middleware for Real-Time Applications. In: *ISORC 2006, Ninth IEEE International Symposium on Object and component-oriented Real-time distributed Computing, Gyeongju, Korea* (2006), April

[PRT+08] Prothmann, Holger ; Rochner, Fabian ; Tomforde, Sven ; Branke, Jürgen ; Müller-Schloer, Christian ; Schmeck, Hartmut: Organic Control of Traffic Lights. In: *Autonomic and Trusted Computing, 5th*

International Conference (ATC 2008). Oslo, Norway, June 2008, S. 219–233

[RBW08] Renteln, Alexander von ; Brinkschulte, Uwe ; Weiss, Michael: Examinating Task Distribution by an Artificial Hormone System Based Middleware. In: *ISORC*, 2008, S. 119–123

[RG00] Radulescu, Andrei ; Gemund, Arjan J. C.: Fast and Effective Task Scheduling in Heterogeneous Systems. In: *IEEE Computer - 9th Heterogeneous Computing Workshop*. Cancun, Mexico, 2000

[RMB+06] Richter, U. ; Mnif, M. ; Branke, J. ; Müller-Schloer, C. ; Schmeck, H.: Towards a Generic Observer/Controller Architecture for Organic Computing. In: *C. Hochberger and R. Liskowsky, editors, INFORMATIK 2006 - Informatik für Menschen!, volume P-93 of GI-Edition - Lecture Notes in Informatics (LNI)*. Köllen Verlag, 2006, S. 112–119

[RN03] Russell, Stuart ; Norvig, Peter: *Artificial Intelligence: A Modern Approach*. 2nd edition. Prentice Hall, 2003

[RSAU91] Rudolph, Larry ; Slivkin-Allalouf, Miriam ; Upfal, Eli: A Simple Load Balancing Scheme for Task Allocation in Parallel Machines. In: *ACM Symposium on Parallel Algorithms and Architectures* (1991), S. 237–245

[Sch05] Schmeck, Hartmut: Organic Computing - A New Vision for Distributed Embedded Systems. In: *8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*. Seattle, USA, May 2005, S. 201–203

[SCW02a] Shen, W.M. ; Chuong, C.M. ; Will, P.: Digital hormone model for self-organization. In: *The 8th International Conference on Artificial Life* (2002)

[SCW02b] Shen, W.M. ; Chuong, C.M. ; Will, P.: Simulating self-organization for multi-robot systems. In: *2002 IEEE/RSJ International Conference on Intelligent Robots and System* (2002)

[She03] Shen, Wei-Min: Self-organization through digital hormones. In: *IEEE Intelligent Systems* (2003)

[SMB+09] Satzger, B. ; Mutschelknaus, F. ; Bagci, F. ; Kluge, F. ; Ungerer, Th.: Towards Trustworthy Self-optimization for Distributed Systems. In: *7th IFIP Workshop on Software Technologies for Future Em-*

bedded and Ubiquitous Systems (SEUS 2009). Newport Beach, USA, November 2009

[SWGC04] Shen, Wei-Min ; Will, Peter ; Galstyan, Aram ; Chuong, Cheng-Ming: Hormone-Inspired Self-Organization and Distributed Control of Robotic Swarms. In: *Autonomous Robots* 17 (2004), Juli, Nr. 1, S. 93–105

[TPSU07] Trumler, Wolfgang ; Pietzowski, Andreas ; Satzger, Benjamin ; Ungerer, Theo: Adaptive Self-optimization in Distributed Dynamic Environments. In: *1st IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*. Boston, USA, July 2007

[VDE03] VDE/ITG (Hrsg.): VDE/ITG/GI-Positionspapier Organic Computing: Computer und Systemarchitektur im Jahr 2010. In: *GI, ITG, VDE* (2003)

[War62] Warshall, Stephen: A Theorem on Boolean Matrices. In: *J. ACM* 9 (1962), Nr. 1, 11–12. http://doi.acm.org/10.1145/321105.321107. – ISSN 0004–5411

[Whi95] Whitaker, R.: Self-Organization, Autopoiesis, and Enterprises. (1995). http://www710.univ-lyon1.fr/~jmathon/autopoesis/Main.html

[Wil08] Wildemann, Horst: *KANBAN-Produktionssteuerung*. Tcw Verlag, 2008 (16). – 214 S.

[Wol] Wolf, Karsten: Vorlesungsskript zu "Modelchecking". http://www2.informatik.hu-berlin.de/~kschmidt/modelchecking/node12.html

[WZT06] Wildermann, S. ; Ziermann, T. ; Teich, J.: Organic Bus. (2006). http://www12.informatik.uni-erlangen.de/research/organicbus/?lang=en

[XL94] Xu, Chengzhong ; Lau, Francis: Decentralized Remapping of Data Parallel Computations with the Generalized Dimension Exchange Method. (1994), S. 414 – 421

[Yu85] Yu, Po-Lung: *Multiple-Criteria Decision Making: Concepts, Techniques, and Extensions*. Springer Verlag, 1985. – 402 S.

Lebenslauf

Persönliche Daten

Vorname:	Mathias
Nachname:	Pacher
Nationalität:	Deutsch
Geburtsort:	Karlsruhe, Deutschland
Geburtsdatum:	19. Dezember 1978

Ausbildung

1985 - 1998	Schulausbildung Abschluss: Abitur am Goethe-Gymnasium Karlsruhe
1998 - 1999	Zivildienst im technischen Dienst der St. Vincentius-Krankenhäuser Karlsruhe.
1999 - 2005	Studium der Informatik an der Universität Karlsruhe (TH). Abschluss: Diplom-Informatiker
2005 - 08.2008	Wissenschaftlicher Mitarbeiter am Institut für Prozessrechentechnik, Automation und Robotik, Abteilung Prof. Uwe Brinkschulte, der Universität Karlsruhe (TH). Forschung am *SIMON-Projekt* der Landesstiftung Baden Württemberg.
09.2008 -	Wissenschaftlicher Mitarbeiter an der Professur für Eingebettete Systeme von Prof. Uwe Brinkschulte des Fachbereichs Informatik und Mathematik der Johann Wolfgang Goethe-Universität. Forschung im Bereich selbst-organisierender Echtzeitsysteme.