Terminologische Sprachen zur Repräsentation und Lösung von ressourcenbeschränkten Ablaufplanungsproblemen mit Prozeßvarianten

Inaugural-Dissertation
zur Erlangung des Doktorgrades
des Fachbereichs Wirtschaftswissenschaften
der Johann Wolfgang Goethe-Universität
Frankfurt am Main

vorgelegt von
Pok-Son Kim
aus Pohang / Süd-Korea

Frankfurt 2000

Erstgutachter:	Prof. Dr. Wolfgang König
Zweitgutachter:	
Tag der Promotion:	

Danksagung

An dieser Stelle möchte ich all denjenigen meinen Dank aussprechen, die mir bei der Erstellung der Dissertation unterstützend zur Seite gestanden haben.
Zunächst einmal bin ich Herrn Prof. Dr. Wolfgang König dafür zu Dank verpflichtet, daß er die Betreuung meiner Dissertation übernommen hat und mich zusammen mit Herrn Dr. Oliver Wendt in die Dissertationsthematik eingeführt hat. Sein Interesse an meiner Promotion war stets motivierend, sein Bemühen, gute wissenschaftliche Rahmenbedingungen bereitzustellen, war mir im Verlauf der Promotion eine große Unterstützung.

Ferner möchte ich Herrn Prof. Dr. Manfred Schmidt-Schauß dafür danken, daß er in Fortsetzung seiner Betreuung meiner Diplomarbeit auch meine Promotion begleitet hat. Sein großes Interesse an meiner Arbeit und die vielen wertvollen Hinweise und Anregungen werde ich dankbar in Erinnerung behalten.

Meinem Ehemann Arne Kutzner danke ich für die Korrektur meiner Dissertation im Bezug auf die deutsche Sprache. Sein teilweise offenes Bekunden von Mißmut über meine Arbeit resultierte häufig in größeren, jedoch zumeist konstruktiven Diskussionen. Meiner Mutter und meinen Geschwistern bin ich dafür zu Dank verpflichtet, daß sie soviel Geduld mit mir hatten und mich immer zum Weitermachen angehalten haben. Meinen Schwiegereltern danke ich für ihre motivierenden und unterstützenden Worte.

Zu erwähnen ist hier auch das an der Professur stattfindende Doktorandenkolloqium, das mir die Möglichkeit gab, meine Ergebnisse einem breiten Publikum vorzustellen. In diesem Zusammenhang möchte ich mich bei den Assistenten der Professur für ihr Engagement bedanken.

Dem Land Hessen und der Johann Wolfgang Goethe-Universität möchte ich für die Unterstützung meiner Promotion durch ein Stipendium in den Jahren 1996 bis 1998 danken, das mir eine große Hilfe war.

Frankfurt am Main, im Juni 2000

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Symbol- und Abkürzungsverzeichnis

$A(\sigma)_i$	Ausdruck aus $\mathcal{RSV}$
t_{SL}	Zeitpunkt, auf dem sich die Scan-line befindet
(D, t_{SL})	$\mathcal{RSV}$-Diagramm
$\mathtt{p} = (X, P)$	Netzplan
$evalS$, $evalSL$	Funktionen
seq	Operator der Sprache $\mathcal{RSV}$
pll	Operator der Sprache $\mathcal{RSV}$
xor	Operator der Sprache $\mathcal{RSV}$ und der Sprache $\mathcal{RCPSV}$
hnet	Operator der Sprache $\mathcal{RCPSV}$
$t(H)$	Haltepunkt der Scan-line
$T = \{1, 2, \cdots, \|T\|\}$	Knotenmenge
$E \subseteq T \times T$	Kantenmenge
(T, E)	gerichteter Graph
$\longrightarrow$	grafische Repräsentation des Operators '**seq**'
$\boxed{\|\|}$	grafische Repräsentation des Operators '**pll**'
$\boxed{+}$	grafische Repräsentation des Operators '**xor**'
$\doteq$	Äquivalenzbeziehung
σ	aktiver Ablaufplan
a, b, c, d, e, r	Ressourcen
A, B, C, D	Aktionen
$C \doteq D$	Schedulingaxiom
E	das Erfüllbarkeitsproblem
$\mathcal{I}$	eine Interpretation
$.^{\mathcal{I}}$	die Interpretationsfunktion von $\mathcal{I}$
L, L_1, L_2, L_3	Sprachen bzw. Probleme
$LE(P(r,t))$, $RE(P(r,t))$	linker sowie rechter Endpunkt einer Grundaktion $P(r,t)$
$L(TM)$	die von einer Turingmaschine TM akzeptierte Sprache
$\mathcal{NP}$	Menge aller Probleme, die mittels einer nichtdeterministischen polynomiell beschränkten Turingmaschine akzeptiert werden können

NT	nichtdeterministische Turingmaschine
$\mathcal{P}$	Klasse der *effizient lösbaren* Probleme
$P(r,t), P_1(a,t), \cdots$	Grundaktionen
$\mathcal{RCPS}$	resource constrained project scheduling
$\mathcal{RCPS}V$	resource constrained project scheduling with variants
$\mathcal{RCPS}V_{pa}$	resource constrained project scheduling with variant primitive activities
$\mathcal{RSV}$	resource (constrained project) scheduling with variants
$\mathcal{RSV}^*$	das $\mathcal{RSV}$-Entscheidungsproblem
$\mathcal{RSVX}$	$\mathcal{RSV}$ extended
SE	spezielle Erfüllbarkeit
TM	Turingmaschine

Kapitel 1

Einleitung

Ein materielles bzw. immaterielles Gut kann in der Praxis häufig auf verschiedene Art und Weise produziert werden. Besonders der Projektablauf bzw. Produktionsablauf, das heißt die Festlegung der Terminlichkeiten, wann mit welchem Ressourceneinsatz was zu tun ist, um das Gesamtgut zu erhalten, ist häufig variabel. Die Existenz variabler Abläufe beruht auf den beiden in der Praxis häufig auftretenden Sachverhalten:

- Die benötigten Ressourcen können während des Projektablaufs bzw. Produktionsablaufs nur beschränkt zur Verfügung gestellt werden.
- Zur Erfüllung einer Teilaufgabe kann zwischen mehreren alternativen Handlungsvarianten ausgewählt werden, die alle dieselbe Aufgabe erfüllen.

Die Problemklasse, die die Formulierung von Problemen erlaubt, die den oberen Anforderungen gerecht werden, wird als die Klasse der Ablaufplanungsprobleme [1] bezeichnet.
Ein Ausschnitt dieser Problemklasse bildet das ressourcenbeschränkte Ablaufplanungsproblem $\mathcal{RCPS}$ (***r****esource* ***c****onstrained* ***p****roject* ***s****cheduling*), bei dem unter Berücksichtigung beschränkter Ressourcen die zeitliche Zuordnung von Teilprojekten zu Ressourcen im Rahmen einer Produktions- bzw. Projektplanung untersucht wird. Eine Erweiterung des $\mathcal{RCPS}$-Problems bildet das $\mathcal{RCPSV}$-Problem (***r****esource* ***c****onstrained* ***p****roject* ***s****cheduling with* ***v****ariant processes*). Beim $\mathcal{RCPSV}$-Problem (siehe [KWR93], [Rit98]) besteht gegenüber dem $\mathcal{RCPS}$-Problem die zusätzliche Möglichkeit, alternative Aktivitäten bzw. Aufgabenträger zur Erfüllung einer Aktivität auszuwählen. Diese zusätzliche Eigenschaft wird oft als fehlend beim $\mathcal{RCPS}$-Problem angesehen.

[1] Die genauen Definition des Begriffs "Ablaufplanugsproblem" kann in [Wit93] gefunden werden.

In dieser Arbeit wird zudem eine weitere Klasse von Ablaufplanungsproblemen vorgestellt, die bisher nicht in der Literatur aufgeführt wird und mit dem $\mathcal{RCPSV}$-Problem sehr verwandt[2] ist. Für diese neue Klasse wird die Bezeichnung $\mathcal{RSV}$-Problem (***r****esource constrained project* ***s****cheduling with* ***v****ariant processes*) gewählt. Das $\mathcal{RSV}$-Problem ist eine Oberklasse des des Job Shop-Problems und kann als Subklasse des $\mathcal{RCPSV}$-Problems angesehen werden. Das $\mathcal{RSV}$-Problem zeichnet sich durch die besondere Eigenschaft aus, eine elegante und transparente Problemspezifikation zu ermöglichen.

Eine Aufgabe der Wirtschaftsinformatik ist es, algorithmische Verfahrensweisen aufzuzeigen, die unter Berücksichtigung der oben genannten Nebenbedingungen eine Rechner-gestützte Ermittlung zeitlich optimaler Ablaufpläne erlauben. Um eine Rechner-gestützte Ermittlung zeitlich optimaler Ablaufpläne zu ermöglichen, werden sowohl *eine angemessene Methode zur Formulierung* der Ablaufplanungsprobleme als auch *effektive Algorithmen* zum Berechnen zeitlich optimaler Ablaufpläne benötigt.

In dieser Arbeit wird eine *neuartige logikbasierte terminologische Repräsentation* (*eine neuartige Methode zur Formulierung*) des $\mathcal{RSV}$- sowie des $\mathcal{RCPSV}$-Problems vorgestellt. Ferner werden auf Basis dieser Repräsentationen Lösungen beider Problemklassen (*effektive Algorithmen* zum Berechnen zeitlich optimaler Ablaufpläne) entwickelt. In diesem Kontext wird die zeitliche Komplexität (Rechenaufwand) der Ablaufplanungsprobleme untersucht.
Die Idee der Repräsentation der Ablaufplanungsprobleme mittels einer terminologischen Sprache beruht auf $\mathbf{KL-ONE}$ basierten, terminologischen Wissensrepräsentationssprachen (siehe z. B. $\mathcal{ALC}$ [SSS88]).

Terminologische Wissensrepräsentationssysteme, deren Entwicklung durch die Debatte über die Rolle der Logik in der Künstlichen Intelligenz motiviert wurde, sind Werkzeuge zur Darstellung und Auswertung von in Wissensbasen festgehaltenen Kenntnissen über eine modellhafte Umwelt. Zur Spezifikation des Wissens werden in derartigen Systemen terminologische Sprachen (oder Konzeptsprachen) [BS85] verwendet, die auf einer *klaren, einfachen* und *abstrakten Darstellungsmethode* basieren.
Die bisher entwickelten Methoden zur Darstellung der Ablaufplanungsprobleme basieren meist auf der linearen Programmierung. Bei einer derartigen Problemdarstellung sind die Ablaufstruktur sowie der Inhalt eines Ablaufplanungsproblems (z. B. welche Aktivität welche Ressource beansprucht) i. a. schwer ablesbar. Im Gegensatz dazu können von einem Ablaufplanungsproblem, das mittels einer in dieser Arbeit vorgestellten terminologischen Repräsentationsmethode formuliert wird, die Ablaufstruktur sowie der Inhalt direkt und klar abgelesen werden. Wesentlich ist dabei, daß die Syntax und Semantik einer terminologischen Spra-

[2] Die Natur der Verwandtschaft wird in den Kapiteln 3 und 8 dieser Arbeit ausführlich betrachtet.

che eine terminologische Logik bilden und diese Logik einen effektiven Ansatz zur Problemlösung bietet. Im Rahmen dieser Arbeit werden die zu Beginn erwähnten Ablaufplanungsprobleme mittels einer eigens angepaßten terminologischen Logik dargestellt und gelöst. Nähere Erläuterungen über die eigens angepaßte terminologische Logik befinden sich in Abschnitt 2.3.

In Kapitel 2 werden eine Reihe von Grundlagen vorgestellt, die für das Verständnis dieser Arbeit erforderlich sind.

Für das Problem $\mathcal{RSV}$ werden *eine eigene logik-basierte Schedulingsprache* sowie *ein neues Diagramm-basiertes Berechnungsmodell* entwickelt. Mittels der Schedulingsprache kann ein beliebiges $\mathcal{RSV}$-Problem *einfach* und *klar* terminologisch formuliert werden. Ferner kann auf dieser Sprache ein Kalkül definiert werden, mit dem aus einem Ausdruck A aus $\mathcal{RSV}$ durch Termumformungen ein semantisch äquivalenter, *normalisierter* Ausdruck A' aus $\mathcal{RSV}$ abgeleitet werden kann. Ein normalisierter Ausdruck A' ist strukturell einfacher, in ihm werden alle in A enthaltenen *nichtredundanten* Netzpläne einzeln getrennt beschrieben. Diese strukturelle Einfachheit erlaubt eine vereinfachte algorithmische Berechnung der zeitlich optimalen, ressourcenkonfliktfreien Ablaufplanungen.
Das zu $\mathcal{RSV}$ gehörige *Diagramm-basierte Berechnungsmodell* verwendet ein Scan-line-Prinzip, unter dessen Zuhilfenahme die entstehenden Ressourcenkonflikte aufgelöst werden können. Die auf einer Scan-line basierende Methode zur Auflösung entstehender Ressourcenkonflikte ist *neu* und wurde bisher nicht in der Literatur vorgestellt. Das Berechnungsmodell zeichnet sich durch die folgenden Eigenschaften aus:

- Für jeden zugehörigen Netzplan eines $\mathcal{RSV}$-Problems können mittels des Berechnungsmodells *alle nichtredundanten aktiven* Ablaufpläne berechnet werden, d.h. jeder zugehörige *aktive* Ablaufplan wird *nur genau einmal* berechnet.

- Rechenvorgänge können mittels Diagrammen dargestellt werden, in deren Zentrum eine Scan-line steht. Dies ermöglicht ein einfaches Verständnis der Berechnungsvorgänge.

In Kapitel 3 werden die logikbasierte Schedulingsprache $\mathcal{RSV}$ sowie ein zugehöriger Kalkül vorgestellt.

Das Berechnungsmodell des $\mathcal{RSV}$-Problems wird in Kapitel 4 vorgestellt. In diesem Zusammenhang wird ein Korrektheitsbeweis geführt, bei dem formal bewiesen wird, daß für einen beliebigen Netzplan eines $\mathcal{RSV}$-Problems das Berechnungsmodell alle *nichtredundanten* Ablaufpläne berechnet.

In Kapitel 5 werden weitere *algebraische Eigenschaften* untersucht, die die Sprache $\mathcal{RSV}$ aufweist. Diese Eigenschaften sind für große Projekte, die in sequentiell durchführbare Teilprojekte zerlegbar sind, besonders interessant.

In Kapitel 6 wird *eine weitere terminologische Schedulingsprache* vorgestellt, unter deren Zuhilfenahme das $\mathcal{RCPSV}$-Problem dargestellt und gelöst werden kann. Ferner wird ein zugehöriger Kalkül vorgestellt, unter dessen Zuhilfenahme genauso wie beim $\mathcal{RSV}$-Problem ein Ausdruck in einen semantisch äquivalenten *normalisierten* Ausdruck transformiert werden kann.

In Kapitel 7 wird *ein algorithmisches Verfahren* vorgestellt, mit dem das $\mathcal{RCPSV}$-Problem gelöst werden kann. Dieser Algorithmus basiert wie zuvor beim $\mathcal{RSV}$-Problem auf Diagrammen, bei denen das Scan-line-Prinzip zur Berechnung aller nichtredundanten aktiven Ablaufpläne benutzt wird. Für den Algorithmus wird abschließend formal bewiesen, daß er für einen Netzplan eines $\mathcal{RCPSV}$-Problems alle *nichtredundanten* Ablaufpläne berechnet.

In Kapitel 8 wird formal bewiesen, daß von der Ausdruckskraft her die beiden Sprachen $\mathcal{RSV}$ und $\mathcal{RCPSV}$ *gleich mächtig* sind, d.h. jeder $\mathcal{RCPSV}$-Ausdruck kann als ein $\mathcal{RSV}$-Ausdruck dargestellt werden und umgekehrt. Ferner wird gezeigt, daß die Sprache $\mathcal{RCPSV}$ *alleinig von der syntaktischen Repräsentationsfähigkeit eines Problems her* eine *echte* Oberklasse der Sprache $\mathcal{RSV}$ bildet.

Mit dem Modell der Turingmaschine kann der Aufwand an Rechenzeit für die maschinelle Berechnung eines Problems gemessen werden. Die $\mathcal{NP}$-Vollständigkeitstheorie baut auf dem Modell der Turingmaschine auf und wird im Bereich der theoretischen Informatik als ein Bewertungskriterium für die relative Komplexität von Problemen benutzt. Ein $\mathcal{NP}$-vollständiges Problem entspricht einem der schwierigsten Probleme aus der Klasse $\mathcal{NP}$. In Kapitel 9 wird der zeitliche Rechenaufwand des $\mathcal{RSV}$-Problems untersucht und bewiesen, daß das $\mathcal{RSV}$-Problem *ein $\mathcal{NP}$-vollständiges Problem* ist.

Kapitel 2

Grundlagen

In diesem Kapitel werden eine Einführung sowie ein Überblick über die in dieser Arbeit behandelten Problemstellungen gegeben. Ferner werden wissenschaftliche Grundlagen, auf denen diese Arbeit aufbaut, kurz vorgestellt.
In Abschnitt 2.1 werden zunächst die Problemklassen der Ablaufplanungsprobleme dargelegt. Im darauf folgenden Abschnitt 2.2 werden Methoden vorgestellt, mit denen Ablaufplanungsprobleme graphisch dargestellt werden können. In dieser Arbeit werden bisher nicht in der Literatur vorgestellte Methoden zur Darstellung und Lösung der Ablaufplanungsprobleme entwickelt. Eine Grundlage dieser neuen Methoden bilden terminologische Wissensrepräsentationssysteme. Deshalb wird in Abschnitt 2.3 eine kurze Einführung in Wissensrepräsentationssysteme gegeben. Als Implementierungsmedium für die in dieser Arbeit vorgestellten Lösungsmodelle wird die funktionale Programmiersprache 'Haskell' [HAB+96], [PHA+97] verwendet. Aus diesem Grund wird in Abschnitt 2.4 eine kurze Einführung in den Themenbereich der funktionalen Programmierung gegeben. Im Rahmen dieser Arbeit wird die zeitliche Komplexität der Ablaufplanungsprobleme untersucht. Die Untersuchungen über die zeitliche Komplexität werden auf dem Modell der Turingmaschine basieren. In Abschnitt 2.5 wird in diesem Zusammenhang eine kurze Einführung in den Themenbereich der Komplexitätstheorie gegeben.

2.1 Ablaufplanung

Seit den 50'er Jahren haben sich viele Wissenschaftler mit den Ablaufplanungsproblemen beschäftigt. In den letzten Jahren wurde die Problemklasse der Ablaufplanung immer mehr erweitert, da die Fertigstellung von Produkten bzw. Projekten immer komplexere Ablaufstrukturen verlangte. Zu Beginn wurde als kleinste Problemklasse die "Maschinenbelegungsplanung" betrachtet. Ein Maschinenbelegungsproblem besteht aus mehreren Aufträgen, wobei jeder Auftrag eine

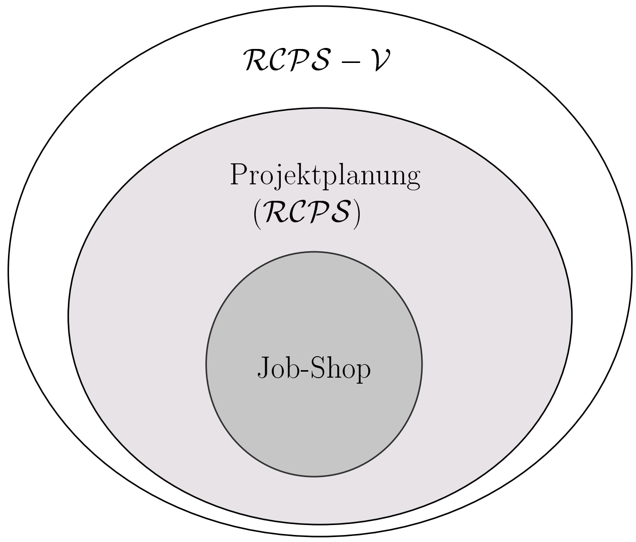


Abbildung 2.1: Problemklassen der Ablaufplanung

oder mehrere Maschinen zu durchlaufen hat. Wenn hierbei ein elementarer Arbeitsteil[1] als *ein Arbeitsgang* bzw. *eine Operation* aufgefaßt wird, besteht ein Auftrag aus mehreren Arbeitsgängen, die sequentiell durchgeführt werden. Bei einer Maschinenbelegungsplanung dürfen keine Präzedenzbeziehungen zwischen zwei Arbeitsgängen bestehen, die zu unterschiedlichen Aufträgen gehören.
Die Problemklasse "Maschinenbelegungsplanung" wurde dann zu der Problemklasse "Projektplanung ($\mathcal{RCPS}$ = resource constrained project scheduling)" erweitert. Bei einer Projektplanung können zwischen beliebigen Arbeitsgängen Präzedenzbeziehungen fest vorgegeben werden, wobei als Nebenbedingung zu erfüllen ist, daß keine zyklischen Abhängigkeiten entstehen. Im Rahmen der Projektplanung ist es daher möglich, komplexere Ablaufstrukturen zu betrachten.
In den 80'er Jahren wurde dann die Problemklasse "Projektplanung" zur Problemklasse "$\mathcal{RCPS}V$ ($\mathcal{RCPS}$ with variants)" erweitert. Die Erweiterung besteht darin, daß die Berücksichtigung alternativer Vorgänge (Grundaktivitäten[2]) bzw. Vorgangsnetze (Teilabläufe, Aktivitäten) für einen beliebigen Teil einer Aufgabe ermöglicht wurde. Die Subklassenbeziehung zwischen den drei Problemklassen zeigt Abbildung 2.1.

Bei der Ablaufplanung[3], bei der unter Berücksichtigung vorgegebener Präzedenz-

[1] ein nicht weiter zerlegbarer Arbeitsteil (Ein elementarer Arbeitsteil wird auf einer Maschine bearbeitet.)

[2] Als Grundaktivitäten werden elementare Aktivitäten, d.h. nicht weiter zerlegbare Aktivitäten, bezeichnet. Diese können auch als nicht weiter zerlegbare Produktionsabschnitte aufgefaßt werden.

[3] [Pin95] definiert Ablaufplanung (*engl.* scheduling) als den Prozeß, vorgegebene Aktivitäten unter Einhaltung von technischen und ökonomischen Nebenbedingungen (*engl.* constraints) zeitlich auf Ressourcen zu verteilen.

beziehungen sowie beschränkter Ressourcen die Terminlichkeit, wann welcher Arbeitsgang bzw. welche Aktivität mittels welcher Ressource zu bearbeiten ist, festgelegt werden soll, können verschiedene Zielkriterien verfolgt werden[4]. Im Rahmen dieser Arbeit wird hauptsächlich das Ziel der Minimierung der Gesamtbearbeitungszeit - letztere wird auch als Zykluszeit (*engl. makespan, schedule length, project duration bzw. project completion time*) bezeichnet - betrachtet. Die Gesamtbearbeitungszeit wird in der Literatur[5] wie folgt definiert:

Gesamtbearbeitungszeit
Die Gesamtbearbeitungszeit ist diejenige Zeitspanne, die vom Bearbeitungsbeginn des zuerst zu bearbeitenden Arbeitsganges (zum Zeitpunkt 0) bis zur Fertigung des zuletzt zu bearbeitenden Arbeitsganges vergeht.

Im folgenden wird mit einem optimalen Maschinenbelegungsplan bzw. Ablaufplan derjenige Plan bezeichnet, der die minimale Gesamtbearbeitungszeit benötigt.

Bei der Suche nach optimalen Ablaufplänen führen wir folgende qualitative Kriterien[6] ein, um die Zahl der zu betrachtenden, alternativen Ablaufpläne zu reduzieren:

- Ein Ablaufplan ist *zulässig*, wenn die vorgegebenen Präzedenzbeziehungen sowie sonstige vorausgesetzte Nebenbedingungen wie z. B. beschränkte Ressourcen nicht verletzt werden.
- Ein Ablaufplan ist *ein aktiver Ablaufplan*[7] genau dann, wenn
    1. er ein zulässiger Ablaufplan ist und
    2. jeder Arbeitsgang seine beanspruchte Ressource zu dem frühesten Termin belegt, an dem er sie belegen kann.

Die zweite Eigenschaft eines aktiven Ablaufplans verlangt, daß bei zeitlicher Verteilung von Arbeitsgängen bzw. Aktivitäten auf Ressourcen (unter Berücksichtigung der vorgegebenen Präzedenzbeziehungen) alle Arbeitsgänge bzw. Aktivitäten *frühestmöglich* bei den zugehörigen Ressourcen eingeplant werden müssen. Für eine endliche Menge von Aktivitäten ist somit die Menge aller aktiven Ablaufpläne stets endlich. Ferner existiert für das oben genannte Zielkriterium immer ein optimaler Ablaufplan, der aktiv ist. Im Rahmen dieser Arbeit werden *nur aktive Ablaufpläne* betrachtet. Desweiteren werden für alle Problemklassen die folgenden Nebenbedingungen vorausgesetzt:

[4] vgl. [See75], [DSV97]
[5] z. B. [DSV97]
[6] vgl. [KS90], [See75]
[7] vgl. [GT60], [CM62]

- Ressourcen:
  Bei den Produktionsfaktoren (Ressourcen) wird zwischen Verbrauchsfaktoren (Materialfaktoren, Repetierfaktoren) und Gebrauchsfaktoren (Potentialfaktoren) unterschieden. In dieser Arbeit beschränken wir uns auf die Betrachtung von Potentialfaktoren. Potentialfaktoren wie z. B. Maschinen, Arbeitskräfte und Flächen für die Montage werden benötigt, um Arbeitsgänge bzw. Aktivitäten durchzuführen. Wir betrachten Ressourcen ferner als beschränkt. Die Nebenbedingung "beschränkter Ressourcen" hat dabei folgende Semantik:

  > Jede Ressource wird eindeutig mit einem Namen identifiziert und kann in einer Zeiteinheit von maximal einer Aktivität belegt werden. Sie wird erst nach Beendigung der Aktivität wieder freigegeben (Diese Ressourcenbeschränkung kann Ressourcenkonflikte verursachen, die eine parallele Bearbeitung mehrerer Aktivitäten verhindern).

- Ein durchzuführender Arbeitsgang (bzw. Aktivität) kann nicht unterbrochen werden (*engl. nonpreemptive case*).

Im folgenden wird aufgezeigt, wie die einzelnen Problemklassen formal dargestellt, bzw. definiert werden können.

2.1.1 Die Maschinenbelegungsplanung (Job Shop Scheduling)

Im Rahmen eines Job Shop-Problems sind n Aufträge auf m Maschinen, die von unterschiedlichen Typen sind, zu bearbeiten. Jeder Auftrag besteht aus höchstens m Arbeitsgängen (Operationen), die in einer *fest vorgegebenen Reihenfolge* zu bearbeiten sind. Um einen Arbeitsgang zu bearbeiten, wird eine Maschine für eine bestimmte zeitliche Dauer benötigt. Die Zuordnung von einem Arbeitsgang zu einer Maschine sowie die zeitliche Dauer ist hierbei bekannt, d.h. die Maschinenfolge sowie die mit der jeweiligen Maschine verbundenen Bearbeitungszeit der einzelnen Aufträge sind eindeutig festgelegt (siehe Beispiel 2.1.1). Zu einem Zeitpunkt kann maximal ein Auftrag auf einer Maschine bearbeitet werden. Wenn mehrere Arbeitsgänge aus verschiedenen Aufträgen gleichzeitig dieselbe Maschine beanspruchen, muß eine Reihenfolge, in der die konkurrierenden Arbeitsgänge nacheinander zu bearbeiten sind, bestimmt werden. Aufgabe der Maschinenbelegungsplanung ist es nun, den Bearbeitungszeitraum jedes Arbeitsganges auf der zugeordneten Maschine so festzulegen, daß ein optimaler Maschinenbelegungsplan entsteht.

Auftrag	Maschinenfolge Bearbeitungszeit
1	c a d b 1 2 5 3
2	a b c d 3 2 1 2
3	a d c 1 5 1

Tabelle 2.1: Beispiel zur Maschinenbelegungsplanung

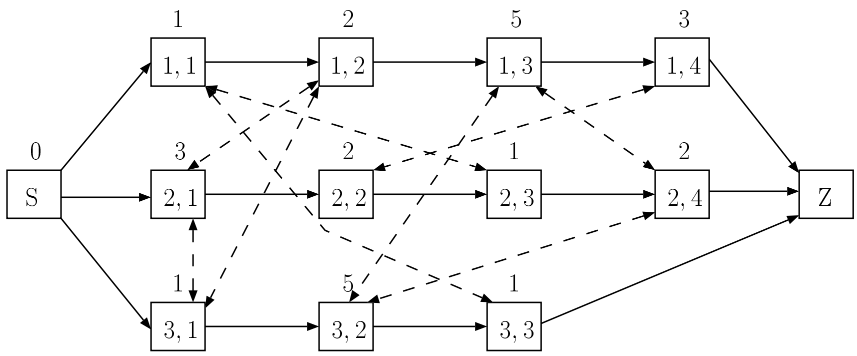


Abbildung 2.2: Disjunktiver Graph

Im folgenden wird das Maschinenbelegungsproblem mittels eines Beispiels näher erläutert.

Beispiel 2.1.1. 3 Aufträge $\{1, 2, 3\}$ seien auf 4 Maschinen $\{a, b, c, d\}$ zu durchlaufen. Die Aufträge 1 und 2 bestehen aus 4 Arbeitsgängen. Der Auftrag 3 besteht aus 3 Arbeitsgängen. Die Zuordnung von Arbeitsgängen zu Maschinen sei wie in Tabelle 2.1 gezeigt vorgegeben. Jedem Arbeitsgang wird eine Bearbeitungszeit zugeordnet.

Der disjunktive Graph[8] aus Abbildung 2.2 stellt dieses Maschinenbelegungsproblem dar.

Ein Maschinenbelegungsplan kann mittels eines Gantt-Diagramms [Cla35] dargestellt werden. Hierbei werden die Bearbeitungszeiten über der Abszisse (Zeitachse) sowie die Maschinen (Maschinenorientiertes Gantt-Diagramm) bzw. alternativ die Aufträge (Auftragsorientiertes Gantt-Diagramm) über der Ordinate aufgetragen.

Die beiden Gantt-Diagramme aus Abbildungen 2.3 und 2.4 zeigen einen optimalen Maschinenbelegungsplan für obiges Beispiel. Wie zu ersehen ist, beträgt

[8] siehe Abschnitt 2.2

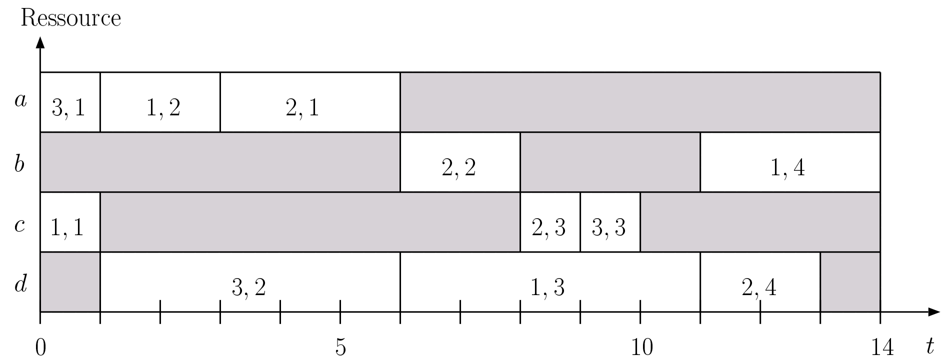


Abbildung 2.3: Maschinenorientiertes Gantt-Diagramm

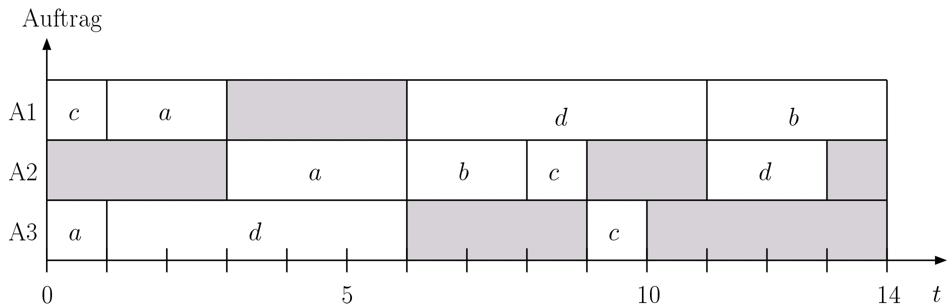


Abbildung 2.4: Auftragsorientiertes Gantt-Diagramm

die minimale Gesamtbearbeitungszeit 14 Zeiteinheiten. Abbildung 2.3 zeigt dabei ein maschinenorientiertes Gantt-Diagramm, wohingegen Abbildung 2.4 ein auftragsorientiertes Gantt-Diagramm zeigt.

Aus dem maschinenorientierten Gantt-Diagramm kann für jede Maschine eine Auftragsfolge abgelesen werden. Eine Auftragsfolge beschreibt eine Reihenfolge, in der Aufträge auf einer Maschine bearbeitet werden.
Das auftragsorientierte Gantt-Diagramm gibt für jeden Auftrag die *vorgegebene* Maschinenfolge wieder. Diese kann alternativ auch aus Tabelle 2.1 abgelesen werden.

Flow Shop
Wie bereits erwähnt, wird bei einem Maschinenbelegungsproblem für jeden Auftrag eine Maschinenfolge vorgegeben. Im folgenden wird eine echte Subklasse der Klasse der Maschinenbelegungsprobleme vorgestellt, die *Reihenfertigungsplanung* (Flow Shop Scheduling) genannt wird. Die Reihenfertigungsplanung erfüllt die folgende zusätzliche Eigenschaft:

- Bei einem Flow Shop-Problem wird für alle Aufträge dieselbe Maschinenfolge, in der sie zu bearbeiten sind, fest vorgegeben.

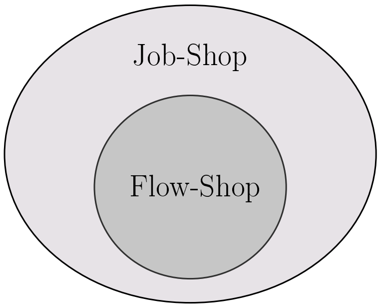


Abbildung 2.5: Subklassenbeziehung zwischen Job-Shop und Flow-Shop

Auftrag	Maschinenfolge Bearbeitungszeit
1	a b 4 2
2	a b 4 6
3	a b 2 3
4	a b 3 1
5	a b 6 4
6	a b 2 4

Tabelle 2.2: Beispiel zur Reihenfertigungsplanung

Abbildung 2.5 zeigt die Subklassenbeziehung zwischen den Problemklassen Job-Shop-Scheduling und Flow-Shop-Scheduling.

Als ein Beispiel betrachten wir 6 Aufträge und 2 Maschinen a und b[9] Hierbei sei jeder Auftrag zuerst von a und dann von b durchzuführen. Das maschinenorientierte Gantt-Diagramm aus Abbildung 2.6 gibt einen zugehörigen optimalen Plan wieder.

Wenn n Aufträge und m Maschinen gegeben sind, erhalten wir insgesamt $(n!)^m$ aktive Pläne als obere Schranke. Hierbei beschreibt $n!$ die Anzahl aller Anordnungsmöglichkeiten der (maximal) n Aufträge bei einer Maschine. Im schlimmsten Fall können n Aufträge um eine Maschine konkurrieren.
Desweiteren kann ein Auftrag maximal aus m Arbeitsgängen bestehen. Anders ausgedrückt ist jeder Auftrag auf maximal m Maschinen zu durchlaufen. Für jede der m Maschinen existieren $n!$ verschiedene Anordnungsmöglichkeiten als

[9] vgl. Coffman [CJ76]

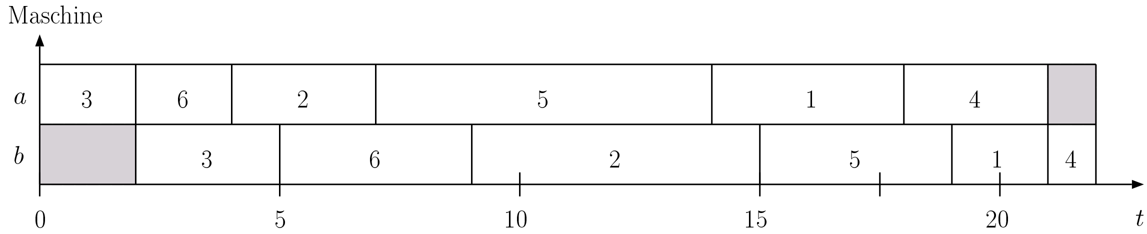


Abbildung 2.6: Ein optimaler Flow-Shop-Schedule (22 Zeiteinheiten)

obere Schranke. Die Anordnungen können maschinen-unabhängig festgelegt werden. Somit erhalten wir $(n!)^m$ als obere Schranke für die Anzahl der Pläne.

Bei der Untersuchung des Job Shop-Problems wurde mit speziellen Problemstellungen begonnen, die Zahl der Maschinen sowie die Anzahl der Aufträge wurden auf fixe Größen (z. B. 2 bzw. 10) beschränkt.
Eine erste Lösung des Flow-Shop Problems, bei der die Zahl der Maschinen auf maximal 3 beschränkt war, gab Johnson [Joh54]. Zwei Jahre später ermittelte Jackson [Jac56] ein exaktes, *polynomiell beschränktes* (*effizientes*) Lösungsverfahren für das Job Shop-Problem bei einer Beschränkung auf 2 Maschinen. Muth und Thompson [MT63] gaben ein Lösungsverfahren für das Job Shop-Problem bei einer Beschränkung auf 10 Maschinen und 10 Aufträgen.

Manne beschreibt in [Man60] ein Modell zur Darstellung und Lösung des allgemeinen Job Shop-Problems auf der Grundlage der linearen Programmierung. Dieses Verfahren verlangt jedoch bereits für moderate Problemgrößen erheblichen Rechenaufwand[10]. Kistner und Steven [KS90] argumentieren daher, daß das auf der linearen Programmierung basierte Modell zur Maschinenbelegungsplanung lediglich als Erklärungsmodell von Bedeutung ist, aber keine Relevanz für praktische Entscheidungen hat. Weitere Modelle, die auf die Grundlage der linearen Programmierung basiert sind, befinden sich in [Bow59], [Wag59], [PW68], [PWW69].

Ein weiteres Verfahren basiert darauf, daß jede kombinatorisch mögliche Reihenfolge[11] berechnet wird, bei der die Operationen (Arbeitsgänge) unter Berücksichtigung aller Nebenbedingungen angeordnet werden. Hierbei wird versucht, redundante Reihenfolgen sowie Reihenfolgen, die ihre Nichtoptimalität vorher erkennen läßt, frühzeitig zu verwerfen. Ausgewählt werden abschließend diejenigen Reihenfolgen, die nach dem angestrebten Kriterium optimal sind. Ein derartiges Verfahren wird deshalb als ein kombinatorisches Verfahren bezeichnet. Häufig basiert ein kombinatorisches Verfahren im wesentlichen auf den folgenden zwei Schritten, die im Rahmen des Algorithmus rekursiv angewendet werden:

[10] siehe für näheres [KS90], [See75], [MM75],
[11] für näheres siehe Abschnitt 4.1

- Branching (Verzweigung)
  Von einer Knospe [12] aus können mehrere Operationen im nächsten Schritt durchgeführt werden. So können von einer Knospe aus mehrere (neue) Knospen bzw. Blätter erreicht werden, wobei durch diese Verzweigung neue Reihenfolgen erzeugt werden.

- Bounding (Begrenzung)
  Werden alle kombinatorisch möglichen Reihenfolgen verfolgt, wird bereits für moderate Problemgrößen erheblicher Rechenaufwand erforderlich. Von jeder Knospe aus werden daher nur bestimmte Nachfolgeoperationen weiter verfolgt, indem ein Auswahlskriterium angewendet wird.

In der Literatur sind zahlreiche Verfahren beschrieben worden, die auf Branch-and-Bound-Techniken basieren[13]. Balas beschreibt in [Bal69] eine auf der Grundlage des Branch-and-Bound-Verfahrens basierte Problemlösung unter Zuhilfenahme von disjunktiven Graphen[14]. Er stellt ein Job-Shop Problem als ein disjunktiver Graph dar und versucht, einen mini-maximalen Weg in dem Graph zu bestimmen[15]. Weitere exakte Lösungsalgorithmen, die auf ähnliche Methoden basiert sind, ermitteln Lageweg, Lenstra und Rinnooy Kan [LLRK77] sowie Carlier und Pinson [CP89]. Desweiteren führt Carlier und Pinson [CP89] einen Korrektheitsbeweis für ihr Verfahren durch. Diese Lösungsansätze sind aber sehr komplex aufgebaut, so daß sie nur sehr schwer verstanden werden können.

Heuristische Lösungsverfahren[16] für das Job Shop-Problem basieren häufig auf der Grundlage von Prioritätsregeln. Den erheblichen Rechenaufwand, der zur Lösung des Job Shop-Problems erforderlich ist, verursachen die beiden zu berücksichtigenden Nebenbedingungen, Präzedenzbeziehungen und beschränkte Ressourcen. Bei der Lösung eines Job Shop-Problems entstehen häufig Situationen, wo mehrere Arbeitsgänge um eine Maschine konkurrieren. Eine Prioritätsregel beschreibt ein Auswahlkriterium, nach dem einer der konkurrierenden Arbeitsgänge ausgewählt wird. Dies verhindert, daß die Anzahl der verfolgten Reihenfolgen exponentiell wächst. Die Prioritätsregeln werden in Hinblick auf eine bzw.

[12] Der Erzeugungsprozeß aller im Rahmen eines Maschinenbelegungsproblems betrachteten Reihenfolgen kann durch einen verzweigenden Baum dargestellt werden (siehe Abschnitt 4.1). Ein Knoten stellt in einem solchen Baum eine Operation dar. Ein Knoten, dessen Nachfolger noch nicht untersucht wurden, heißt Knospe.

[13] vgl. [Gre68], [SD67], [Bal69], [CD70], [AH73], [Tow77], [Car82], [CP90], [WR90], [BJS91], [BJK92], [BJ93], [CP94]

[14] siehe Abschnitt 2.2

[15] Es ist bakannt, daß ein Maschinenbelegungsproblem durch einen disjunktiven Graph modelliert werden kann und die Länge des mini-maximalen Weg in dem Graph der optimalen zeitlichen Dauer entspricht [RS64], [Bal67], [Bal67].

[16] vgl. [Row59], [Row60], [CJM60], [HL62], [Alb65], [BW65], [Con65b], [Con65a], [Ger66], [DH70], [Buz71], [Jon73], [New75], [PI77], [BPH82], [Küp82], [OH85], [Hau89], [Cor90], [HC90], [Dul93], [WW95]

mehrere Zielsetzungen formuliert. Eine Prioritätsregel ist beispielsweise die "kürzeste Operationszeit-Regel", bei der derjenige Auftrag die höchste Priorität erhält, dessen Bearbeitungszeit auf der Maschine am kürzesten ist. Hierbei werden die Ziele einer guten Kapazitätsauslastung und kurzer Durchlaufzeiten erreicht, da dadurch die Maschine schnell wieder für andere Arbeitsschritte zur Verfügung steht[17].

Im Rahmen der von uns entwickelten terminologischen Repräsentationssysteme kann ein beliebiges Job Shop-Problem dargestellt und gelöst werden. In Abschnitt 4.7 wird ein herkömmliches Modell "Modell von Manne [Man60]" einem der in dieser Arbeit vorgestellten Modelle gegenübergestellt.

2.1.2 Die Projektplanung ($\mathcal{RCPS}$ = resource constrained project scheduling)

Seit der Veröffentlichung der Pionierarbeiten von Kelly [Kel63] und Wiest [Wie63] haben zahlreiche Wissenschaftler versucht, das $\mathcal{RCPS}$-Problem darzustellen und zu lösen. Die einzelnen dabei entwickelten Ansätze können nach verschiedenen Kriterien geordnet werden. Ein Kriterium bilden dabei die Eigenschaften der zu optimierenden Zielfunktion. Der am häufigsten verfolgte Gegenstand der Zielfunktion entspricht der Minimierung *der Gesamtbearbeitungszeit* (*engl.* makespan, schedule length, project duration bzw. project completion time) ([BMS64], [Wie67], [PWW69], [Fis70], [DH71], [PH74], [TP78], [HNS79], [Tal82], [Boc90]). Ein weiterer Gegenstand ist z. B. die Minimierung *der gesamten Projektkosten* ([Slo80], [Slo81], [Tal82]).
Der *unterbrechbare (engl. preemptive)* bzw. *nichtunterbrechbare (engl. not preemptive)* Fall[18] bildet ein weiteres Kriterium. Patton [Pat68], Schrage [Sch72], Weglarz et al. [WBCS77], [WBCS80] und Slowinski [Slo80], [Slo81] untersuchen den preemtiven Fall. Die Publikationen ([Sch70], [DH71], [MM67], [Joh67], [PH74], [Sti75], [PR76], [TP78], [Tal82], [Boc90], [HNS79], [OB94]) behandeln den nichtpreemtiven Fall.
Weglarz et al. [WBCS79] und Slowinski [Slo80], [Slo81] führen 3 verschiedene Ressourcenkategorien ein, die Ressourcen gemäß ihrer Eigenschaften klassifizieren. *Erneuerbare Ressourcen*[19] sind diejenigen, die in beschränkter Menge in jeder Periode verfügbar und von Periode zu Periode erneuerbar sind. Ein Beispiel für eine erneuerbare Ressource ist ein Labor. Wenn die gesamte Menge einer Ressource für die gesamte Projektdauer beschränkt ist, heißt eine Ressource *nichterneuerbare Ressource*[20]. Ein Beispiel für eine nichterneuerbare Ressource ist Geld. In

[17] vgl. [KS90]
[18] Wie bereits erwähnt, kann beim nichtpreemtiven Fall eine durchzuführende Aktivität nicht unterbrochen werden.
[19] Diese Ressourcen sind nicht anders als Gebrauchsfaktoren.
[20] Nichterneuerbare Ressourcen sind gleichbedeutend wie Verbrauchsfaktoren.

[WBCS79], [Slo80] und [Slo81] werden *doppelt beschränkte Ressourcen* betrachtet. Bei doppelt beschränkten Ressourcen ist die verfügbare Menge während der gesamten Projektdauer sowie die verfügbare Menge pro Periode beschränkt. Geld ist ein Beispiel für eine doppelt beschränkte Ressource. Meistens wird jedoch in der Literatur der Fall erneuerbarer Ressourcen betrachtet.
Desweiteren bildet Holloway *et al.* [HNS79] eine Klassifikation nach folgenden Kriterien:

- die Anzahl der Ressourcentypen,
- die verfügbare Anzahl der Ressourcen für einen Ressourcentyp und
- die Anzahl der Ressourcentypen, die durch eine Aktivität beansprucht werden kann.

Nach Holloway's Klassifikation beschäftigen sich Davis und Heidorn [DH71], Davis und Patterson [DP75], Patterson und Huber [PH74], Talbot [Tal82] sowie Oguz und Bala [OB94] mit dem $\mathcal{RCPS}$-Problem der Kategorie $n/n/n$. Muth und Thompson [MT63], Müller-Mehrbach [MM67], Johnson [Joh67], Conway *et al.* [CMM67], Patton [Pat68], Schrage [Sch70] und Holloway *et al.* [HNS79]behandeln das $\mathcal{RCPS}$-Problem der Kategorie $n/1/1$, die wir auch in Kapitel 7 dieser Arbeit betrachten werden.

Im folgenden wird eine Version des $\mathcal{RCPS}$-Problems der Kategorie $n/1/1$ formal beschrieben[21].

- Gegeben seien eine endliche Menge R von Ressourcen und ein Netzplan (A, P) mit den folgenden Eigenschaften:
  - $A = \{1, 2, \cdots, n\}$ ist eine endliche Menge von elementaren Aktivitäten (Grundaktivitäten).
  - Jeder Grundaktivität i ist mit *einer* bestimmten Ressource $r(i) \in R$ und *einer* bestimmten Zeit $d(i) > 0$ verbunden. Um die Grundaktivität durchzuführen, wird die Ressource $r(i)$ für die zeitliche Dauer $d(i)$ benötigt.
  - $P = \{(i, j) \,|\, i, j \in A,\ j \text{ ist ein direkter Nachfolger von } i\}$ ist eine zyklusfreie Menge von Tupeln, die Präzedenzbeziehungen zwischen Grundaktivitäten beschreiben. j kann erst nach Abarbeitung von i durchgeführt werden. Die Zyklusfreiheit der Menge P bedeutet, daß bei P keine Folge $(1_1, i_2), (i_2, i_3), \cdots, (i_n, i_1)$ existiert.

[21] vgl. [Sch70]

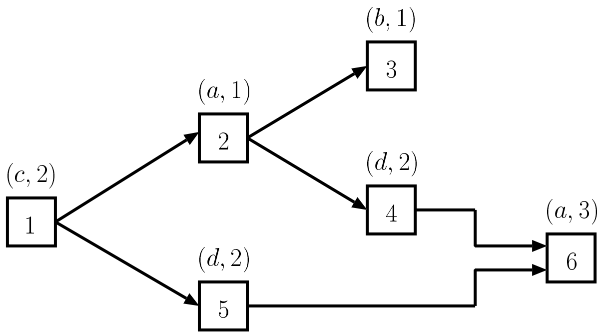


Abbildung 2.7: Netzplan eines $\mathcal{RCPS}$-Problems

- Eine Startzeit $s(i) \geq 0$ für jede Grundaktivität i aus A ist zu bestimmen, so daß ein *optimaler* Ablaufplan ermittelt werden kann. Hierbei müssen die folgenden *Nebenbedingungen* berücksichtigt werden:

  1. Präzedenzbeschränkung: $s(j) \geq s(i) + d(i)$ für alle $(i, j) \in P$.
  2. Ressourcenbeschränkung: $s(m) \geq s(l) + d(l)$ oder $s(l) \geq s(m) + d(m)$ für alle $l, m \in A$ mit $r(l) = r(m)$.

Im folgenden wird ein Beispiel des $\mathcal{RCPS}$-Problems der Kategorie $n/1/1$ vorgestellt.

Beispiel 2.1.2. Gegeben seien 6 Grundaktivitäten $A = \{1, 2, \cdots, 6\}$ sowie 4 Ressourcen $R = \{a, b, c, d\}$. Die Menge der Präzedenzbeziehungen, die zyklusfrei sein soll, laute $P = \{(1, 2), (1, 5), (2, 3), (2, 4), (4, 6), (5, 6)\}$. Die benötigte Ressource sowie zeitliche Dauer der Grundaktivitäten seien wie folgt gegeben:

$$\begin{array}{ll} r(1) = c & t(1) = 2 \\ r(2) = a & t(2) = 1 \\ r(3) = b & t(3) = 1 \\ r(4) = d & t(4) = 2 \\ r(5) = d & t(5) = 2 \\ r(6) = a & t(6) = 3 \end{array}$$

Zu bestimmen ist unter Berücksichtigung der oben aufgeführten Nebenbedingungen eine Startzeit $s(i) \geq 0$ für jede Grundaktivität i aus A, so daß ein optimaler Ablaufplan ermittelt werden kann.
Den zugehörigen Netzplan, aus dem die Ablaufstruktur dieses $\mathcal{RCPS}$-Problems abgelesen werden kann, zeigt Abbildung 2.7.

Startzeiten für Grundaktivitäten, die einen optimalen Ablaufplan dieses Problems ermitteln, lauten $s(1) = 0, s(2) = 2, s(3) = 3, s(4) = 4, s(5) = 2, s(6) = 6$. Den

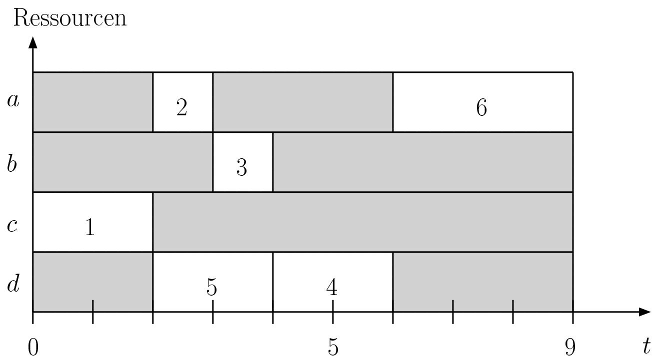


Abbildung 2.8: Ein optimaler Ablaufplan

entsprechenden optimalen Ablaufplan gibt das Gantt-Diagramm aus Abbildung 2.11 wieder.

Wie beim Job Shop-Scheduling können auch die Lösungsmethoden des $\mathcal{RCPS}$-Problems in folgende zwei Gruppen geteilt werden:

1. exakte Lösungsverfahren, bei denen optimale Lösungen berechnet werden.

2. heuristische Lösungsverfahren, die mit geringem Rechenaufwand nicht zwingend zur optimalen, aber zu einer guten Lösung führen.

Frühzeitig wurde bei der Formulierung von exakten Modellen der Projektplanung die Methode der linearen Programmierung verwendet. Aber wie bereits bei der Betrachtung der 'Maschinenbelegungsplanung' aufgeführt, verlangen diese Verfahren erheblichen Rechenaufwand bereits für moderate Problemgrößen [Bal65], [BMS64], [MM66]. Daher ist für die Lösung des $\mathcal{RCPS}$-Problems diese Methode unpraktikabel [PH74], [Hub71], [BMS64], [MM66].
Als eine Alternative ermitteln Patterson und Huber [PH74] ein exaktes Verfahren, das auf 0–1-linearer Programmierung basiert und zusätzlich Bounding-Techniken verwendet. Die verwendeten Bounding-Techniken ähneln dabei den früher beschriebenen Branching-and-Bounding-Techniken.

Schrage [Sch70] führt Beweise für 4 Ansätze, die Grundlagen der Entwicklung eines Branching-and-Bounding-Verfahrens bilden. Mittels dieses Verfahrens können *alle aktiven, nichtredundanten* Ablaufpläne ermittelt werden. Die Erzeugung aller aktiven, nichtredundanten Ablaufpläne kann durch einen Verzweigungsbaum dargestellt werden. Jeder Pfad von der Wurzel des Verzweigungsbaums, dem Knoten ohne Vorgänger, bis zu seinen Blättern, den Knoten ohne Nachfolger, entspricht einer Reihenfolge (einer Permutation), aus der ein aktiver Ablaufplan hergeleitet werden kann. Eine Reihenfolge, die einen aktiven Ablaufplan *eindeutig* repräsentiert, entspricht hierbei einer *lexikografisch geordneten* Permutation der Tupel $(s(1), 1), \cdots, (s(i), i), \cdots (s(n), n)$, wobei i einer Grundaktivität und $s(i)$ die Startzeit der Grundaktivität i entspricht[22].

Weitere exakte Lösungsverfahren, die auf der Branching-and-Bounding-Technik basieren, finden sich in [Joh67], [MM67], [Dav69], [Bal70], [DH71], [Gor72], [Fis73], [EH77], [SDK78], [CAVT87], [BP90] und [DH92].

Heuristische Verfahren zur Lösung des $\mathcal{RCPS}$-Problems finden sich in [Wie67], [DP75], [Coo76], [HNS77], [HNS79], [Tal82], [EN86], [Boc90], [BH91], [KC91], [OB94]. Dort können auch Vergleiche zwischen verschiedenen heuristischen Verfahren und exakten Verfahren gefunden werden.

2.1.3 Ablaufplanung mit varianten Teilabläufen ($\mathcal{RCPS}V$ = $\mathcal{RCPS}$ with variants)

Beim $\mathcal{RCPS}$-Problem wird vorausgesetzt, daß jeder Vorgang (Grundaktivität) durch eine einzige Kombination von einer Ressource und der zugehörigen zeitlichen Dauer erfüllt werden kann. In der Praxis entsteht jedoch häufig der Fall, daß für eine Aufgabe (ein Vorgang bzw. ein Vorgangsnetz) mehrere alternative Handlungsvarianten, die dieselbe Aufgabe erfüllen, zur Auswahl existieren. Ein Planungsmodell, im dessen Rahmen nicht nur die entstehenden Ressourcenkonflikte, sondern auch eine Auswahlsmöglichkeit unter alternativen Handlungsvarianten berücksichtigt werden kann, erfaßt das $\mathcal{RCPS}V$-Problem.
Wie bei $\mathcal{RCPS}$ ist das Ziel des $\mathcal{RCPS}V$-Problems die zeitlich optimale Zuordnung von Aktivitäten zu Ressourcen, d.h. ein zeitlich optimaler Ablaufplan des gesamten Projektes ist zu bestimmen.

Obwohl bereits Schrage [Sch70] die Erweiterungsmöglichkeit vom $\mathcal{RCPS}$-Problem zu der Problemklasse, bei der auch eine Auswahl *unter alternativen Grundaktivitäten* berücksichtigt werden kann, angedeutet hat, gibt es heute vergleichweise wenig Publikationen [Elm77], [Tal82], [Rit98], die sich mit einer erweiterten Problemklasse des $\mathcal{RCPS}$-Problems beschäftigen.

Talbot [Tal82] gibt ein Modell, mit dessen Hilfe Probleme, bei denen alternative Aufgabenträger der Grundaktivitäten berücksichtigt werden können, dargestellt und heuristisch gelöst werden können. Er nennt eine zur Erfüllung einer Grundaktivität auswählbare Kombination von einer Ressource und der zugehörigen zeitlichen Dauer "*job operating mode*" bzw. "*mode*". Eine Grundaktivität i kann beispielsweise in 5 Zeiteinheiten durch eine geschickte Person (mode 1) oder in 7 Zeiteinheiten durch eine ungeschickte Person (mode 2) erfüllt werden. Für jede Grundaktivität stehen eine bzw. mehrere modes zur Auswahl.

[22] für näheres siehe Abschnitt 4.1

Talbot [Tal82] stellt dieses erweiterte Problem des $\mathcal{RCPS}$-Problems als ein lineares Programmierungsproblem (*engl.* integer programming problem) dar. Für jede Grundaktivität werden $0-1-$ganzzahlige Variablen x_{itm} gebildet, wobei $x_{itm} = 1$ gilt, wenn die Grundaktivität i mittels der mode m in Periode t erfüllt wird; ansonsten, gilt $x_{itm} = 0$. Desweiteren werden für jede Grundaktivität i der früheste Endezeitpunkt F_i und der späteste Endezeitpunkt S_i berechnet[23]. Der Wert t bei x_{itm} wird dann zwischen F_i und S_i, d.h. $F_i \leq t \leq S_i$, beschränkt. Auf diesen Variablen werden, wie in einer linearen Programming üblich ist, Gleichungen aufgebaut, die die Zielfunktionen sowie die einzuhaltenden Nebenbedingungen formulieren.

Rittgen stellt in [Rit98] eine formale Darstellung des $\mathcal{RCPSV}$-Problems vor, die auf algebraische Darstellungsweise basiert ist. Desweiteren deuten Wendt, Rittgen und König [WRK96], [Rit98], [WR93] bei Einführung dieser Problemklasse an, daß das $\mathcal{RCPSV}$-Problem mittels hierarchischer Netzpläne grafisch spezifiziert werden kann. Besonders gibt Rittgen in [Rit98] eine ausführliche Erläuterung über hierarchische Netzpläne, die wir auch in Abschnitt 2.2.3 näher betrachten werden.

In Kapitel 6 wird die logikbasierte Schedulingsprache $\mathcal{RCPSV}$ definiert, die das $\mathcal{RCPSV}$-Problem terminologisch darstellt. Unter Zuhilfenahme dieser Sprache sowie des in Kapitel 7 vorgestellten Lösungsalgorithmus kann das im Anschluß vorgestellte $\mathcal{RCPSV}$-Problem gelöst werden.

Das $\mathcal{RCPSV}$-Problem

Das $\mathcal{RCPSV}$-Problem entspricht der Berechnung *eines optimalen Ablaufplans* für einen gegebenen Ausdruck der Sprache $\mathcal{RCPSV}$.

Nähere Erläuterungen dazu befinden sich in Kapitel 6.

Bevor dieser Abschnitt beendet wird, soll eine Subklasse des $\mathcal{RCPSV}$-Problems bzw. eine erweiterte Problemklasse des $\mathcal{RCPS}$-Problems, bei der (wie bei Modell von Talbot [Tal82]) *nur* alternative Aufgabenträger *der Grundaktivitäten* berücksichtigt werden, formal definiert werden. Diese Problemklasse nennen wir $\mathcal{RCPSV}_{pa}$ (= $\mathcal{RCPS}$ with variant primitive activities). Die Subklassenbeziehung zwischen den bisher vorgestellten Problemklassen zeigt Abbildung 2.9.

- Gegeben sind eine endliche Menge $A = \{1, \cdots, n\}$ von elementaren Aktivitäten (Grundaktivitäten) sowie eine endliche Menge R von Ressourcen.

- Jeder Grundaktivität i wird eine Menge $OP(i) = \{(r(i_k), t(i_k)) | 1 \leq k \leq M_i,\ r(i_k) \in R,\ t(i_k) > 0, M_i \in \mathbb{N}\}$ zugeordnet, wobei $OP(i)$ der Menge aller zur Auswahl stehenden alternativen Kombinationen entspricht, die aus einer Ressource und einer zeitlichen Dauer gebildet werden können. Eine

[23] siehe [LW77]

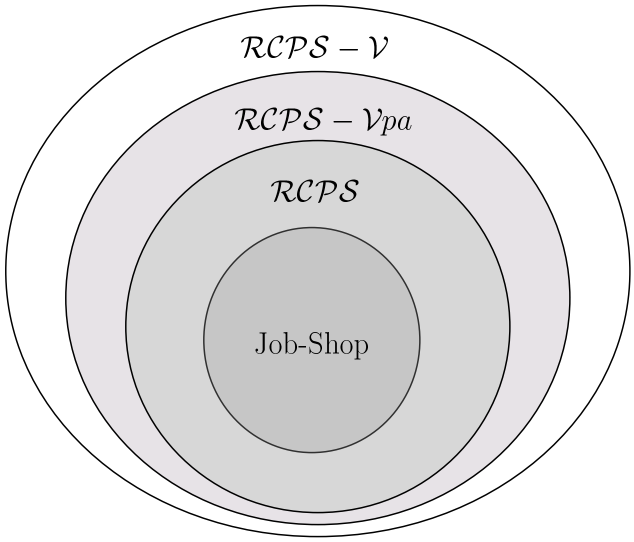


Abbildung 2.9: Problemklassen der Ablaufplanung

Kombination aus der Menge $OP(i)$ wird hierbei zur Erfüllung der Aktivität i ausgewählt bzw. benötigt.

- Gegeben sei eine zyklusfreie Menge $P = \{(i,j)|\, i,j \in A,\, j \text{ ist ein direkter Nachfolger von } i\}$[24] von Tupeln, die Präzedenzbeziehungen zwischen Grundaktivitäten beschreiben. Ein Tupel (i,j) besagt, daß die Aktivität j erst nach Erfüllung der Aktivität i durchgeführt werden kann.

- Für jede Grundaktivität i aus A ist ein Tupel $(r(i_k), t(i_k))$ aus $OP(i)$ auszuwählen, das im folgenden mit $(\hat{r}_i, \hat{t}_i)$ bezeichnet wird. Ferner ist für jede Grundaktivität i eine Startzeit $s(i) \geq 0$ zu bestimmen, so daß ein *optimaler* Ablaufplan ermittelt werden kann. Hierbei müssen die folgenden *Nebenbedingungen* erfüllt werden:

  1. Präzedenzbeschränkung: $s(j) \geq s(i) + \hat{t}_i$ für alle $(i,j) \in P$.
  2. Ressourcenbeschränkung: $s(j) \geq s(i) + \hat{t}_i$ oder $s(i) \geq s(j) + \hat{t}_j$ für alle $i,j \in A$ mit $\hat{r}_i = \hat{r}_j$.

Im folgenden wird ein Beispiel des $\mathcal{RCPS}V_{pa}$-Problems vorgestellt.

[24] Ausführliche Erläuterungen zur Thematik "zyklusfreie Menge" können bei der Definition des $\mathcal{RCPS}$-Problems in Abschnitt 2.1.2 gefunden werden.

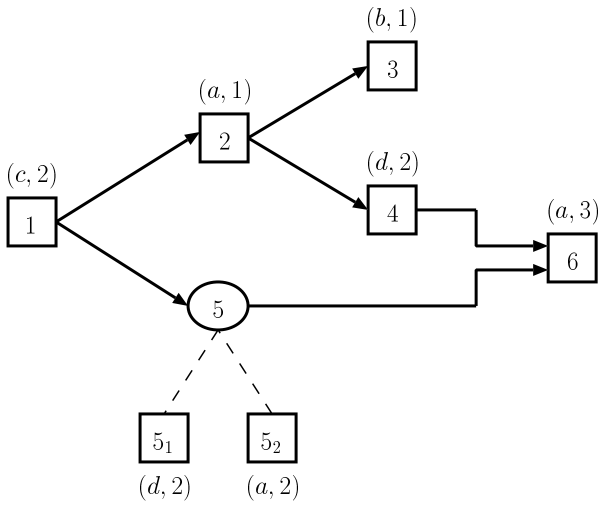


Abbildung 2.10: Der hierarchische Netzplan eines $\mathcal{RCPS}V_{pa}$-Problems

Beispiel 2.1.3. Das zuvor betrachtete Beispiel 2.1.2 wird erweitert, indem für die Grundaktivität 5 eine weitere auswählbare Kombination $(a, 2)$ existiere. Die ursprüngliche Kombination aus Beispiel 2.1.2 wird mit 5_1 bezeichnet, während die letztere mit 5_2 bezeichnet wird. Die Menge der Vorgänge ist somit zu $A = \{1, 2, \cdots, 5_1, 5_2, 6\}$ erweitert worden, die Menge der Ressourcen $R = \{a, b, c, d\}$ jedoch gleich geblieben. Die Menge der Präzedenzbeziehungen $P = \{(1, 2), (1, 5_1), (1, 5_2), (2, 3), (2, 4), (4, 6), (5_1, 6)(5_2, 6)\}$ wird entsprechend angeglichen. Die benötigten Ressourcen sowie zeitliche Dauer der Aktivitäten sind dann wie folgt gegeben:

$$\begin{array}{ll}
r(1) = c & t(1) = 2 \\
r(2) = a & t(2) = 1 \\
r(3) = b & t(3) = 1 \\
r(4) = d & t(4) = 2 \\
r(5_1) = d & t(5_1) = 2 \\
r(5_2) = a & t(5_2) = 2 \\
r(6) = a & t(6) = 3
\end{array}$$

Unter Erfüllung der Nebenbedingungen “beschränkte Verfügbarkeit von Ressourcen und Einhaltung der Präzedenzbeziehungen” ist ein optimaler Ablaufplan zu bestimmen. Den zugehörigen hierarchischen Netzplan, aus dem die Ablaufstruktur dieses $\mathcal{RCPS}V_{pa}$-Problems abgelesen werden kann, zeigt Abbildung 2.10.

Wie in Beispiel 2.1.2 erläutert, beträgt die Gesamtbearbeitungszeit bei der Auswahl von 5_1 minimal 9 Zeiteinheiten. Bei der Auswahl von 5_2 werden, je nachdem, in welcher Reihenfolge die an dem entstehenden Ressourcenkonflikt beteiligten Aktivitäten 2 und 5_2 angeordnet werden, unterschiedliche Gesamtbearbeitungszeit benötigt. Wird 2 vor 5_2abgearbeitet, werden insgesamt 8 Zeiteinheiten

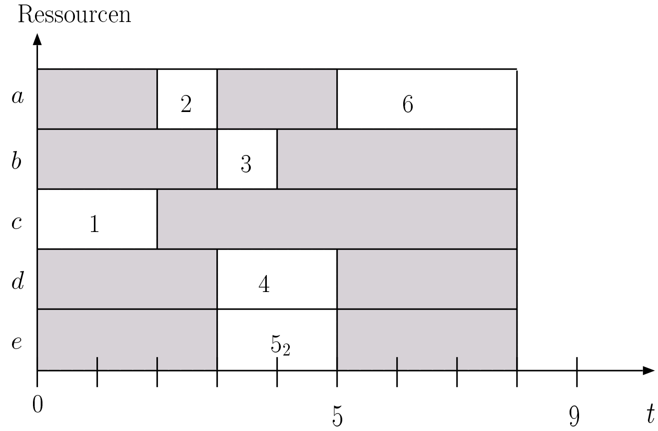


Abbildung 2.11: Ein optimaler Ablaufplan

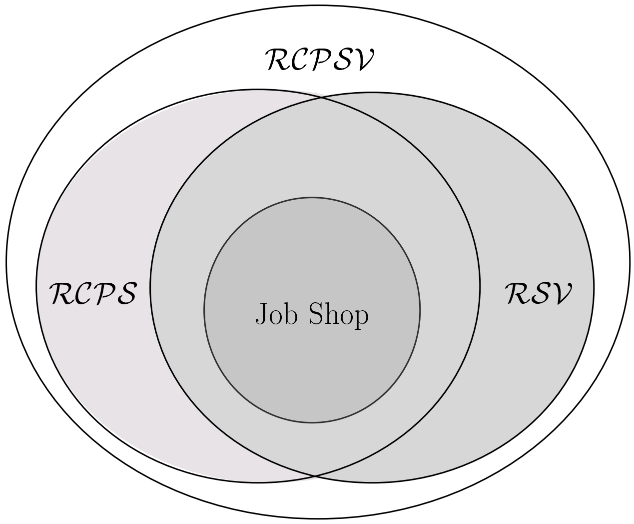


Abbildung 2.12: Sub- bzw. Oberklasse des $\mathcal{RSV}$-Problems

benötigt, während im umgekehrten Fall 10 Zeiteinheiten benötigt werden. Das Gantt-Diagramm aus Abbildung 2.11 stellt den optimalen Ablaufplan mit der 8 Zeiteinheiten dar, wobei 2 vor 5_2 durchgeführt wird. Die zugehörigen Startzeiten der Aktivitäten lauten $s(1) = 0, s(2) = 2, s(3) = 3, s(4) = 3, s(5_2) = 3, s(6) = 5$.

2.1.4 Das $\mathcal{RSV}$-Problem

Nachdem verschiedene Formen des Ablaufplanungsproblems vorgestellt wurden, wird nun eine weitere Klasse $\mathcal{RSV}$ von Ablaufplanungsproblemen eingeführt. Wie bereits erwähnt, kommt die Klasse $\mathcal{RSV}$ bisher nicht in der Literatur vor und wird in dieser Arbeit erstmalig definiert und untersucht. Die Klasse $\mathcal{RSV}$ bildet eine gleichzeitige Oberklasse des Job Shop-Problems und Subklasse des $\mathcal{RCPSV}$-Problems, wobei letztere Aussage gewissen Einschränkungen unterliegt, die später in diesem Abschnitt aufgeführt werden. Abbildung 2.12 zeigt die Klassenbeziehungen in grafischer Form. Das $\mathcal{RSV}$-Problem kann wie folgt charakterisiert werden:

Das $\mathcal{RSV}$-Problem ist auf der logikbasierten Sprache $\mathcal{RSV}$ aufgebaut, die es erlaubt, $\mathcal{RSV}$-Ausdrücke zu spezifizieren. Ein Ausdruck der Sprache $\mathcal{RSV}$ wird mittels dreier Operatoren, die unterschiedliche Planungszusammenhänge wiederspiegeln, auf Basis einer Menge von Grundaktionen aufgebaut.

Das $\mathcal{RSV}$-Problem

Das $\mathcal{RSV}$-Problem besteht darin, für einen gegebenen Ausdruck aus der Sprache $\mathcal{RSV}$ *einen optimalen Ablaufplan* zu berechnen.

Im Rahmen der Untersuchung des $\mathcal{RSV}$-Problems wird ein Diagramm-basiertes Berechnungsmodell vorgestellt werden, das eine elegante und leicht verständliche Problemlösung ermöglicht. Eine ausführliche Untersuchung des $\mathcal{RSV}$-Problems enthält Kapitel 3. Das zuletzt erwähnte Berechnungsmodell wird in Kapitel 4 vorgestellt.
Ein weiteres wesentliches Ergebnis dieser Arbeit wird in Kapitel 8 vorgestellt. In diesem Kapitel wird die Beziehung zwischen den beiden Sprachen (Problemen) $\mathcal{RSV}$ und $\mathcal{RCPSV}$ genau untersucht und gezeigt, daß beide Sprachen über *dieselbe Ausdruckskraft* verfügen, obwohl sie *unterschiedliche Repräsentationsmechanismen* benutzen. Die Identität der Ausdruckskraft ist gleichbedeutend damit, daß beide Sprachen gleich mächtig sind.

2.2 Graphen und Ablaufplanung

In diesem Abschnitt werden verschiedene Methoden vorgestellt, mit denen Ablaufplanungsprobleme grafisch dargestellt werden können.

2.2.1 Netzplantechnik

In den 50ziger Jahren wurden Methoden der Netzplantechnik entwickelt. Die Netzplantechnik dient zur Steuerung, Planung und Überwachung des Ablaufs von großen Projekten. Ein Projekt wird in eine Vielzahl von Vorgängen (Grundaktivitäten) unterteilt, zwischen denen Präzedenzbeziehungen erfaßt werden. Anschließend wird ein Netzplan erstellt, wobei Vorgänge als Knoten und Präzedenzbeziehungen als gerichtete Pfeile dargestellt werden. Bindet ein gerichteter Pfeil einen Vorgang i und einen weiteren Vorgang j, bedeutet dies, daß j erst dann durchgeführt werden kann, wenn i fertiggestellt ist.

Jedem Vorgang in einem Netzplan können die von ihm gebundenen Ressourcen (Einsatzmittel) sowie die benötigte zeitliche Dauer zugeordnet werden (siehe Beispiel 2.1.2).

> So kann offensichtlich ein $\mathcal{RCPS}$-Problem mittels der Netzplantechnik grafisch (als ein Netzplan) dargestellt werden. Hierbei ist zu beachten, daß im Rahmen des ein $\mathcal{RCPS}$-Problem repräsentierenden Netzplans die Beschränktheit der verfügbaren Ressourcen vorausgesetzt wird. Desweiteren ist ein Netzplan, der ein $\mathcal{RCPS}$-Problem darstellt, *ein gerichteter schlichter azyklischer Graph.*

Bei dem in dieser Arbeit betrachteten Schedulingproblem werden Schleifen sowie Zyklen beinhaltende Netzpläne ausgeschlossen, da die Gesamtbearbeitungszeit eines zyklenbehafteten Netzplans nicht vernünftig spezifizierbar ist (siehe Definition 6.1.1).

Die graphische Darstellung des $\mathcal{RCPS}$-Problems mittels der Netzplantechnik macht das $\mathcal{RCPS}$-Problem sehr *transparent*, da die Ablaufstruktur, d.h. die zeitliche Aufeinanderfolge von Aktivitäten sowie die Verteilung der von Aktivitäten beanspruchten Ressourcen deutlich abgelesen werden können.

Wir wollen nicht tiefer in diese Thematik eingehen. Stattdessen weisen wir auf die sich mit der Netzplantechnik beschäftigenden Literaturen hin (Siehe z. B. [Ohs98], [Sch94], [Zim97], [Alt96], [BB74], [HL97], [Joh74][DD98], [Gal92], [EBL98], [NM93]).

2.2.2 Disjunktiver Graph

Die Arbeitsgänge, die bei einem Job Shop-Problem dieselbe Ressource beanspruchen, müssen angeordnet werden. Da ein $\mathcal{RCPS}$-Problem mittels eines Netzplans dargestellt werden kann, ist trivialerweise ein Job Shop-Problem mittels eines Netzplans darstellbar. Um die anzuordnenden Arbeitsgänge eines Job Shop-Problems in einem Netzplan grafisch zu kennzeichnen, führte Balas [Bal69] die sogenannten disjunktiven Kanten ein. Solche Arbeitsgänge werden mittels gestrichelter Kanten (disjunktiver Kanten) miteinander verbunden[25]. Wie bereits erwähnt, wird dies dazu genutzt, bei einem derartigen Netzplan einen minimaximalen Weg zu bestimmen, dessen Länge der optimalen zeitlichen Dauer entsprechen soll[26].
Eine disjunktive Kante zwischen zwei anzuordnenden Aktivitäten i und j wird formal mit $[i, j] = \{(i, j), (j, i)\}$ bezeichnet. So wird neben der Menge P aus dem

[25] siehe z. B. Abbildung 2.2
[26] siehe Abschnitt 2.1.1

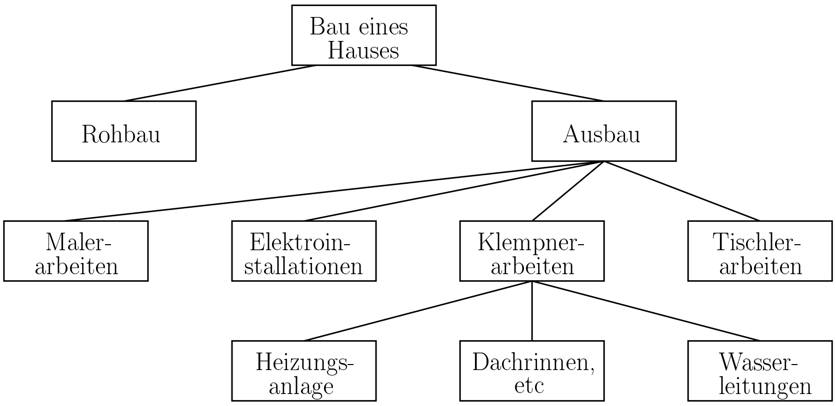


Abbildung 2.13: Ein Projektstrukturplan (Schwarz [Sch94])

Abschnitt 2.1.2, die Präzedenzbeziehungen zwischen Aktivitäten beschreiben, eine weitere Menge D festgehalten. Diese Menge D soll aus solchen disjunktiven Kanten $[i, j] = \{(i, j), (j, i)\}$ bestehen.
Disjunktive Kanten, die für die Projektplanung (Problemklasse $\mathcal{RCPS}$) beschränkt auf der Kategorie $n/1/1$ nach Holloway [HNS79] angewendet werden können, wurden meistens bei Job Shop-Scheduling angewendet. Ein disjunktiver Graph, der einen verfeinerten Netzplan darstellt, wird formal wie folgt definiert:

Definition 2.2.1. [27]Ein disjunktiver Graph $\mathcal{G} = (\mathcal{V}, \mathcal{C}, \mathcal{D})$ *besteht dabei aus*

- *einer Menge von elementaren Aktivitäten bzw. Operationen* $V = \{i \in \mathbb{N} | i = 1, \cdots, n\}$,[28]
- *einer Menge* C *konjunktiver Kanten (d.h. der normalen Präzedenzkanten), d.h.* $(i, j) \in C$, *wenn die Operation* i *vor Operation* j *stattfinden muß, und*
- *einer Menge* disjunktiver Kanten D, *d.h.* $\{(i, j), (j, i)\} \in D$, *wenn entweder* i *vor* j *stattfinden muß oder umgekehrt.*

2.2.3 Hierarchischer Netzplan

Der Strukturplan eines großen Projektes analysiert den Ablauf des Projektes in einem Top-Down-Ansatz. Abbildung 2.13 zeigt ein Beispiel des Strukturplans.

[27] zu finden bei Rittgen [Rit98]

[28] Wie in Abschnitt 2.1.2 beschrieben, ist jede elementare Aktivität (Operation) i mit *einer* bestimmten Ressource $r(i) \in R$ und *einer* bestimmten Zeit $d(i) > 0$ verbunden.

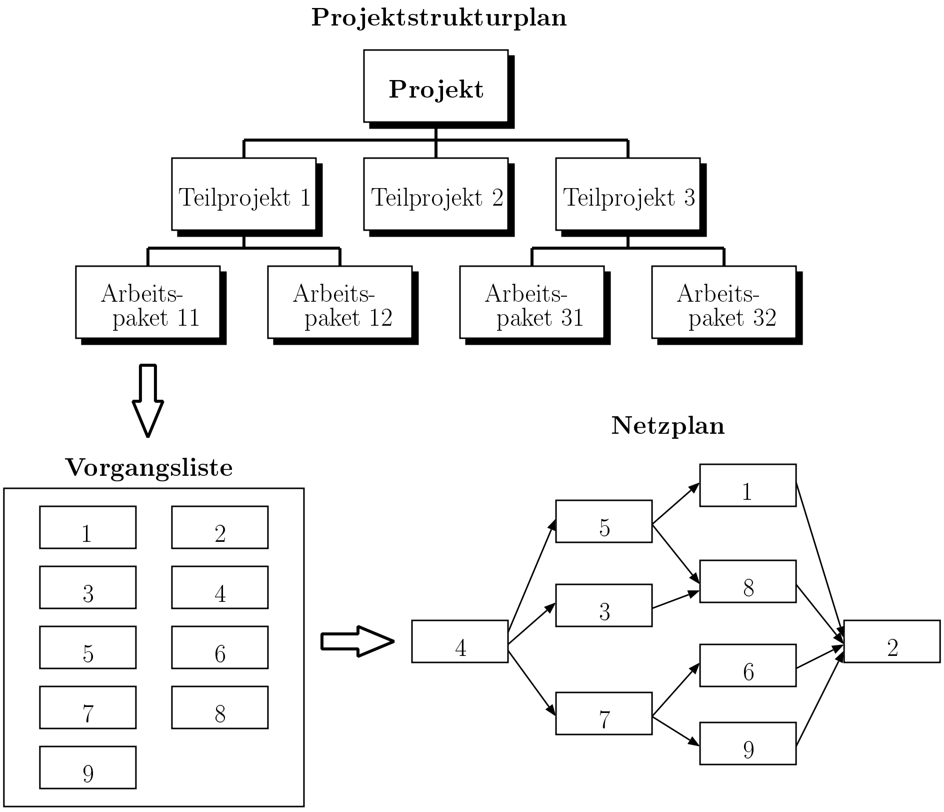


Abbildung 2.14: Zusammenhang zwischen Projektstrukturplan und Netzplan (Schwarz [Sch94])

Zwischen einem Strukturplan und einem Netzplan besteht ein enger Zusammenhang, der aus Abbildung 2.14 entnommen werden kann.
Bei einem Strukturplan handelt sich nur um diejenigen Aktivitäten (Teilprojekte) bzw. Grundaktivitäten (Vorgänge), die im Rahmen eines Projektes erfüllt werden müssen. Die alternativen Handlungsvarianten für einen Teil des Projektes, d.h. ein bestimmtes Teilprojekt kann beispielsweise unterschiedliche Agenten beauftragen bzw. durch Verwendung unterschiedlicher Ressourcen in unterschiedlichen Produktionswegen durchgeführt werden, bleiben hierbei unberücksichtigt.
Ein grafisches Verfahren "Hierarchischer Netzplan", dessen nahestehende Verfahren bereits in [Gen79], [WR93] und [WRK96] vorgestellt wurden, ist eine Erweiterung dieses Projektstrukturplans, wobei nicht nur die alternativen Handlungsvarianten sondern auch die Präzedenzbeziehungen zwischen Teilprojekten berücksichtigt werden können.
Im realen Umfeld wäre beispielsweise denkbar, daß im Rahmen des durch Abbildung 2.14 spezifizierten Projektes für das Arbeitspaket 32 sowie den Vorgang 7 jeweilig zwei alternative Handlungsvarianten existieren. Desweiteren seien das

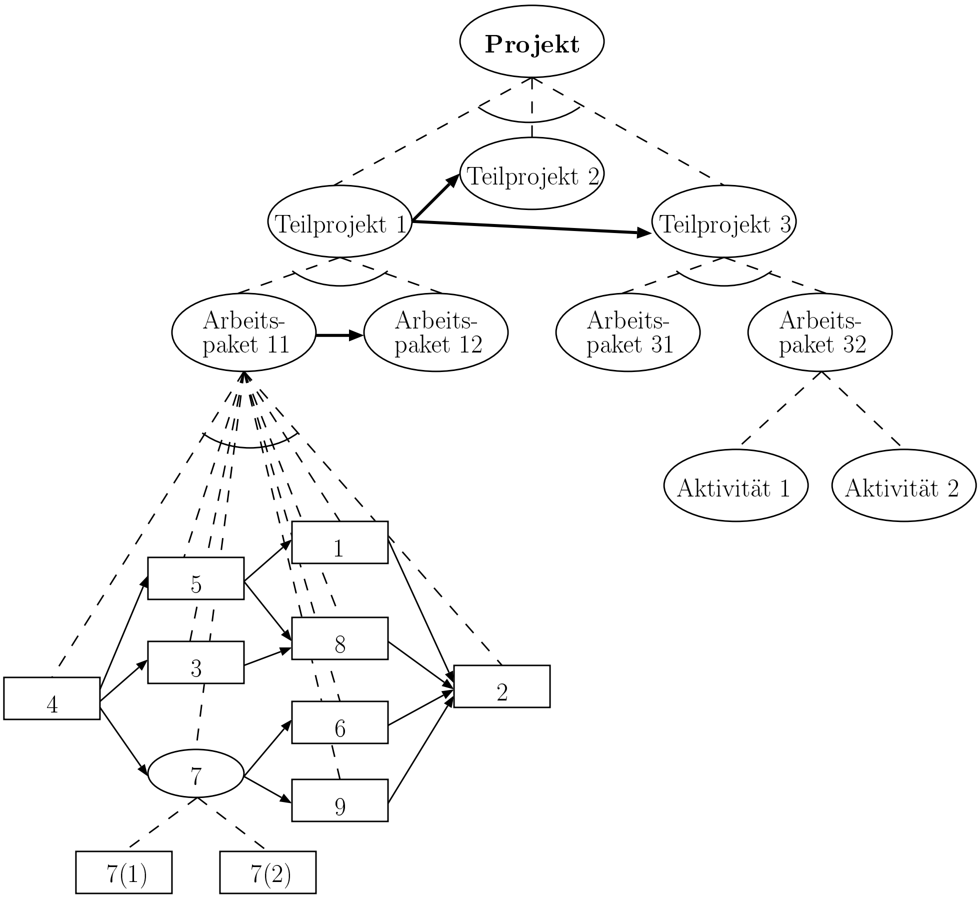


Abbildung 2.15: Ein hierarchischer Netzplan

Teilprojekt 2 sowie das Teilprojekt 3 erst nach der Fertigstellung des Teilprojektes 1 und das Arbeitspaket 12 erst nach der Fertigstellung des Arbeitspakets 11 durchzuführen. Derartige zusätzlichen Beschreibungen können mittels eines hierarchischen Netzplans grafisch dargestellt werden. Den entsprechenden hierarchischen Netzplan gibt Abbildung 2.15 wieder.

Wie bereits in Abschnitt 2.1.3 erwähnt, kann mittels dieses Verfahrens "Hierarchischer Netzplan" das $\mathcal{RCPSV}$-Problem grafisch spezifiziert werden, wobei wir die beschränkte Verfügbarkeit von Ressourcen voraussetzen. Im folgenden wird dieses Verfahren näher betrachtet:

Der Zerlegungsprozeß vom gesamten Projekt in die immer kleinere Aktivitäten (Teilprojekte) sowie die Spezifikation der alternativen Aktivitäten werden durch baumartige Strukturen dargestellt. So baut der hierarchische Netzplan sowohl auf der Netzplantechnik als auch auf dem AND/OR-Graph[29] auf, der auf AND/OR-Bäume basiert ist. Bei einem AND/OR-Baum können zunächst die folgenden

[29] vgl. [Pea84] und [Rit98]

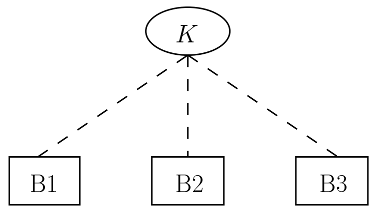


Abbildung 2.16: Ein hierarchischer Netzplan (Knospe K und Blätter)

zwei Arten von Knoten betrachtet werden:

- Knospen, die in weitere Knospen bzw. Blätter zerlegt werden können. Diese sollen *Aktivitäten* beschreiben, die in weitere Teilaktivitäten bzw. Grundaktivitäten zerlegt werden. Grafisch werden diese durch *runde* bzw. *ovale Knoten* dargestellt.

- Blätter, die nicht weiter zerlegt werden können. Diese beschreiben *Grundaktivitäten.* Grafisch werden diese durch rechteckige Knoten dargestellt (siehe Abbildung 2.16).

Die folgenden weiteren Eigenschaften werden in einem AND/OR-Baum mittels zwei unterschiedlicher Arten von Kanten dargestellt:

- AND-Kanten: Alle direkt von derselben Aktivität (z. B. $K1$ in Abbildung 2.17) stammenden Kanten werden mit einem Bogen verbunden. Diese AND-Kanten beschreiben, daß alle Söhnen (Teilaktivitäten z. B. $K2$ und B4 in Abbildung 2.17) durchgeführt werden müssen, um die Aktivität (z. B. $K1$ in Abbildung 2.17) zu erfüllen.

- OR-Kanten: Alle direkt von derselben Aktivität stammenden Kanten stehen einzeln da (nicht mit einem Bogen verbunden z. B. siehe $K2$ aus Abbildung 2.17). Dies beschreibt die Auswahlmöglichkeit. Um die Aktivität (z. B. $K2$ aus Abbildung 2.17 bzw. Arbeitspaket 32 aus Abbildung 2.15) zu erfüllen, wird *genau* eine der Kanten ausgewählt und die zugehörige (Teil-)Aktivität bzw. Grundaktivität durchgeführt. Alle zu OR-Kanten gehörigen (Teil-)Aktivitäten entsprechen unterschiedlichen Handlungsvarianten und erfüllen daher alternativ dieselbe Aufgabe (Um bei Abbildung 2.17 die Aktivität $K2$ zu erfüllen, ist eine der 3 Grundaktivitäten B1, B2 und B3 auszuwählen und durchzuführen.).

Angenommen, ein Projekt P bestehe aus zwei zu erfüllenden Aktivitäten (Teilprojekten) $A1$ und $A2$, wobei zwischen $A1$ und $A2$ keine Präzedenzbeziehung

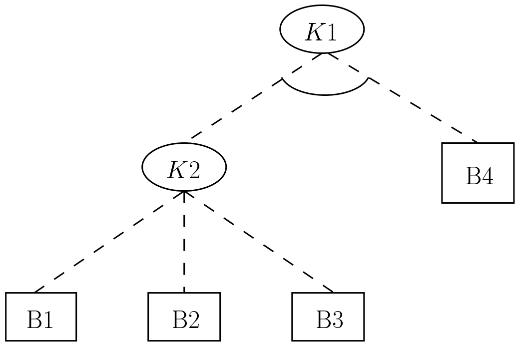


Abbildung 2.17: Ein hierarchischer Netzplan

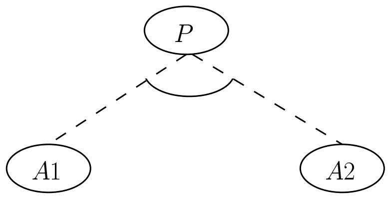


Abbildung 2.18: Teil eines hierarchischen Netzplans

besteht. Dies zeigt Abbildung 2.18. Desweiteren wird angenommen, daß die Aktivität $A1$ mittels des Netzplans aus Abbildung 2.19 beschrieben werden kann. Die Transformation von einem Netzplan zu dem zugehörigen hierarchischen Netzplan zeigt die Abbildung 2.20.

$A2$ bestehe aus drei Grundaktivitäten, die sequentiell durchgeführt werden müssen. Der Netzplan aus Abbildung 2.21 gibt dann $A2$ wieder. Die entsprechende Transformation von diesem Netzplan zum zugehörigen hierarchischen Netzplan zeigt Abbildung 2.22.

Desweiteren nehmen wir hierbei an, daß für die Grundaktivität $G6$ zwei alternative Auswahlvarianten existieren. Diese bezeichnen wir als $G61$ und $G62$. Dann sieht der zugehörige hierarchische Netzplan wie Abbildung 2.23 zeigt aus. Hierbei ist der zu $G6$ gehörige Knoten durch einen ovalen Knoten ausgetauscht, da dieser

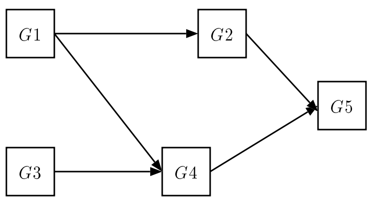


Abbildung 2.19: Netzplan von Aktivität $A1$

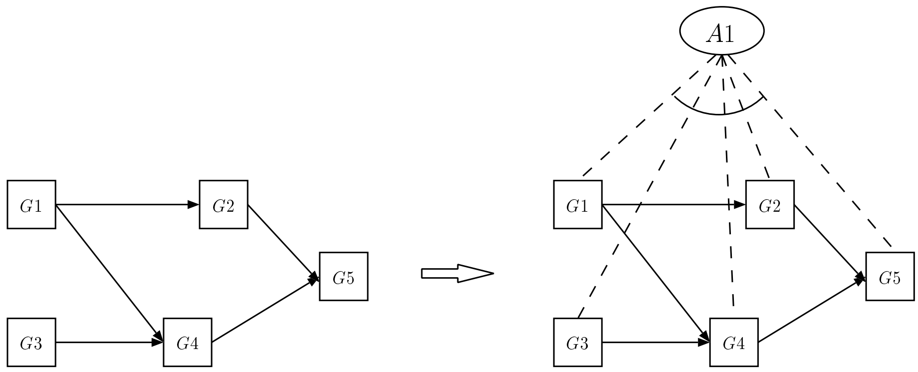


Abbildung 2.20: Transformation vom Netzplan der Aktivität $A1$ zum zugehörigen hierarchischen Netzplan

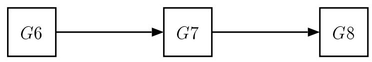


Abbildung 2.21: Netzplan von Aktivität $A2$

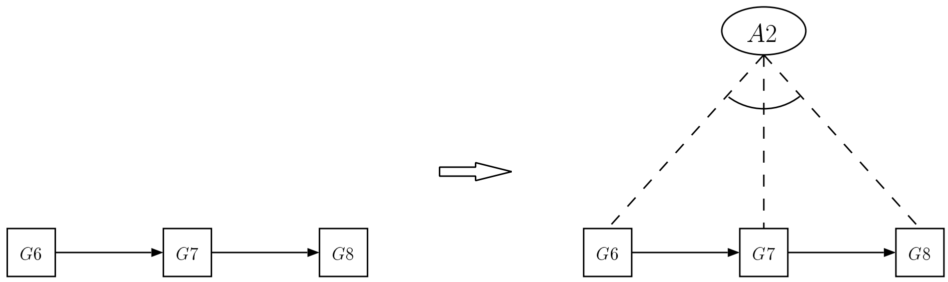


Abbildung 2.22: Transformation vom Netzplan der Aktivität $A2$ zum zugehörigen hierarchischen Netzplan

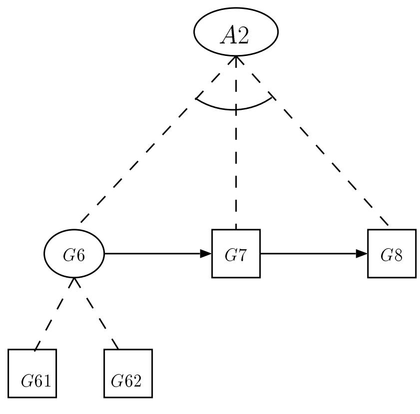


Abbildung 2.23: Hierarchischer Netzplan der Aktivität $A2$ mit Aktivitätsvarianten

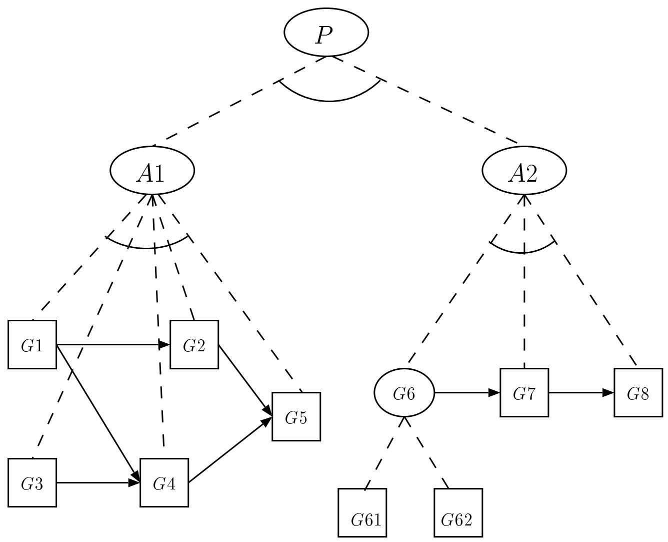


Abbildung 2.24: Der hierarchische Netzplan des gesamten Projektes P

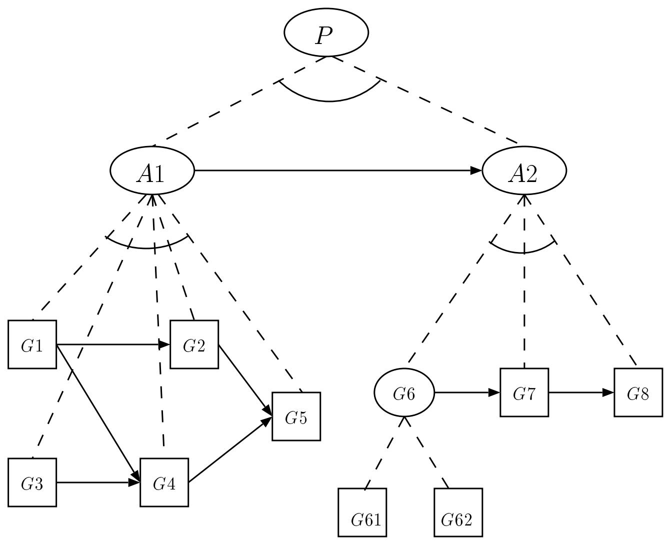


Abbildung 2.25: Der hierarchische Netzplan des Projektes P mit der zusätzlichen Nebenbedingung der sequentiellen Durchführung der Aktivitäten $A1$ und $A2$

nicht mehr einen Blatt sondern eine Knospe beschreiben soll.

Das gesamte Projekt P kann somit mittels des hierarchischen Netzplans aus Abbildung 2.24 dargestellt werden. Von diesem hierarchischen Netzplan kann abgelesen werden, daß die beiden mit AND-Kante verbundenen Aktivitäten $A1$ und $A2$ parallel durchgeführt werden können, da sie nicht durch eine Präzedenzkante verbunden sind.

Im Rahmen eines hierarchischen Netzplanes können Präzedenzbeziehungen zwischen Knospen bzw. Knospen und Blättern bestehen, d.h. zwei Knospen bzw. eine Knospe und ein Blatt können durch eine Präzedenzkante verbunden werden. Hierbei ist zu beachten, daß diese verbundenen Knoten denselben Vater haben müssen und mit einem Bogen verbunden sein müssen, d.h. ihre Inputkanten bilden gemeinsam AND-Kanten. Diese Erweiterung der Verwendung der Präzedenzkanten dient dazu, daß die sequentiellen Durchführungen der Aktivitäten (Teilabläufe) beschrieben werden können.
Wie Abbildung 2.25 zeigt, können beispielsweise die beiden Aktivitäten $A1$ und $A2$ aus Abbildung 2.24 durch eine Präzedenzkante verbunden werden. Nun beschreibt diese Präzedenzkante, daß die beiden Aktivitäten $A1$ und $A2$ sequentiell durchgeführt werden müssen.
So kann bei einem hierarchischen Netzplan für eine beliebige Menge T von Knoten, die durch AND-Kanten mit demselben Vater verbunden sind, eine Relation $E \subset T \times T$ erhalten werden, wobei E aus Präzedenzkanten bei T besteht. Hierbei ist zu beachten, daß bei einem *ein $\mathcal{RCPSV}$-Problem darstellenden* hierarchischen Netzplan jedes solche Tupel (T, E) *ein gerichteter schlichter azyklischer Graph* sein muß[30].
Bei dem hierarchischen Netzplan aus Abbildung 2.25 können z. B. insgesamt 3 verschiedene solche Relationen (T, E) erhalten werden. Diese sind

- $(\{A_1, A_2\}, \{(A_1, A_2)\})$,
- $(\{G_1, G_2, G_3, G_4, G_5\}, \{(G_1, G_2), (G_1, G_4), (G_3, G_4), (G_2, G_5), (G_4, G_5)\})$ und
- $(\{G_6, G_7, G_8\}, \{(G_6, G_7), (G_7, G_8)\})$.

Nun betrachten wir weiter hierarchische Netzpläne, die $\mathcal{RCPSV}$-Probleme spezifizieren. Wie in Abschnitt 2.1.3 erwähnt, kann im Rahmen eines $\mathcal{RCPSV}$-Problems eine Auswahlmöglichkeit unter alternativen Handlungsvarianten berücksichtigt werden. Diese Auswahlmöglichkeiten können bei einem hierarchischem Netzplan mittels OR-Kanten grafisch spezifiziert werden.
Wie bei dem $\mathcal{RCPS}$-Problem betrachtet, kann eine Ablaufstruktur zur Realisierung eines Projektes mittels eines Netzplans dargestellt werden (siehe Abschnitt 2.1.2). Von einem hierarchischen Netzplan, der ein $\mathcal{RCPSV}$-Problem spezifiziert, können alle zugehörigen alternativen Netzpläne abgeleitet werden. Jeder Netzplan stellt hierbei eine mögliche Ablaufstruktur zur Realisierung desselben Projektes dar. Vom hierarchischen Netzplan aus Abbildung 2.25 können beispielsweise zwei Netzpläne abgeleitet werden, wobei jeder Netzplan eine mögliche Ablaufstruktur zur Realisierung des betrachteten Projektes P darstellt. Die beiden abgeleiteten

[30] Präzedenzkanten, die zwei Knospen bzw. eine Knospe und einen Blatt verbinden, werden auch als Informationskanten bezeichnet (siehe z. B. [Rit98]).

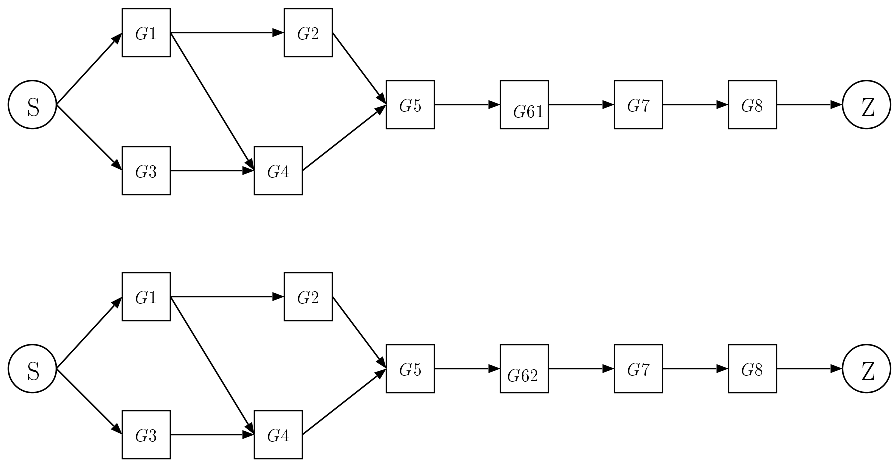


Abbildung 2.26: Ein hierarchischer Netzplan

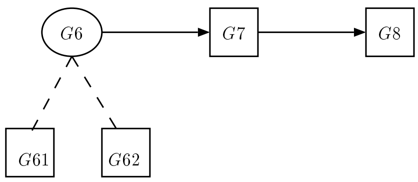


Abbildung 2.27: Ein hierarchischer Netzplan

Netzpläne zeigen Abbildung 2.26, wobei das Symbol 'S' Start und das Symbol 'Z' Ziel bedeutet.

Zum Schluß vereinbaren wir, daß bei einem hierarchischen Netzplan, der ein Problem aus der Spracheklasse $\mathcal{RCPSV}_{pa}$ darstellt, der Wurzel sowie die AND-Kanten weggelassen werden können. Der hierarchische Netzplan aus Abbildung 2.23 kann z. B. wie Abbildung 2.27 zeigt alternativ dargestellt werden.

2.3 Wissensrepräsentationssystem

Eine logikbasierte Sprache ist eine formale Sprache, in deren Syntax Aussagen (Ausdrücke) formuliert werden können. Die Semantik der Sprache verleiht den Aussagen einen Inhalt bzw. eine Bedeutung. Von uns werden die logikbasierten Sprachen $\mathcal{RSV}$ sowie $\mathcal{RCPSV}$ (siehe Kapiteln 3 und 6) entwickelt werden, unter deren Zuhilfenahme das $\mathcal{RSV}$- sowie $\mathcal{RCPSV}$-Problem terminologisch repräsentiert und gelöst werden können (siehe Abschnitte 2.1.4 und 2.1.3). Diese terminologische Repräsentation ist *eine neuartige Darstellungsweise* und *Lösungsmethode* der Ablaufplanungsprobleme. Ablaufplanungsprobleme können als terminologische Ausdrücke, von denen vor allem die Ablaufstrukturen der Projekte direkt ablesbar sind, *klar, einfach und abstrakt* formuliert werden. Ausgangspunkte bei der Entwicklung der terminologischen Repräsentation sind dabei die auf `KL-ONE` basierenden, terminologischen Wissensrepräsentationssprachen (siehe z. B. $\mathcal{ALC}$, $\mathcal{ALCR}$, und $\mathcal{ALCNR}$ [SSS88], [Hol89], [HNSS90], [DLNN91]).

Eine Wissensbasis[31], die in einer angemessenen Repräsentationssprache formuliertes Wissen aufnimmt und Verarbeitung von Wissen ermöglicht, wird Wissensrepräsentationssystem (*engl. knowledge representation system*) genannt. Eine Art von Wissensrepräsentationssystemen sind die terminologischen Repräsentationssysteme, die einer Weiterentwicklung des Wissensrepräsentationssystems `KL-ONE` [BS85] entsprechen.

Eine terminologische Sprache ist ein Repräsentationsschema für ein terminologisches Repräsentationssystem; sie ist eine formale Sprache und besteht aus zwei Elementen. Diese Elemente sind die *syntaktischen Primitive* und die *Operatoren.* Die syntaktischen Primitive setzen sich aus *Konzepten* und *Rollen* zusammen. Ein Operator ist *konzept-konstruierend* oder *rollen-konstruierend*, je nach der Art der durch einen Operator realisierten Verknüpfung.
Die verschiedenen terminologischen Repräsentationen unterscheiden sich bezüglich der syntaktischen Operatoren. Im folgenden wird ein Ausschnitt der von Schmidt-Schauß und Smolka [SSS88] eingeführten Terminologischen Sprache $\mathcal{ALC}$ vorgestellt.

Das *Vokabular* von $\mathcal{ALC}$ besteht aus drei *disjunkten* Mengen von Symbolen. Diese Mengen sind:

- Eine Menge von Zeichen für primitive Konzepte.
- Eine Menge von Zeichen für primitive Rollen.
- Eine Menge von strukturellen Symbolen.

Die Menge der Zeichen, die zur Identifikation primitiver Konzepte benutzt werden, enthält stets zwei besonders ausgezeichnete Symbole. Diese beiden Symbole sind '$\top$' und '$\perp$' und werden Konzeptsymbole genannt. Das Symbol '$\top$' wird zur Identifikation des *Topkonzepts*, das Symbol '$\perp$' zur Identifikation des *Bottomkonzepts* (leeres Konzepts) benutzt.
Die Menge der strukturellen Symbole enthält die 5 Symbole '**and**', '**or**', '**not**', '**some**' und '**all**'. Diese Symbole heißen Operatoren und haben die folgenden Bedeutungen:

- '**and**' : Dieser Operator dient zur Spezifikation *der Konjunktion* zweier Konzepte.

[31] Die hier aufgeführte Erläuterung über `KL-ONE`-Wissensrepräsentationssysteme wurde teilweise [Kim94] entnommen.

- '**or**' : Dieser Operator dient zur Spezifikation *der Disjunktion* zweier Konzepte.
- '**not**' : Dieser Operator dient zur Spezifikation *der Negation* eines Konzepts.
- '**some**' : Dieser Operator dient zur Spezifikation *der existentiellen Beschränkung* auf einer Rolle und einem Konzept.
- ' **all**' : Dieser Operator dient zur Spezifikation *der universellen Beschränkung* auf einer Rolle und einem Konzept.

Die Operatoren '**and**', '**or**', '**not**' sind konzept-konstruierend auf Konzepten und die Operatoren '**some**', '**all**' sind konzept-konstruierend auf Rollen und Konzepten. Die Konzeptbeschreibung ist wie folgt definiert [Sch91]:

Wenn 'A' ein beliebiges primitives Konzeptsymbol (Konzeptname) ist, können zwei durch 'C' und 'D' bezeichnete Konzeptbeschreibungen (Konzeptterme) durch andere Konzepte nach der folgenden Regel (in Backus Naur Form) konstruiert werden:

$$\mathrm{C},\ \mathrm{D} \rightarrow \mathrm{A} \mid (\mathbf{and}\,\mathrm{C}\,\mathrm{D}) \mid (\mathbf{or}\,\mathrm{C}\,\mathrm{D}) \mid (\mathbf{not}\,\mathrm{C}) \tag{1}$$

Sei 'Q' ein primitives Rollensymbol. Erweitert wird die Regel (1) durch das Hinzufügen von der existenziellen und universellen Beschränkungen (*engl. existential and engl. universal restriction constructs*):

$$\mathrm{C},\ \mathrm{D} \rightarrow \cdots \mid (\mathbf{some}\,\mathrm{Q}\,\mathrm{C}) \mid (\mathbf{all}\,\mathrm{Q}\,C). \tag{2}$$

Alle möglichen Konzeptbeschreibungen 'C' oder 'D' werden rekursiv durch die Regeln (1) und (2) definiert.

Die modell-theoretische Semantik von Konzeptbeschreibungen in $\mathcal{ALC}$ ist durch eine *Interpretation* $\mathcal{I}$ gegeben, die als ein Paar $(\mathcal{D}^{\mathcal{I}},\ \cdot^{\mathcal{I}})$ definiert ist. $\mathcal{D}^{\mathcal{I}}$ ist *die Domäne* der Interpretation, und $\cdot^{\mathcal{I}}$ ist *eine Interpretationsfunktion.* Eine Interpretationsfunktion ordnet jeder Konzeptbeschreibung 'C' eine Untermenge $\mathrm{C}^{\mathcal{I}}$ von $\mathcal{D}^{\mathcal{I}}$ und jeder Rolle 'Q' eine binäre Relation $\mathrm{Q}^{\mathcal{I}}$ über der Menge $\mathcal{D}^{\mathcal{I}}$ (d.h. $\mathrm{Q}^{\mathcal{I}} \subseteq \mathcal{D}^{\mathcal{I}} \times \mathcal{D}^{\mathcal{I}}$) zu. Die Interpretationsfunktion spezifiziert die Bedeutung der gekennzeichneten und zusammengesetzten Konzepte wie folgt:

$$\begin{aligned}
\top^{\mathcal{I}} &= \mathcal{D}^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \phi \\
(\mathbf{and}\,\mathrm{C}\,\mathrm{D})^{\mathcal{I}} &= \mathrm{C}^{\mathcal{I}} \cap \mathrm{D}^{\mathcal{I}} \\
(\mathbf{or}\,\mathrm{C}\,\mathrm{D})^{\mathcal{I}} &= \mathrm{C}^{\mathcal{I}} \cup \mathrm{D}^{\mathcal{I}} \\
(\mathbf{not}\,\mathrm{C})^{\mathcal{I}} &= (\mathrm{C}^{\mathcal{I}})' \qquad (= \mathcal{D}^{\mathcal{I}} - \mathrm{C}^{\mathcal{I}}) \\
(\mathbf{some}\,\mathrm{Q}\,\mathrm{C})^{\mathcal{I}} &= \{x \mid (\exists y)[(x,y) \in \mathrm{Q}^{\mathcal{I}}\ \&\ y \in \mathrm{C}^{\mathcal{I}}]\} \\
(\mathbf{all}\,\mathrm{Q}\,\mathrm{C})^{\mathcal{I}} &= \{x \mid (\forall y)[(x,y) \in \mathrm{Q}^{\mathcal{I}} \Rightarrow y \in \mathrm{C}^{\mathcal{I}}]\}
\end{aligned}$$

Allen auf `KL-ONE` basierenden Wissensrepräsentationssystemen ist gemeinsam, daß sie aus zwei Komponenten zusammengesetzt sind (siehe z. B. `KRIS` [BH90], `BACK` [NvL88]und [Neb90], `CLASSIC` [BBMR89]). Diese zwei Komponenten sind eine **TBox** und eine **ABox**. Jede dieser beiden Komponenten hat ihre eigene Repräsentationssprache und ihr eigenes Folgerungssystem (*engl. inference system*):
Die **TBox** dient zur terminologischen Repräsentation. Sie enthält in *einer terminologischen Sprache* beschriebene Informationen. Diese Informationen entsprechen Wissen über Konzepte und Rollen sowie über Beziehungen zwischen Konzepten (bzw. Rollen).
Die **Abox** enthält *die behauptenden Repräsentationen* (*engl. assertional representations*), d.h. sie enthält Informationen über die einzelnen Instanzen von Konzepten (Elemente von Mengen) bzw. Instanzen von Rollen (Paare von Elementen). Durch diese Informationen wird eine konkrete Welt beschrieben.
Auf Basis der beiden Komponenten **TBox** sowie **ABox** werden Algorithmen entwickelt, um das Abfragen und Folgern (Verarbeiten) von impliziten Wissen zu ermöglichen.

Nun zeigen wir anhand eines Beispiels, wie die Operatoren in $\mathcal{ALC}$ verwendet werden, um zusammengesetzte Konzepte zu beschreiben. Als Beispiel wird hierbei mittels der Sprache $\mathcal{ALC}$ Wissen über eine Umwelt von Personen repräsentiert.

'Person', 'Weiblich', 'Männlich ', 'Student', 'BWL-Student', 'Vegetarier' und 'Asien' seien primitive Konzepte. 'hat-Kind' und 'kommt-aus' seien primitive Rollen, wobei 'Person' dem Top-Konzept entspreche. Die folgenden Konzeptbeschreibungen sind damit beispielsweise möglich.

(**and** Männlich (**not** Vegetarier))
(**all** hat-Kind Student)
(**and** BWL-Student (**some** kommt-aus Asien))

Alle obigen Ausdrücke beschreiben eine bestimmte Menge von Individuen. Die Ausdrücke repräsentieren die folgende Menge:

- Der Ausdruck (**not** Vegetarier) beschreibt die Menge aller nicht vegetarischen Individuen. In der Folge beschreibt der Ausdruck (**and** Männlich (**not** Vegetarier)) die Menge aller männlichen nicht vegetarischen Individuen.

- Der Ausdruck (**all** hat-Kind Student) beschreibt die Menge aller Personen, deren Kinder sämtlich Studenten sind.

- Der Ausdruck (**and** BWL-Student (**some** kommt-aus Asien)) beschreibt die Menge aller BWL-Studenten, die aus Asien kommen.

Zur Lösung von Ablaufplanungsproblemen werden die Ansätze der Wissensrepräsentationssysteme in der vorliegenden Arbeit wie folgt angewendet:

Wie in den Kapiteln 3 und 6 gezeigt werden wird, ähneln primitive Konzepte der Wissensrepräsentationssprache $\mathcal{ALC}$ Grundaktionen (atomare Formeln; atomare Bestandteile der Logik)[32] aus $\mathcal{RSV}$ bzw. $\mathcal{RCPSV}$. Jede Grundaktion wird durch die Verbindung einer Ressource mit einer zeitlichen Dauer charakterisiert.
Um Nebenbedingungen zwischen Aktivitäten zu beschreiben, werden in $\mathcal{RSV}$ insgesamt drei Operatoren '**seq**', '**xor**' und '**pll**' eingeführt, während in $\mathcal{RCPSV}$ zwei Operatoren '**xor**' und '**hnet**' eingeführt werden. Diese haben die folgenden Bedeutungen:

- Mittels des Operators '**seq**' werden Präzedenzbeziehungen sowie AND-Dekompositionen spezifiziert.
- Mittels des Operators '**xor**' werden OR-Dekompositionen spezifiziert.
- Mittels des Operators '**pll**' wird die parallele Bearbeitung von Teilprojekten (Teilabläufen) spezifiziert.
- Mittels des Operators '**hnet**' wird die Anordnung von Aktivitäten, entsprechend vorgegebener Präzedenzbeziehungen, spezifiziert[33] .

In $\mathcal{RSV}$ bzw. $\mathcal{RCPSV}$ können ebenfalls durch einen induktiven Prozeß (ähnlicher Prozeß wie die vorherigen Regeln (1) und (2)), der die Art und Weise der syntaktischen Zusammensetzung von (atomaren) Formeln und Operatoren bestimmt, komplexere syntaktische Formeln der Logik gebildet werden. Mittels solcher zusammengesetzten Formeln, die wir als Aktionen bezeichnen werden, können Teilprojekte innerhalb eines Projekts bzw. ein komplettes Projekt (ein $\mathcal{RSV}$-Problem bzw. ein $\mathcal{RCPSV}$-Problem) repräsentiert werden.

Ein grundsätzlicher Unterschied zwischen herkömmlichen terminologischen Sprachen und den in dieser Arbeit vorgestellten terminologischen Sprachen besteht bei der Definition der Semantik (Definition einer Interpretationsfunktion für alle Ausdrücke einer Sprache). Statt der bei Wissensrepräsentationssystemen üblichen mengenweisen Interpretation bildet eine im Rahmen der Logik definierte Interpretationsfunktion jede Formel (Ausdruck) auf eine Menge ab, die aus allen aus dem Ausdruck herleitbaren aktiven Ablaufplänen besteht. Auf dieser Semantik basierend wird ein Kalkül definiert, mit dem ein komplexer Ausdruck *normalisiert* werden kann (siehe für nähere Erläuterungen Kapitel 3), wobei mehrere Netzplan darstellende Ausdrücke durch einen einzelnen **xor**-Operator verknüpft

[32] Wie bereits erwähnt, können diese als nicht weiter zerlegbare Produktionsabschnitte bzw. Aktivitäten betrachtet werden.
[33] siehe für näheres Kapitel 6

werden. Auf diese durch einen Normalisierungsprozeß erhaltenen einzelnen Ausdrücke (Netzpläne) kann eines der beiden algorithmischen Verfahren, die in Kapiteln 4 und 7 vorstellt werden, angewendet werden. Diese Verfahren erlauben die maschinelle Ermittlung *aller nichtredundanten (ressourcenkonfliktfreien) Ablaufpläne mit minimaler Gesamtbearbeitungszeit.*

2.4 Funktionale Programmiersprachen

In den folgenden Kapiteln werden wir Modelle kennenlernen, die zur Lösung der in den letzten Abschnitten vorgestellten Schedulingprobleme geeignet sind. Desweiteren wird gezeigt werden, wie solche Modelle unter Zuhilfenahme der funktionalen Programmiersprache Haskell ([HAB+96], [PHA+97]) implementiert werden können. In diesem Abschnitt soll eine kurze Einführung in den Themenbereich der funktionalen Programmierung insbesondere im Bezug auf Haskell gegeben werden.

Die Geschichte der funktionalen Programmierung reicht in die 30er Jahre dieses Jahrhunderts zurück. Das Konzept der funktionalen Programmierung basiert auf dem von Church entwickelten λ-Kalkül ([Chu32], [Chu41]). Der λ-Kalkül ist ein mathematischer Formalismus zur Beschreibung von Funktionen. Gegenüber anderen mathematischen Formalismen zur Beschreibung von Funktionen zeichnet sich der λ-Kalkül dadurch aus, daß er auch die Formulierung unbenannter Funktionen ermöglicht.

Sollen für eine gegebene Funktion die Funktionswerte durch einen Computer berechnet werden, so muß die Funktion zunächst in einer für den Computer verständlichen Form beschrieben werden. Für eine derartige Beschreibung wird gewünscht, daß sie die Vorteile der prägnanten mathematischen Definitionsweise beibehält. Aus diesem Wunsch heraus hat sich das Konzept der funktionalen Programmierung entwickelt. Funktionale Programmierung ermöglicht Funktionsbeschreibungen auf eine mathematische, vom Berechnungsvorgang abstrahierende, Weise.
Funktionale Sprachen sind im Gegensatz zu imperativen Sprachen wie beispielsweise C [KR77], COBOL [ANS85], FORTRAN [ISO97] von ihrer Natur her deklarativ. Während eine imperative Sprache (im allgemeinen ausgedrückt durch eine Folge sequentiell abzuarbeitender Befehle) genau verschreibt, wie ein Problem zu lösen ist, erlaubt die Verwendung einer funktionalen Sprache die Abstrahierung vom “wie” hin zum “was” gelöst werden soll.
Ein funktionales Programm besteht aus einer Menge von Definitionen, die mathematische Funktionen beschreiben und im allgemeinen skriptartig festgehalten werden. Programmieren in einer funktionalen Programmiersprache bedeutet Funktionen zu konstruieren, um ein gegebenes Problem zu lösen.

Funktionale Sprachen sind zur Erstellung komplexer Software-Systeme in manchen Fällen (z. B. Compiler[34]) besser geeignet als herkömmliche, imperative Sprachen. Dies ist darin begründet, daß funktionale Programmiersprachen bei einer Vielzahl von Problemstellungen eine knappere und prägnantere Programmformulierung ermöglichen als imperative Sprachen. Besonders gut geeignet sind funktionale Programmiersprachen zur Behandlung mathematisch orientierter Problemstellungen.
Funktionale Sprachen bieten sich als Spezifikationsmedium an, weil ein Problem knapp, präzise und mit angemessenem Abstraktionsgrad formuliert werden kann. Funktionale Sprachen können in diesem Zusammenhang im Bereich des "rapid prototyping" sinnvoll eingesetzt werden. "Rapid prototyping" bedeutet die schnelle Erstellung einer Applikation mit modellhaftem Charakter. "Rapid prototyping" wird eingesetzt, um die Tragfähigkeit der grundlegenden Ideen und Konzepte bei der Realisation eines Softwaresystems zu überprüfen.

Zum Thema "funktionale Programmierung" existiert ein reichhaltiges Informationsangebot. Eine Übersicht über verfügbare Informationsquellen zum Thema "funktionale Programmierung" kann im Internet unter der Adresse `http:\\www.haskell.org` gefunden werden. Eine Auswahl von Literaturquellen, die aus letzterem Internetangebot übernommen wurde, soll kurz vorgestellt werden.

Für Anfänger im Bereich der funktionalen Programmierung sind Thompson [Tho99] und Bird [Bir98] zu empfehlen. Beide Bücher beinhalten zahlreiche in Haskell geschriebene Beispiele funktionaler Programme. Thompson [Tho99] gibt hilfreiche Hinweise bzgl. Programmdesign, Problemlösungen und weist auf vermeidbare, häufig auftretende Fehler hin. Bird [Bir98] erläutert ausführlich, auf welche Prinzipien sowie Techniken das Programmieren mit Funktionen basieren sollte.
Zur Erweiterung und Vertiefung der Kenntnisse über funktionale Programmierung können Davie [Dav92], Rabhi und Lapalme [RL99] sowie Okasaki [Oka98] herangezogen werden. Davie [Dav92] stellt die Basiskonzepte der funktionalen Programmierung vor. Desweiteren beschäftigt er sich vor allem mit der Frage, in welcher Beziehung der λ-Kalkül und Haskell stehen. Okasaki [Oka98] beschreibt Datenstrukturen und Techniken des Datenstrukturdesign im Bezug auf funktionale Sprachen.
Funktionale Programmiersprachen können nach verschiedenen Merkmalen differenziert werden, wie der Auswertungsstrategie und der Differenzierung nach puren und impuren Sprachen. Einen Einblick in diese Differenzierungsmöglichkeiten gibt die Einführung der Dissertation [Kut00].

In der vorliegenden Arbeit benutzen wir alleinig die funktionale Sprache Haskell. Haskell ist eine pure, verzögernd auswertende, polymorph getypte funktionale

[34] Der HASKELL-Compiler GHC besipielsweise ist selber in HASKELL programmiert.

Programmiersprache. Um einen kurzen Einblick in Haskell zu geben und die erwähnten Vorzüge funktionaler Programmierung an einem Beispiel zu verdeutlichen, wird die Programmierung des bekannten Quicksort-Algorithmus zunächst Haskell und dann in der imperativen Sprache PASCAL gezeigt.

Quicksort in Haskell

```
qsort []     = []
qsort (x:xs) = qsort [ u| u<-xs, u<x] ++
                     [x] ++
               qsort [ u| u<-xs, u>=x]
```

Es soll kurz erläutert werden, wie das obige Programm zu lesen ist:

Die erste der zwei Gleichungen besagt, daß bei der Anwendung der Funktion `qsort` auf eine leere Liste eine leere Liste zurückgegeben wird (`qsort [] = []`). Für eine nichtleere Liste `x:xs`, wobei `x` das erste Element der Liste identifiziert, wird die Teilliste `xs` in die zwei Listen `[ u| u<-xs, u<x]` und `[ u| u<-xs, u>=x]` zerlegt. In der ersteren `[ u| u<-xs, u<x]` werden alle Elemente aus `xs` (`u<-xs`), die kleiner als `x` sind, aufgenommen und auf diese Liste wird die Funktion `qsort` rekursiv angewendet. In der letzteren Liste `[ u| u<-xs, u>=x]` werden alle Elemente aus `xs` (`u<-xs`), die größer oder gleich `x` sind, aufgenommen und auf diese Liste wird die Funktion `qsort` rekursiv angewendet. Zum Schluß erfolgt eine Konkatenation von Listen (++), die als Ergebnis die gesuchte aufsteigend sortierte Liste liefert.

Quicksort in PASCAL[35]

```
Procedure quicksort (l,r: integer);
var x, i, j, tmp : integer;
begin
   if r > l then
        begin
           x:=a[l]; i:=l; j:=r+1;
           repeat
              repeat i:=i+1 until a[i]≥x;
              repeat j:=j-1 until a[j]≤x;
              tmp:= a[j]; a[j]:= a[i]; a[i]:=tmp;
           until j≤i;
           a[i]:=a[j]; a[j]:=a[l]; a[l]:=tmp;
           quicksort (l, j-1);
           quicksort (j+1, r);
         end
 end
```

[35] Der PASCAL-Quellcode des Quicksort-Algorithmus wurde aus [Loo90] entnommen.

Es ist offensichtlich, daß die Programmierung des Quicksort-Algorithmus in PASCAL deutlich länger ist. Ferner ist die divide-and-conquer-Strategie des Algorithmus bei der Version als Haskell-Programm direkt ersichtlich, in der Version als PASCAL-Programm jedoch erst nach genauerem Hinsehen zu erkennen. Dies unterstreicht die zu Beginn dieses Abschnitts getroffene Aussage, daß funktionale Programme von ihrer Natur her sehr viel deklarativer sind als Programme, die in imperativen Sprachen formuliert sind.

2.5 Komplexitätstheorie

Die $\mathcal{NP}$-Vollständigkeit des $\mathcal{RCPS}$-Problems wurde bereits in [LRK78] gezeigt. Wie zuvor in Abschnitt 2.1.3 erwähnt, gibt es heute vergleichweise wenig Publikationen [Elm77], [Tal82], [Rit98], die sich mit erweiterten Formen des $\mathcal{RCPS}$-Problems beschäftigen. Entsprechend fehlt heute die Untersuchung über die zeitliche Komplexität erweiterter Formen des $\mathcal{RCPS}$-Problems wie z. B. des $\mathcal{RCPSV}$-Problems und dem diesen gleichmächtigen $\mathcal{RSV}$-Problems.

Im Rahmen dieser Arbeit wird untersucht, welcher zeitliche Aufwand erforderlich ist, um das $\mathcal{RSV}$-Problem maschinell zu lösen. Die Untersuchungen über die zeitliche Komplexität des $\mathcal{RSV}$-Problems, die in Kapitel 9 erfolgen werden, basieren auf dem Modell der Turingmaschine. Das mathematische Rechnermodell "Turingmaschine" wird vor allem im Gebiet der Komplexitätstheorie häufig verwendet, da dieses Modell die leichte, intuitive Vorstellung der zeitlichen Dauer[36] ermöglicht[37].

Das Modell "Turingmaschine" wurde 1936 von Alan Mathison Turing (1912 - 1954) vorgestellt[38]. Dieses Modell ist so mächtig, daß eine partielle Funktion $f : \sum^* \to \sum^*$ genau dann algorithmisch berechenbar ist, wenn sie auf einer Turingmaschine berechenbar ist (Churchsche These) [Bö92], [Weg93].

Alle bekannten Modelle universeller Rechner können die gleiche Klasse von Problemen lösen [Weg93]. Ein Rechnermodell, das Registermaschine genannt wird, bildet ein stark vereinfachtes Modell realer Rechner und dient daher auch als Basis für realistische Rechenzeitbetrachtungen. Eine partielle Funktion ist genau dann auf einer Turingmaschine berechenbar, wenn sie auf einer Registermaschine berechenbar ist. Desweiteren ist eine partielle Funktion genau dann auf einer

[36] Die Rechenzeit einer Turingmaschine auf einer Eingabe ist gleich der Anzahl der durchgeführten Rechenschritte.

[37] Die einzelnen Rechenschritte der Turingmaschinen haben nur sehr lokale Wirkungen. In jedem Augenblick der Berechnung steht fest, an welcher Stelle der Zeichenfolge gearbeitet wird, und der einzelne Rechenschritt besteht darin, daß entweder das an dieser Stelle stehende Zeichen geändert wird oder daß die bearbeitete Stelle um eins nach rechts oder links verschoben wird.

[38] Grundlegende Erläuterungen können in [PS92], [Bö92], [Mar91] [HU88] und [Weg93] gefunden werden.

Turingmaschine effizient, d. h. in polynomieller Zeit, berechenbar, wenn sie auf einer Registermaschine effizient berechenbar ist [Weg93].

2.5.1 Turingmaschine, Die Klasse $\mathcal{P}$, Die Klasse $\mathcal{NP}$

Das Modell der Turingmaschine, mit dem quantitative Aussagen über den Aufwand an Rechenzeit gemacht werden können, formalisiert Algorithmen umfassend und exakt[39]. Es gibt deterministische sowie nichtdeterministische Turingmaschinen [GJ79]. Formale Beschreibungen der deterministischen sowie nichtdeterministischen Turingmaschinen werden in diesem Abschnitt gegeben. Die Komplexität von Problemen wird klassifiziert, je nachdem mit welcher Art von Turingmaschine das Problem mit welchem Rechenaufwand gelöst werden kann.

2.5.1.1 Deterministische Turingmaschinen

Eine Turingmaschine besteht aus einer Endlichen Kontrolle mit k Lese-Schreib-Köpfen und k in Felder unterteilten Bändern. Die Endliche Kontrolle besteht aus einer Menge von Anweisungen, die mittels einer partiellen Funktion dargestellt werden. Jedem Band ist ein Lese-Schreib-Kopf zugeordnet, der zu einem Zeitpunkt genau ein Feld des Bandes bearbeiten kann. Jedes Band ist links beschränkt und rechts unendlich. In jedem Feld der Bänder steht genau ein Zeichen aus einer endlichen Menge von Bandsymbolen.
Formal wird eine Turingmaschine wie folgt beschrieben:

Definition 2.5.1. *Eine k(mit $k \geq 1$)-Band deterministische Turingmaschine TM ist ein 7-Tupel $(Q, \Sigma, \Gamma, \delta, q_0, B, q_f)$, wobei folgendes gilt:*

Q	*ist eine endliche Menge von Zuständen.*
Γ	*ist eine endliche Menge von erlaubten Bandsymbolen.*
B	*ist das Blank, ein Symbol aus Γ.*
$\sum$	*ist die Menge der Eingabesymbole, eine Teilmenge von Γ, die das Blank B nicht einschließt.*
δ	*ist eine partielle Funktion, eine Abbildung von $Q \times \Gamma^k$ nach $Q \times \Gamma^k \times \{L, R, S\}^k$.*
q_0	*aus Q ist der Anfangszustand.*
q_f	*aus Q ist der akzeptierende Zustand.*

Anfangs enthalten die n (mit $n \geq 0$) am weitesten links stehenden Felder des ersten Bandes die Eingabe, deren Symbole aus der Menge Σ (Eingabesymbolen) stammen. Die verbleibenden unendlich vielen Felder des ersten Bandes und der sonstigen Bänder enthalten jeweils das Leerzeichen B.

[39] Grundlegende Erläuterungen befinden sich in [PS92].

Falls die TM im Zustand q die k Zeichen $(a_1, a_2, \cdots, a_k)$ mittels des Lese-Schreib-Köpfe einliest und $\delta(q,(a_1,a_2,\cdots,a_k)) =< q',(a'_1,\cdots,a'_k),(m_1,\cdots,m_k) >$mit $a'_1,\cdots, a'_k \in \mathit{\Gamma}$ und $m_1,\cdots, m_k \in \{L,R,S\}$ ist, sagt folgende die Gleichung aus:

- Auf jedem Band wird das Symbol a_i $(i = 1,\cdots,k)$ durch das Symbol a'_i ersetzt.
- Die k-Lese-Schreib-Köpfe werden gemäß $m_1,\cdots, m_k \in \{L,R,S\}$ um ein Feld bewegt oder stehengelassen. $L(R)$ bezeichnet dabei eine Bewegung um ein Feld nach links (rechts). S bezeichnet, daß der entsprechende Kopf nicht bewegt wird.
- Der Zustand wird von q nach q' geändert.

Falls $\delta(q,(a_1,a_2,\cdots,a_k))$nicht definiert ist, bleibt die Maschine ohne zu akzeptieren stehen.
Die Abbildung 2.28 zeigt einen Beispielübergang für eine *Zweiband-*Turingmaschine.

Definition 2.5.2. *Eine Konfiguration* [40] *einer Turingmaschine TM ist ein Tupel $(\alpha_{11}q\alpha_{12},\alpha_{21}q\alpha_{22},\cdots,\alpha_{k1}q\alpha_{k2})$, wobei*

- *q aus Q der gegenwärtige Zustand der Maschine TM ist und*
- *$\alpha_{11}\alpha_{12},\alpha_{21}\alpha_{22},\cdots \alpha_{k1}\alpha_{k2}$ Zeichenketten sind, die die Inhalte der k Bänder bis zum am weitesten rechts stehenden, von einem Blank B verschiedenen Symbol darstellen.*

Hierbei wird angenommen, daß Q und $\mathit{\Gamma}$ disjunkt sind, und die Bandköpfe das am weitesten links stehende Symbol von $\alpha_{12},\alpha_{22},\alpha_{32},\cdots,\alpha_{k2}$ lesen oder, falls $\alpha_{i2} = B$ $(i = 1,\cdots k)$, ein Blank.

Definition 2.5.3. *Sei* $(X_{11}X_{12}\cdots X_{1i-1}qX_{1i}\cdots X_{1n_1}, \cdots, X_{k1}X_{k2}\cdots X_{kj-1}qX_{kj}\cdots X_{kn_k})$ *eine Konfiguration und* $\delta(q,(X_{1i},\cdots,X_{kj})) =< q',(X'_{1i},\cdots,X'_{kj}),(L,\cdots,R) >$. *Für $i > 1$ heißt die nächste Konfiguration*

$$(X_{11}X_{12}\cdots q'X_{1i-1}X'_{1i}\cdots X_{1n_1},\cdots X_{k1}X_{k2}\cdots X_{kj-1}X'_{kj}q'X_{kj+1}\cdots X_{kn_k})$$

direkte Folgekonfiguration.

Falls in der Definition 2.5.3 $i-1 = n_1,\cdots$ bzw. $j-1 = n_k$ gilt, dann sind $X_{1i},\cdots$ bzw. X_{kj} ein B. Für $i = 1$ gibt es keine nächste Konfiguration, da sich der zu dem Band gehörige Kopf nicht über das linke Ende hinaus bewegen darf.

[40] Eine Konfiguration ist eine Momentaufnahme der Turingmaschine während einer Berechnung.

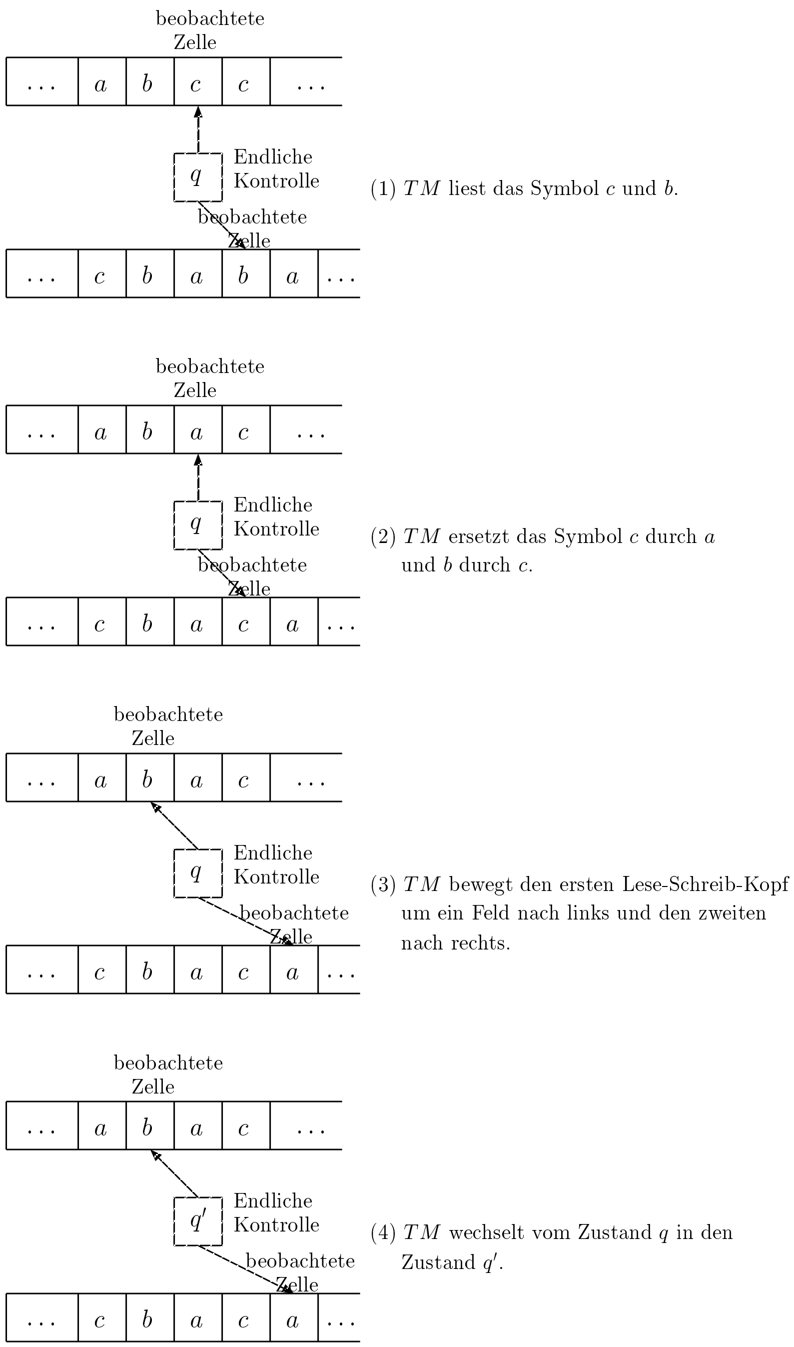


Abbildung 2.28: Ein Übergang für eine Zweiband-Turingmaschine

Definition 2.5.4. *Der Übergang von einer Konfiguration zur direkten Folgekonfiguration einer Turingmaschine wird als ein Rechenschritt einer Turingmaschine definiert.*

Definition 2.5.5. *Eine TM-Berechnung ist eine endliche Folge* $\boldsymbol{B} =< K_0, \cdots, K_n >$*von TM-Konfigurationen, so daß für alle* $i < n$ K_{i+1} *die direkte Folgekonfiguration von* K_i *ist.* K_n *ist die Endkonfiguration von* $\boldsymbol{B}$*, wenn keine Folgekonfiguration existiert. Ist* K_n *eine Endkonfiguration, dann benötigt die Berechnung* $\boldsymbol{B}$ n *Rechenschritte.*

Definition 2.5.6. *Eine TM-Berechnung* $\boldsymbol{B} =< K_0, \cdots K_n >$ *ist vollständig, wenn* K_n *eine Endkonfiguration ist.*

Definition 2.5.7. *Eine vollständige Berechnung* $\boldsymbol{B} =< K_0, \cdots K_n >$ *ist eine akzeptierende Berechnung, wenn der Zustand von* K_n *gleich dem akzeptierenden Zustand* q_f *ist.*

Definition 2.5.8. *Sei* $X \in \Sigma^*$*ein beliebiges Wort. Eine Turingmaschine* TM *akzeptiert* X*, wenn eine akzeptierende Berechnung mit* X *als Eingabe existiert. Die Menge* $L(TM) = \{X \in \Sigma^* | \ TM \text{ akzeptiert } X\}$ *heißt die von TM akzeptierte Sprache.*

Bemerkung: Jede Berechnung einer deterministischen Turingmaschinen kann als eine Folge betrachtet werden, bei der es für jede Konfiguration höchstens eine direkte Folgekonfiguration gibt; somit ist der Rechenweg einer deterministischen Turingmaschine eindeutig bestimmt.

2.5.1.2 Deterministische Polynomielle Berechenbarkeit - Die Klasse $\mathcal{P}$

Die Klasse der Probleme, die mittels einer deterministischen Turingmaschine in polynomiell beschränkter Laufzeit gelöst werden können, wird wie folgt spezifiziert:

Im weiteren bezeichnet $\mathbb{N}$ die Menge der natürlichen Zahlen und $\mathbb{R}$ die Menge der reellen Zahlen.

Definition 2.5.9. *Sei* TM *eine (k-Band) deterministische Turingmaschine über dem Alphabet* $\sum$ *und* T *eine Funktion* $\mathbb{N}$ *nach* $\mathbb{R}$*. Die worst case Rechenzeit von* TM *ist* $T(n)$*, wenn* TM *für jede Eingabe* $(\in \sum^*)$ *der Länge* n *höchstens* $T(n)$ *Rechenschritte macht, bevor sie anhält.*

Definition 2.5.10. *Eine deterministische Turingmaschine TM heißt polynomiell beschränkt, wenn die worst case Rechenzeit $T(n)$ von TM polynomiell beschränkt ist.*

Definition 2.5.11. *Ein Problem heißt polynomiell (effizient) berechenbar, wenn eine deterministische Turingmaschine (ein Algorithmus) zur Lösung des Problems existiert, deren worst case Rechenzeit $T(n)$ polynomiell beschränkt ist.*

Eine 1 (mit $k = 1$)-Band deterministische Turingmaschine ist eine grundlegende Turingmaschine, die am häufigsten bei der Vorstellung einer Turingmaschine verwendet wird. Alle Varianten, mehrbändige ($k \geq 2$) sowie einbändige ($k = 1$)Turingmaschinen, sind rechnerisch äquivalent, wie die folgenden Lemmata besagen.

Lemma 2.5.1. *[Mar91] [HU88] Wenn eine partielle Funktion auf einer mehrbändigen Turingmaschine berechenbar ist, dann ist sie auch auf einer einbändigen Turingmaschine berechenbar.*

Lemma 2.5.2. *[HU88] Wenn eine partielle Funktion auf einer mehrbändigen Turingmaschine effizient berechenbar ist, dann ist sie auch auf einer einbändigen Turingmaschine effizient berechenbar.*

Die Klasse $\mathcal{P}$

Die Klasse $\mathcal{P}$ wird aus sämtlichen deterministisch polynomiell lösbaren Problemen gebildet. Diese Klasse wird auch mit der Klasse der *effizient lösbaren* Probleme[41] identifiziert.

Definition 2.5.12. *Die Klasse $\mathcal{P}$ ist die Menge sämtlicher durch eine polynomiell beschränkte deterministische Turingmaschine akzeptierbaren Sprachen. Formal:*

$$\mathcal{P} = \{S \subseteq \textstyle\sum^* \mid \exists \text{ eine polynomiell beschränkte deterministische} \\ \text{Turingmaschine } TM \text{ mit } L(TM) = S\}$$

Bemerkung: Als Sprache eines Problems bezeichnet man die Menge aller Eingaben, die auf Ausgaben mit dem akzeptierenden Zustand abgebildet werden. Die Begriffe "Sprache" und "Problem" werden als gleichbedeutend angesehen.

[41] Es ist plausibel, daß ein n^{100} Algorithmus nicht effizient ist. 2^{64} Rechenschritte gelten beispielsweise zur Zeit als praktisch nicht durchführbar. Die meisten polynomiellen Algorithmen in Praxis haben aber Rechenzeitschranken, die Polynome von kleinem Grad sind. Schon Grad 5 ist selten ([Weg93]).

Bis jetzt wurden Turingmaschinen mit einem beliebigem Alphabet $\sum$ behandelt. Nun wird das Alphabet $\sum$ auf die beiden Elemente 0 und 1 beschränkt. Hierzu muß eine geeignete Kodierung festgelegt werden, die jedes Element aus $\sum$ eindeutig einer Kombination der Elemente aus $\{0, 1\}$ zuordnet.

Der folgende Satz verdeutlicht, daß bei der Berechnung der zeitlichen Komplexität nicht zwischen verschiedenen Typen von Eingaben und deren Kodierung unterschieden werden braucht.

Notation 2.5.1. *TM sei eine beliebige Turingmaschine über dem Alphabet* $\sum$. *Die von TM berechnete partielle Funktion wird als* $Res_{TM} : \sum^* \to \sum^*$ *bezeichnet.*

Satz 2.5.1. *Zu jeder polynomiell beschränkten Turingmaschine* TM_1 *mit Alphabet* $\sum_1$ *existiert eine polynomiell beschränkte Turingmaschine* TM_2 *mit Alphabet* $\{0, 1\}$, *so daß* $Res_{TM_1} = Res_{TM_2}$.

Beweis. $\sum_1$ wird mittels $c : \sum_1 \to \{0, 1\}^k$ eindeutig binär kodiert. Eine Zelle über $\sum_1$ wird durch einen Block von k Zellen über $\{0, 1\}$ simuliert. Das von TM_1 beobachtete Symbol und der Programmzustand von TM_1 werden zu Komponenten der Zustandsmenge von TM_2, d.h. $Q_2 = Q_1 \times \sum_1 \times Q_2^{'}$. Die Anzahl der Rechenschritte wird durch Einführung der Kodierung nach $\{0, 1\}$ proportional zu k vergrößert. □

2.5.1.3 Nichtdeterministische Turingmaschinen

Eine nichtdeterministische Turingmaschine NT besteht wie eine deterministische Turingmaschine auch aus einer Endlichen Kontrolle und k in Felder unterteilten Bändern. Bei einer NT kann es zu einer Konfiguration *mehrere alternative* Folgekonfigurationen geben, so daß der Rechenweg einer nichtdeterministischen Turingmaschine nicht eindeutig bestimmt ist.

Definition 2.5.13. *Eine k(mit* $k \geq 1$*)-Band nichtdeterministische Turingmaschine NT ist ein 7-Tupel* $(Q, \Sigma, \Gamma, P, q_0, B, q_f)$. *Die Bedeutung der einzelnen Tupelkomponenten ist:*

Q	*ist eine endliche Menge von Zuständen.*
Γ	*ist eine endliche Menge von erlaubten Bandsymbolen.*
B	*ist das Blank, ein Symbol aus* Γ.
$\sum$	*ist die Menge der Eingabesymbole, eine Teilmenge von* Γ, *die das Blank B nicht einschließt.*
P	$\subset Q \times \Gamma^k \times Q \times \Gamma^k \times \{L, R, S\}^k$ *ist eine Übergangsrelation.*
q_0	*aus Q ist der Anfangszustand.*
q_f	*aus Q ist der akzeptierende Zustand.*

Die Begriffe Rechenschritt, Konfiguration, Berechnung, akzeptierende Berechnung, akzeptierende Sprache sind durch die Definitionen in Abschnitt 2.5.1.1 für nichtdeterministische Maschinen erklärt. Die Übergangsfunktion einer Turingmaschine wird durch eine Relation ersetzt. Bei einer nichtdeterministischen Turingmaschine wird eine Eingabe ($\in \Sigma^*$) akzeptiert, wenn unter den allen vollständigen Berechnungen mindestens eine akzeptierende Berechnung existiert. Für eine Eingabe, die akzeptiert wird, ist die Rechenzeit einer nichtdeterministischen Turingmaschine die Länge einer kürzesten akzeptierenden Berechnung.

Bemerkung: Der Vorgang, daß eine nichtdeterministische Turingmaschine NT die Folgekonfigurationen zur Endkonfiguration mit dem akzeptierenden Zustand verfolgt, wird mit den Worten "NT errät" bezeichnet.
Nichtdeterministische Turingmaschinen sind nicht konkret baubare Maschinen, sondern sie sind eine formale Abstraktion zur Beschreibung von Rateschritten.

2.5.1.4 Nichtdeterministische polynomielle Berechenbarkeit-Die Klasse $\mathcal{NP}$

Die Klasse der Probleme, die mittels einer nichtdeterministischen Turingmaschine in polynomiell beschränkter Laufzeit gelöst werden können, wird wie folgt spezifiziert:

Definition 2.5.14. *Sei T eine Funktion $\mathbb{N}$ nach $\mathbb{R}$. Eine nichtdeterministische (k-Band) Turingmaschine NT heißt polynomiell beschränkt bzw. nichtdeterministisch polynomiell beschränkt, wenn für jede Eingabe ($\in \Sigma^*$) der Länge n mindestens eine der akzeptierenden Berechnungen eine Länge $\leq T(n)$ hat, wobei T ein Polynom ist.*

Bemerkung: Eine nichtdeterministische Turingmaschine NT benötige für eine Eingabe $x \in \sum^*$ 50 Rechenschritten auf dem kürzesten Berechnungsweg zu dem akzeptierenden Zustand. Jede Konfiguration habe dabei zwei direkte alternative Nachfolgekonfigurationen. Die Abbildung 2.29 zeigt grafisch das Aussehen der nichtdeterministischen Berechnung.

Die Klasse $\mathcal{NP}$

Definition 2.5.15. *Die Klasse $\mathcal{NP}$ ist die Menge sämtlicher durch eine nichtdeterministische polynomiell beschränkte Turingmaschine akzeptierbaren Sprachen. Formal:*

$$\mathcal{NP} = \{S \subseteq \{0,1\}^* | \exists \text{ eine nichtdeterministische polynomiell beschränkte Turingmaschine } NT \text{ mit } L(NT) = S\}$$

Jede deterministische Turingmaschine ist eine nichtdeterministische Turingmaschine. So ist die Klasse $\mathcal{P}$ eine Teilmenge der Klasse $\mathcal{NP}$.

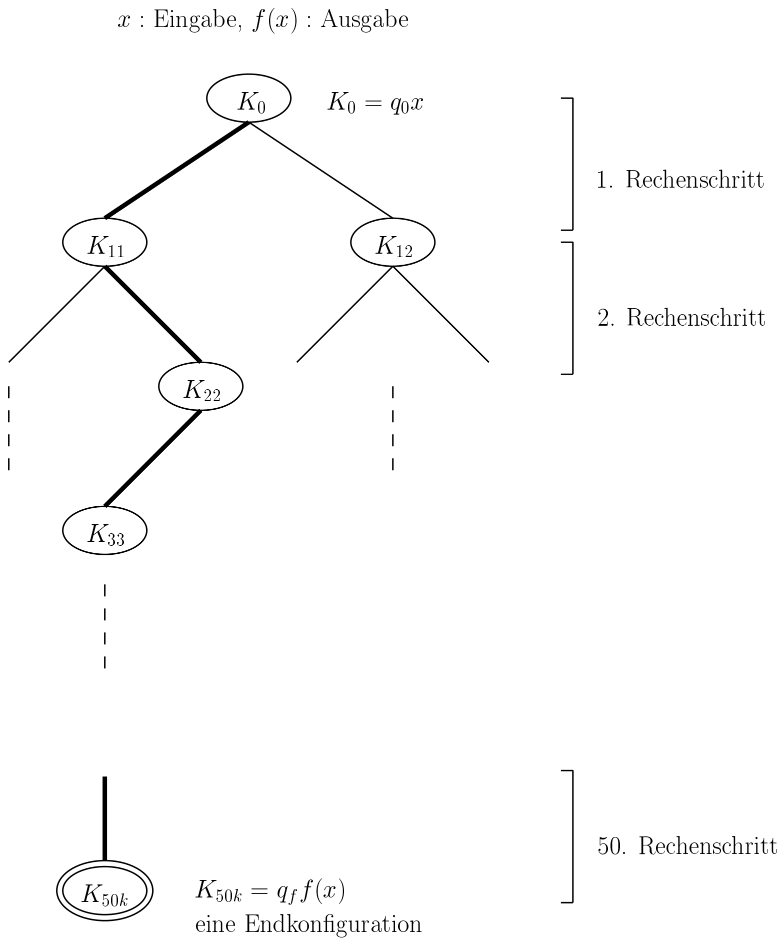


Abbildung 2.29: Grafische Darstellung einer nichtdeterministischen Berechnung

Es gibt zahlreiche Sprachen, die *vermutlich* nicht zu $\mathcal{P}$ gehören, aber von polynomiell beschränkten nichtdeterministischen Turingmaschinen akzeptiert werden können. Traveling Salesman Problem, Rucksack Problem, Hamiltonsche Wege und Clique Problem sind Beispiele für solche Probleme [Cap93], [SS76], [SH96].

2.5.2 $\mathcal{NP}$-Vollständige Sprachen (Entscheidungsprobleme)

Mittels der in diesem Abschnitt vorgestellten $\mathcal{NP}$-Vollständigkeitstheorie wird die relative Komplexität von Problemen gemessen. Als eine Grundlage wird zunächst eine asymtotische Notation eingeführt, da die Zeitkomplexität von Algorithmen (Turingmaschinen) selten genau berechnet werden kann.

2.5.2.1 O-Notation

In diesem Abschnitt wird eine asymtotische Notation, O-Notation, vorgestellt, mit deren Hilfe die Zeitkomplexität von Algorithmen (siehe Abschnitt 2.5.1.1) abgeschätzt werden kann. Die O-Notation dient zur Angabe des ungefähren Wachstumsverhaltens von Zeitkomplexität repräsentierenden Funktionen $T : \mathbb{N} \to \mathbb{R}$ bei Eingabelängen n.

Definition 2.5.16. *Eine Funktion $T : \mathbb{N} \to \mathbb{R}$ liegt in der Klasse $O(k)$einer Funktion $k : \mathbb{N} \to \mathbb{R}$, wenn eine positive reelle Zahl $c \in \mathbb{R}^+$ existiert, so daß ab einer bestimmten Zahl $n_0 \in \mathbb{N}$ für alle größeren $n \in \mathbb{N}, n \geq n_0$ die Ungleichung $T(n) \leq c \cdot k(n)$ gilt. Formal:*

$$O(k) = \{T|(\exists c > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(T(n) \leq c \cdot k(n))\} \text{ mit } T : \mathbb{N} \to \mathbb{R}.$$

"Eine Funktion $T : \mathbb{N} \to \mathbb{R}$ liegt in der Klasse $O(k)$" wird auch als "$T(n)$ ist *von der Größenordnung k*", "$T(n)$ ist $O(k)$" oder "$T(n)$ ist in $O(k)$" genannt und wird $T(n) = O(k)$ oder $T(n) \in O(k)$ geschrieben.

Beispiel 2.5.1. Sei $T(n) = 3n^2 + 5n$. In der Definition werden 8 als c und 1 als n_0 ausgewählt. Dann gilt für $n \geq n_0$:

$n(1-n) \leq 0 \Rightarrow n \leq n^2 \Rightarrow 5n \leq 5n^2 \Rightarrow 3n^2 + 5n \leq 8n^2 \Rightarrow 3n^2 + 5n \in O(n^2)$, so sagt man: $T(n)$ ist von der Größenordnung n^2 oder $T(n) = O(n^2)$.

2.5.2.2 Polynomielle Reduzierbarkeit, $\mathcal{NP}$-Vollständigkeit

In diesem Abschnitt werden die Begriffe "polynomielle Reduzierbarkeit" sowie "$\mathcal{NP}$-Vollständigkeit" eingeführt. Die Begriffe "polynomielle Reduzierbarkeit" sowie "$\mathcal{NP}$-Vollständigkeit" dienen dazu, Probleme bzgl. bestimmter Eigenschaften zu vergleichen. Die polynomielle Reduzierbarkeit eines Problems A in ein Problem B soll bedeuten, daß B "mindestens so schwierig" wie A ist [Weg96]. Ein $\mathcal{NP}$-vollständiges Problem entspricht einem der schwierigsten Probleme aus der Klasse $\mathcal{NP}$.
Zunächst werden einige Definitionen sowie Lemmata eingeführt, die einen formalen Rahmen für die Einführung der Begriffe "polynomielle Reduzierbarkeit" und "$\mathcal{NP}$-Vollständigkeit" geben sollen.

Definition 2.5.17. *Sei $f : \{0,1\}^* \longrightarrow \{0,1\}^*$. Eine deterministische Turingmaschine TM berechnet f, wenn zu jedem $X \in \{0,1\}^*$ eine vollständige Berechnung $< K_0, K_1, \cdots, K_n >$ existiert, mit ($K_0 = q_0X$) als Startkonfiguration und ($K_n = q_f f(X)$) als Endkonfiguration.*

Definition 2.5.18. *Eine Funktion $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ heißt polynomiell berechenbar, wenn es eine polynomiell beschränkte deterministische Turingmaschine gibt, die die Funktion f berechnet.*

Die Menge aller polynomiell berechenbaren Funktionen wird mit π bezeichnet. Formal:

$\pi = \{f : \{0,1\}^* \longrightarrow \{0,1\}^* |$ es gibt eine polynomiell beschränkte Turingmaschine, die f berechnet$\}$.

Lemma 2.5.3. *Aus $f, g \in \pi$ folgt $f \circ g \in \pi$*

Beweis. Die Zusammensetzung von Polynomen ist ein Polynom. □

Definition 2.5.19. *Gegeben seien zwei Sprachen L_1 und L_2. L_1 heißt polynomiell reduzierbar in L_2, wenn es ein $f \in \pi$ gibt mit $X \in L_1 \Leftrightarrow f(X) \in L_2$ für alle $X \in \{0,1\}^*$.*

Notation 2.5.2. *Daß L_1 polynomiell reduzierbar in L_2 ist, wird als $L_1 \leq L_2$ bezeichnet.*

Lemma 2.5.4. *“ $\leq$” ist reflexiv und transitiv, d.h.:*

1. *$L \leq L$ für eine beliebige Sprache L.*
2. *$L_1 \leq L_2$, $L_2 \leq L_3 \Rightarrow L_1 \leq L_3$ für beliebige Sprachen L_1,L_2 und L_3.*

Beweis. 1. Die Identitätfunktion ‘*id*’ ist eine polynomiell-Zeit-berechenbare Funktion und $x \in L \Leftrightarrow id(x) \in L$.

2. Aus $L_1 \leq L_2$ folgt, daß ein $f \in \pi$ existiert mit $X \in L_1 \Leftrightarrow f(X) \in L_2$ für alle $X \in \{0,1\}^*$. Desweiteren folgt aus $L_2 \leq L_3$, daß ein $g \in \pi$ existiert mit $Y \in L_2 \Leftrightarrow g(Y) \in L_3$ für alle $Y \in \{0,1\}^*$. $g \circ f$ ist ein Element von π mit $X \in L_1 \Leftrightarrow g \circ f(X) \in L_3$ für alle $X \in \{0,1\}^*$. □

Nun wird der Kernbegriff “$\mathcal{NP}$-Vollständigkeit” definiert.

Definition 2.5.20. *Eine Sprache L ist vollständig in $\mathcal{NP}$ bzw. $\mathcal{NP}$-vollständig bzgl. $\leq$, falls*

1. *$L \in \mathcal{NP}$.*
2. *für alle $L^{'} \in \mathcal{NP}$ gilt $L^{'} \leq L$.*

Wie in Abschnitt 2.5.1.4 erwähnt, ist es für Probleme wie das Traveling Salesman Problem, Rucksack Problem, Hamiltonsche Wege und Clique Problem nicht bekannt, ob sie in $\mathcal{P}$ sind oder nicht. Sie sind aber in $\mathcal{NP}$. Desweiteren kann für sie gezeigt werden, daß sie $\mathcal{NP}$-vollständig sind [Cap93], [SS76]. Es gibt zahlreiche weitere $\mathcal{NP}$-vollständige Probleme [Weg93].

Ein $\mathcal{NP}$-vollständiges Problem S_1 ist auf ein $\mathcal{NP}$-vollständiges Problem S_2 polynomiell reduzierbar und umgekehrt. Dieser Sachverhalt erlaubt es, viele $\mathcal{NP}$-vollständige Probleme als gleich schwierig zu klassifizieren.
Wenn für ein beliebiges $\mathcal{NP}$-vollständiges Problem S ein effizienter Algorithmus gefunden würde, der entscheidet, ob eine Eingabe ($\in \sum^*$) in S ist oder nicht, wären wegen der polynomiellen Reduzierbarkeit für alle $\mathcal{NP}$ Probleme ein effizienter Algorithmus gefunden.

2.5.2.3 Die $\mathcal{NP}$-Vollständigkeit des aussagenlogischen Erfüllbarkeitsproblems

Es wird nun eine kurze Einführung in die Aussagenlogik gegeben[42]. Auf Basis der Aussagenlogik wird das $\mathcal{NP}$-vollständige Erfüllbarkeitsproblem formal definiert. Zunächst wird die Syntax für aussagenlogische Formeln vorgestellt.

Definition 2.5.21. *(Syntax der Aussagenlogik)*
$p_1, p_2, p_3, \cdots$ *seien aussagenlogische Variablen, die auch atomare Formeln genannt werden.*
Formeln werden durch folgenden induktiven Prozeß definiert.

1. *Jede Variable ist eine Formel.*
2. *Für jede Formel A ist $\neg A$ eine Formel.*
3. *Für alle Formeln A und B sind $(A \wedge B)$ und $(A \vee B)$ Formeln.*

Die zwei binären Operationen $\wedge$ und $\vee$ werden als *Konjunktion* und *Disjunktion*, die einstellige Operation $\neg$ als *Negation* bezeichnet.
Die Bindungspriorität der Operationen ist vom stärksten zum schwächsten: $\neg$, $\wedge$ und $\vee$.

Im folgenden wird die Semantik für aussagenlogische Formeln vorgestellt. Als Semantik einer aussagenlogischen Formel wird ein Wahrheitswert angenommen.

Definition 2.5.22. *(Semantik der Aussagenlogik)*

Die Semantik einer beliebigen aussagenlogischen Formel wird durch eine Interpretation gegeben, die aus der Menge $\{0, 1\}$ und einer Belegung v besteht. Die

[42] Eine ausführliche Einführung in die Aussagenlogik gibt [Sch92].

Elemente der Menge $\{0,1\}$ heißen Wahrheitswerte, wobei 0 falsch und 1 wahr bedeutet. Eine Belegung v spezifiziert den Wahrheitswert (entweder 0 oder 1) einer aussagenlogischen Formel. Sei eine Formel F gegeben und V sei die Menge aller in F vorkommenden aussagenlogischen Variablen. Eine Belegung v ist eine Funktion $v : V \longrightarrow \{0,1\}$. Desweiteren wird eine Belegung durch die folgenden Wahrheitstafeln auf ganze Formeln erweitert. Hierbei seien A und B aussagenlogische Formeln.

1. *Der Operator $\neg$ kehrt den Wahrheitswert einer Formel um.*

$v(A)$	$v(\neg A)$
0	1
1	0

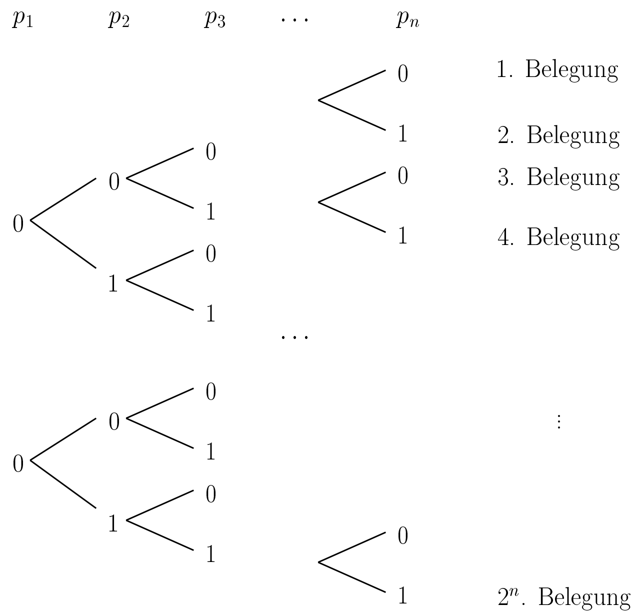
2. *Der Wahrheitswert von $(A \wedge B)$ ist gleich 1, wenn sowohl A als auch B den Wert 1 hat und gleich 0 sonst.*

$v(A)$	$v(B)$	$v(A \wedge B)$
0	0	0
0	1	0
1	0	0
1	1	1

3. *Der Wahrheitswert von $(A \vee B)$ ist gleich 1, wenn A oder B den Wert 1 hat und gleich 0 sonst.*

$v(A)$	$v(B)$	$v(A \vee B)$
0	0	0
0	1	1
1	0	1
1	1	1

Bemerkung: Eine Belegung der Variablen, die in einer Formel F vorkommen, bestimmt den Wahrheitswert der Formel F. So ist der Wert einer Formel F nur abhängig von der Belegung der in F vorkommenden Variablen. Falls in F n Variablen $p_1, p_2, \cdots, p_n$ vorkommen, existieren insgesamt 2^n verschiedene Belegungen für die Variablen der Formel F. Dies zeigt die folgende Abbildung:



Beispiel 2.5.2. $F := (p_1 \vee \neg p_2) \wedge (\neg p_1 \vee p_3)$ Für die Belegung v mit $v(p_1) = 0, v(p_2) = 1$ und $v(p_3) = 0$ ist der Wert von F gleich 0, d. h. $v(F) = 0$. Dies zeigt die folgende Tabelle:

p_1	p_2	$p_1 \vee \neg p_2$	$\neg p_1 \vee p_3$	$(p_1 \vee \neg p_2) \wedge (\neg p_1 \vee p_3)$
0	1	0	1	0

Definition 2.5.23. *Gegeben sei eine Formel A. Existiert eine Belegung v, die $v(A) = 1$ zuordnet, so heißt A erfüllbar.*

Definition 2.5.24. *Eine aussagenlogische Variable sowie die Negation einer aussagenlogischen Variable heißt ein Literal. Eine Formel, die in der Form von einer endlichen Disjunktionen $q_{i1} \vee q_{i2} \vee \cdots \vee q_{ik}$ von Literalen $q_{ij} (j = 1 \cdots, k)$ ist, heißt ein Klausel. Eine Formel F ist in konjunktiver Normalform, falls sie eine endliche Konjunktion von Klauseln ist:*

$$F = (\wedge_{i=1}^{m} (\vee_{j=1}^{n_i} q_{ij}))$$

Das Erfüllbarkeitsproblem

Das Erfüllbarkeitsproblem besteht darin, zu entscheiden, ob eine gegebene aussagenlogische Formel in konjunktiver Normalform erfüllbar ist.

Die Sprache E

Definition 2.5.25. *E ist die Menge der erfüllbaren Formeln in konjunktiver Normalform.*

Satz 2.5.2. *[Coo71], [SS76], [SH96]*

Das Erfüllbarkeitsproblem ist $\mathcal{NP}$-vollständig[43].

Es existiert ein Verfahren (Algorithmus), um die Erfüllbarkeit einer Formel aus E mit n Variablen zu entscheiden. Im Rahmen des Verfahrens wird für alle möglichen 2^n Wahrheitsbelegungen der Wahrheitswert berechnet. Die Formel ist erfüllbar, genau dann wenn sie bei mindestens einer Belegung den Wahrheitswert 1 annimmt. Die exponentielle Anzahl der möglichen Wahrheitsbelegungen führt hierbei dazu, daß die Berechnungskomplexität dieses Entscheidungsverfahrens exponentiell wird. Auf der anderen Seite ist mit Sicherheit das Berechnen des Wertes der Formel bei einer festgelegten Belegung in polynomieller Zeit durchführbar. Nichtdeterministische Turingmaschinen dienen zur Spezifikation der mathematischen Betrachtungen dieser beiden zeitlichen Komplexitätsmaße.

Eine nichtdeterministische Turingmaschine kann konstruiert werden, die die Sprache E erkennt. Desweiteren ist E eine *vollständige Sprache* in $\mathcal{NP}$. Der Beweis hierfür befindet sich in [Coo71], [SS76], [SH96].

2.5.2.4 Nachweis der $\mathcal{NP}$-Vollständigkeit einer Sprache

Wenn man eine $\mathcal{NP}$-vollständige Sprache L hat, kann man die $\mathcal{NP}$-Vollständigkeit einer weiteren $\mathcal{NP}$-Sprache S mittels Anwendung der Transitivitätsregel der polynomiellen Zeit-Reduzierbarkeit beweisen. S ist $\mathcal{NP}$-vollständig genau dann, wenn die folgenden zwei Eigenschaften gelten:

1. $S \in \mathcal{NP}$
2. $L \leq S$.

2.5.2.5 Die $\mathcal{NP}$-Vollständigkeit der Sprache "SPEZIELLE ERFÜLLBARKEIT (SE)"

Eine weitere $\mathcal{NP}$-vollständige Sprache "SPEZIELLE ERFÜLLBARKEIT (SE)" wird nun vorgestellt. Aus der $\mathcal{NP}$-Vollständigkeit der in Abschnitt 2.5.2.3 vorgestellten Sprache E, $E \leq SE$, und $SE \in \mathcal{NP}$ wird auf die $\mathcal{NP}$-Vollständigkeit von SE geschlossen. Die $\mathcal{NP}$-Vollständigkeit von SE wird in Kapitel 9 zum Beweis der $\mathcal{NP}$-Vollständigkeit der Sprache $\mathcal{RSV}$ verwendet werden.

Die Sprache "SPEZIELLE ERFÜLLBARKEIT" ([SS76])

$C_1 \wedge C_2 \wedge \cdots \wedge C_m$ sei eine aussagenlogische Formel in konjunktiver Normalform mit den Variablen $p_1, p_2, \cdots, p_n$. Hierbei hat C_i die Form $x_{i1} \vee \cdots \vee x_{in_i}$ mit

[43] Das Erfüllbarkeitsproblem wurde als das erste $\mathcal{NP}$-Vollständigkeitsproblem von Stephen Cook in 1971 bewiesen.

$x_{ik} \in \{p_1, \cdots p_n, \neg p_1, \cdots \neg p_n\}$. Die Menge, die aus $x_{i1}, \cdots, x_{in_i}$ besteht, wird als *Okkurenzmenge* von C_i ($Occ\, C_i := \{x_{ik} \mid k \leq n_i\}$) bezeichnet. *SPEZIELLE ERFÜLLBARKEIT* (SE) besteht aus denjenigen $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ in konjunktiver Normalform, für die keine der Disjunktionen C_i eine aussagenlogische Variable mehrmals enthält ($h \neq k \Rightarrow (x_{ih} \neq x_{ik}\, und\, x_{ih} \neq \neg x_{ik})$) und in denen zu verschiedenen Indizes Disjunktionen mit verschiedenen Okkurenzmengen gehören ($i \neq j \Rightarrow Occ\, C_i \neq Occ\, C_j$), und mit der Eigenschaft: $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ ist erfüllbar.

Im folgenden wird die $\mathcal{NP}$-Vollständigkeit der Sprache SE gezeigt. Zum Nachweis der Vollständigkeit der Sprache genügt es, $SE \in \mathcal{NP}$ und $E \leq SE$ zu zeigen.

SE ist $\mathcal{NP}$-vollständig

Satz 2.5.3. *SE liegt in $\mathcal{NP}$.*

Beweis. [SS76]Eine nichtdeterministische Turingmaschine NT über $\{0,1\}^*$, die genau die Wörter aus SE akzeptiert, kann wie folgt konstruiert werden:
Für eine gegebene Eingabe $X \in \{0,1\}^*$ prüft NT, ob X die Kodierung einer Formel in konjunktiver Normalform ist. Falls nein, stoppt sie (und verwirft). Im anderen Fall bewertet NT die einzelnen Literale sowie die Wahrheitswerte der einzelnen Disjunktionen auf nichtdeterministische Weise. NT errät eine Belegung der Variablen, bei der mindestens ein Literal jeder Disjunktion zu dem Wahrheitswert 1 führt.
Falls X in SE ist, existiert mit Sicherheit mindestens eine solche Belegung, da X die Kodierung einer erfüllbaren Formel ist. NT endet dabei im Zustand q_f. Ansonsten hält NT nicht akzeptierend an.

Falls umgekehrt NT im Zustand q_f stoppt, so existiert eine Belegung, bei der jede Disjunktion den Wert 1 annimmt. Damit nimmt auch die Formel den Wert 1 an; also ist X in SE. □

Satz 2.5.4. *$E \leq SE$.*

Beweis. [SS76] Bei jedem $w \in \{0,1\}^*$, das nicht in E liegt, wird $f(w) = w$ definiert oder so definiert, daß $f(w) \notin SE$ ist.

$E \leq SE$ wird mit der Zuordnung $B_1 \wedge \cdots \wedge B_l \longmapsto f(B_1 \wedge \cdots \wedge B_l) = C_1 \wedge \cdots \wedge C_m$ gezeigt; diese entsteht durch:

- Weglassen von $B_i = x_{i1} \vee \cdots \vee x_{in_i}$, falls $x_{ih} = \neg x_{ik}$ für ein h, k ist; Komplexität: $O(n^2)$.
- Streichen von überflüssigen x_{ik} in den verbleibenden B_i, so daß auch $h \neq k \Rightarrow x_{ih} \neq x_{ik}$ gilt; Komplexität: $O(n^2)$.

- Weglassen von überflüssigen B_i, wenn mehrere die gleiche Okkurenzmenge besitzen; Komplexität: $O(n^2)$.

Schließlich gilt: $\forall w \in \{0,1\}^* \ w \in E \Leftrightarrow f(w) \in SE$.

f liegt in π, da die gesamte Komplexität $T(n) = O(n^2)$ ist. $\square$

Kapitel 3

Die terminologische Schedulingsprache $\mathcal{RSV}$

In diesem Kapitel werden Syntax sowie Semantik *einer logik-basierten Schedulingsprache* $\mathcal{RSV}$ definiert, mit deren Hilfe das $\mathcal{RSV}$-Problem terminologisch dargestellt und gelöst werden kann. Dazu wird jedem Ausdruck aus $\mathcal{RSV}$ mittels einer Interpretationsfunktion eine Menge aktiver Ablaufpläne zugeordnet[1]. Syntaktisch unterschiedliche Ausdrücke aus $\mathcal{RSV}$ können auf diese Weise auf diesselbe Menge von Ablaufplänen abgebildet werden. Werden zwei Ausdrücke auf dieselbe Menge von Ablaufplänen abgebildet, so werden sie als äquivalent angesehen. Jede Menge derart als äquivalent bewerteter Ausdrücke beinhaltet stets einen Ausdruck mit einer ausgezeichneten Form, bei der auf oberster Ebene ein einzelner Operator '**xor**' vorkommt, der mehrere '**xor**' freie Subausdrücke miteinander verbindet. Ein Ausdruck, der diese ausgezeichnete Form aufweist, wird als "normalisiert" bezeichnet. Die '**xor**' freien Subausdrücke werden dabei als *reduzierte Aktionen* bezeichnet.
In einem späteren Kapitel wird für normalisierte Ausdrücke ein Verfahren zur Lösung des $\mathcal{RSV}$-Problems vorgestellt werden. Durch die Kombination von Normalisierung und des später vorgestellten Lösungsverfahren kann jedes beliebige $\mathcal{RSV}$-Problem gelöst werden. Gelöst heißt dabei, daß zu jedem Ausdruck aus $\mathcal{RSV}$ eine Menge optimaler Ablaufpläne ermittelt wird.

Zu Beginn des Kapitels wird die Schedulingsprache $\mathcal{RSV}$ definiert. Anschließend erfolgt die Definition eines Kalküls, mit dessen Hilfe zu jedem Ausdruck aus $\mathcal{RSV}$ ein eindeutiger semantisch äquivalenter Ausdruck in normalisierter Form ermittelt werden kann. Dies geschieht durch schrittweise Transformation auf Ausdrucksebene entsprechend den Ableitungsregeln des Kalküls.
Im Anschluß wird eine Sprache $\mathcal{RSVX}$ definiert, deren syntaktische Basis die

[1] Wie in Abschnitt 2.1 betrachtet, ist die Menge aller zu einem Ablaufplanungsproblem gehörigen aktiven Ablaufpläne endlich.

Menge aller normalisierten Ausdrücke bildet. Ferner wird ein Verfahren vorgestellt, das dazu geeignet ist, Ausdrücke der Schedulingsprache $\mathcal{RSVX}$ grafisch darzustellen. Diese grafische Darstellung ermöglicht eine einfache Erkennung *alternativer* reduzierter Aktionen sowie der Struktur der reduzierten Aktionen.

3.1 Die Schedulingsprache $\mathcal{RSV}$

Ausdrücke der logikbasierten terminologischen Sprache $\mathcal{RSV}$ werden basierend auf einer Menge atomarer Elemente unter Benutzung von drei Operatoren konstruiert. Da jeder *Ausdruck* aus der Sprache $\mathcal{RSV}$ als eine *Aktion* aufgefaßt werden kann, betrachten wir beide Begriffe im weiteren als austauschbar. Die atomaren Elemente einer Aktion aus $\mathcal{RSV}$ bildet eine Menge gegebener Grundaktionen. Grundaktionen spezifizieren atomare Vorgänge im Rahmen einer Projektplanung, d.h. Vorgänge die nicht weiter zerlegt werden können. Durch wiederholte Verknüpfung von Aktionen mittels der drei zuletzt erwähnten Operatoren können ausgehend von einer Menge von Grundaktionen komplexe Projekte terminologisch dargestellt werden.

3.1.1 Die Syntax der Schedulingsprache $\mathcal{RSV}$

Die Syntax der Sprache $\mathcal{RSV}$ wird wie folgt definiert:

Definition 3.1.1. *Das Vokabular von* $\mathcal{RSV}$ *besteht aus drei disjunkten Mengen von Symbolen. Diese Mengen sind:*

- *Eine Menge von Grundaktionen. Jede Grundaktion besteht aus einem (eindeutigen) Prädikatensymbol und zwei Konstanten. Ein Prädikatensymbol entspricht dem Namen einer Grundaktion und ist zweistellig (besitzt zwei Argumente). Die Argumente eines Prädikats sind Konstanten. Das erste Argument eines Prädikatensymbols bezeichnet eine Ressource, das zweite eine zeitliche Dauer.* $\mathbb{N}$ *sei die Menge natürlicher Zahlen. Wenn* P *ein Prädikatensymbol ist,* r *und* t $(\in \mathbb{N})$ *Konstanten sind, dann heißt* $P(r,t)$ *eine Grundaktion*[2] *. Der Name jeder Grundaktion ist eindeutig zu wählen.*
- *Einer Menge von drei strukturellen Symbole '***seq***', '***xor***' und '***pll***', die als Operatoren bezeichnet werden.*
- *Die beiden Klammersymbole* $(,)$.

Die Beschreibung aller Aktionen aus $\mathcal{RSV}$ *wird wie folgt induktiv gegeben:*

[2] Jede Grundaktion entspricht einem Atom in der Prädikatenlogik.

1. *Jede Grundaktion ist eine Aktion.*

2. *Seien $A_1, A_2, \cdots, A_k$ Aktionen, dann sind*

$$(\mathbf{seq}\, A_1, A_2, \cdots, A_k),$$
$$(\mathbf{xor}\ A_1, A_2, \cdots, A_k),$$
$$(\mathbf{pll}\, A_1, A_2 \cdots, A_k)$$

   Aktionen.

Die Operatoren '**seq**', '**xor**' und '**pll**' sind Aktionen-konstruierend auf Aktionen und haben die folgenden Bedeutungen:

- '**seq**' : Dieser Operator dient zur Spezifikation der *sequentiellen* Abarbeitung einer bzw. mehrerer Aktionen, die durch die Argumente gegeben werden. So spezifiziert $(\mathbf{seq}\, A_1, A_2, \cdots, A_k)$ aus Definition 3.1.1, daß die Aktionen $A_1, A_2, \cdots, A_k$ nacheinander durchzuführen sind (Präzedenzbeschränkung).

- '**xor**' : Dieser Operator dient zur Spezifikation der Abarbeitung einer beliebig *ausgewählten* Aktion aus einer Menge von Aktionen. So spezifiziert $(\mathbf{xor}\ A_1, A_2, \cdots, A_k)$ aus Definition 3.1.1, daß genau eine der Aktionen $A_1, A_2 \cdots, A_k$ durchgeführt werden muß.

- '**pll**' : Dieser Operator dient zur Spezifikation der *parallelen* Abarbeitungsmöglichkeit einer bzw. mehrerer Aktionen. Dabei ist die Nebenbedingung einzuhalten, daß zwischen den einzelnen Aktionen kein Ressourcenkonflikt besteht. So spezifiziert $(\mathbf{pll}\, A_1, A_2 \cdots, A_k)$ aus Definition 3.1.1, daß die Aktionen $A_1, A_2 \cdots, A_k$ parallel durchgeführt werden können.

Ein Ausdruck aus $\mathcal{RSV}$ beinhaltet implizit folgende semantische Eigenschaften:

- Jede Grundaktion ist mit einer bestimmten Ressource und einer bestimmten Zeit verbunden. Ressource und zeitliche Dauer werden zur Erfüllung der Grundaktion benötigt.

- Es wird vorausgesetzt, daß von jeder Ressource jeweils nur *eine* Einheit während der gesamten Projektdauer zur Verfügung gestellt werden kann und jede Ressource zu einer Zeit von maximal einer Grundaktion belegt werden kann. Sie wird erst nach Beendigung der Grundaktion wieder freigegeben (Ressourcenbeschränkung). Diese Ressourcenbeschränkung kann Ressourcenkonflikte verursachen, die eine parallele Bearbeitung mehrerer Aktionen verhindern.

- Eine durchzuführende Grundaktion kann nicht unterbrochen werden.

Im folgenden Abschnitt werden Planungsausschnitte (Vorgänge sowie Teilprojekte) eines zeitlich und sachlich begrenzten Beispielprojekts, das ein Bauvorhaben wiedergibt, mittels $\mathcal{RSV}$-Ausdrücken dargestellt.

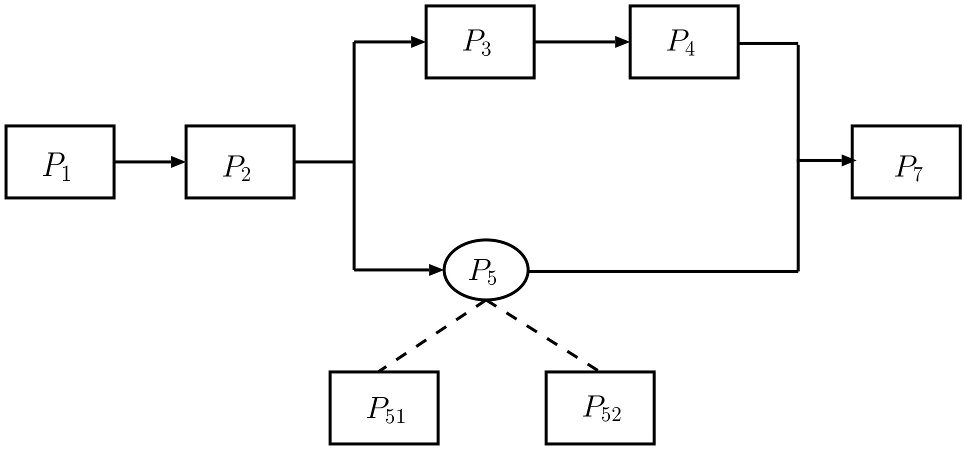
3.1.1.1 Beispiel

Die folgende Vorgangsliste (Grundaktionsliste) sowie Anordnungsbeziehungen seien im Rahmen eines Bauvorhabens gegeben[3] . Hierbei gebe es für den Vorgang 'Fenster und Türensetzen' 2 alternative Auswahlmöglichkeiten:

Grundaktionen		Vorgänger	Nachfolger
'Fundamente und Wände errichten $(a, 20)$'	$:= P_1(a, 20)$	_	P_2
'Elektrische Anschlüsse $(b, 15)$'	$:= P_2(b, 15)$	P_1	P_3, P_{51} oder P_{52}
'Innenputz $(c, 15)$'	$:= P_3(c, 15)$	P_2	P_4
'Installation und Heizung $(c, 16)$'	$:= P_4(c, 16)$	P_3	_
'Fenster und Türen setzen 1 $(c, 3)$'	$:= P_{51}(c, 3)$	P_2	_
'Fenster und Türen setzen 2 $(d, 5)$'	$:= P_{52}(d, 5)$	P_2	_
'Tapezieren $(e, 10)$'	$:= P_6(e, 10)$	P_4, P_{51} oder P_{52}	_

Wie in Abschnitt 2.2.3 erläutert, kann das zu dieser Tabelle gehörige Projekt mittels eines hierarchischen Netzplans dargestellt werden. Der hierarchische Netzplan, der die komplette Projektablaufstruktur abbildet, hat folgendes Aussehen:

[3] Das betrachtete Bauvorhaben entspricht einer leicht variierten Form eines Beispiels aus Schwarze [Sch94] bzw. Ohse [Ohs98].



Teilprojekte sowie die komplette Projektablaufstruktur können mittels Aktionen in $\mathcal{RSV}$ wie folgt repräsentiert werden:

Den Vorgang 'Fundamente und Wände errichten' spezifiziert die Grundaktion

$$P_1(a, 20)$$

Die Aktion

$$(\mathbf{xor}\ P_{51}(c, 3), P_{52}(d, 5)) \tag{1}$$

beschreibt die Auswahl zwischen den zwei Alternativen für "Fenster und Türen setzen". Gemäß der gegebenen Anordnungsbeziehungen sind die beiden Vorgänge 'Innenputz' ($P_3(c, 15)$) und 'Installation und Heizung' ($P_4(c, 16)$) nacheinander auszuführen, was folgende Aktion wiedergibt:

$$(\mathbf{seq}\ P_3(c, 15), P_4(c, 16)) \tag{2}$$

Die Aktion (2) und die eine Alternativität beinhaltende Aktion 'Fenster und Türensetzen' können parallel durchgeführt werden und wir erhalten:

$$(\mathbf{pll}\ (\mathbf{seq}\ P_3(c, 15), P_4(c, 16)), (\mathbf{xor}\ P_{51}(c, 3), P_{52}(d, 5))) \tag{3}$$

Nach der Durchführung des Vorganges 'Elektrische Anschlüsse' ($P_2(b, 15)$) ist die obige Aktion (3) durchzuführen:

$$(\mathbf{seq}\ P_2(b, 15), (\mathbf{pll}\ (\mathbf{seq}\ P_3(c, 15), P_4(c, 16)), (\mathbf{xor}\ P_{51}(c, 3), P_{52}(d, 5)))) \tag{4}$$

Die komplette Projektablaufstruktur wird durch folgenden als $\mathcal{RSV}$ -Ausdruck wiedergegeben:

$$\begin{array}{ll}(\mathbf{seq} & P_1(a, 20),\quad P_2(b, 15), (\mathbf{pll}\ (\mathbf{seq}\ P_3(c, 15), P_4(c, 16)), (\mathbf{xor}\ P_{51}(c, 3), P_{52}(d, 5))), \\ & P_6(e, 10))\end{array} \tag{5}$$

3.1.2 Die Semantik der Schedulingsprache $\mathcal{RSV}$

In einer Aktion aus $\mathcal{RSV}$ kann die Möglichkeit der Wahl zwischen unterschiedlichen Projektabläufen spezifiziert werden, indem alternative Vorgänge (Grundaktionen) oder Teilabläufe (Aktionen) eines Produktions- bzw. eines Projektabschnitts mit dem Operator '**xor**' verknüpft werden. Wir werden nun sehen, daß Variationsmöglichkeiten im Bezug auf die Position des Operators '**xor**' existieren können. Dies verdeutlicht die Aktion

$$(\mathbf{pll}\,(\mathbf{seq}\,P_3(c,15),P_4(c,16)),(\mathbf{xor}\,P_5(c,3),P_6(d,5))) \tag{6}$$

des Beispiels aus 3.1.1.1. Die obige Aktion beinhaltet die beiden alternativen Projektabläufe

$$\begin{array}{l}(\mathbf{pll}\,(\mathbf{seq}\,P_3(c,15),P_4(c,16)),P_5(c,3)) \quad \text{und} \\ (\mathbf{pll}\,(\mathbf{seq}\,P_3(c,15),P_4(c,16)),P_6(d,5))\end{array} \tag{7}$$

und kann daher alternativ auch mittels

$$\begin{array}{ll}(\mathbf{xor} & (\mathbf{pll}\,(\mathbf{seq}\,P_3(c,15),P_4(c,16)),P_5(c,3)) \\ & (\mathbf{pll}\,(\mathbf{seq}\,P_3(c,15),P_4(c,16)),P_6(d,5)))\end{array} \tag{8}$$

spezifiziert werden. Bei der letzteren Darstellung kommt der Operator '**xor**' einmalig auf oberster Ebene vor und ist damit aus dem inneren des Ausdrucks an die äußerste linke Position bewegt worden. Die Menge aller zu der Aktion (6) gehörigen (aktiven) Ablaufpläne sowie die Menge aller zu der Aktion (8) gehörigen Ablaufpläne sind identisch, was sich einfach überprüfen läßt. Als Folge davon betrachten wir die beiden Aktionen (6) und (8) als semantisch äquivalent.

In diesem Abschnitt wird eine Semantik für Aktionen aus $\mathcal{RSV}$ vorgestellt. Als Semantik einer Aktion aus $\mathcal{RSV}$ werden alle zu einer Aktion gehörigen (aktiven) Ablaufpläne angenommen.
Aus der semantischen Äquivalenz zweier Aktionen folgt, daß ein Ablaufplan, der für eine Aktion zeitlich optimal ist, auch für die andere semantisch äquivalente Aktion optimal ist und umgekehrt. Im folgenden soll dies anhand der beiden zuletzt betrachteten Aktionen (6) und (8) verdeutlicht werden:
Bei der Aktion (6) besteht eine Auswahlmöglichkeit, zwischen den Grundaktionen $P_5(c,3)$ und $P_6(d,5)$ im inneren des Ausdrucks, wo der Operator '**xor**' vorkommt. Bei der Auswahl von $P_5(c,3)$ entsteht ein Ressourcenkonflikt, der eine Verzögerung der Anfertigung des Projektes verursacht. Es existieren hierbei insgesamt *drei* verschiedene Auflösungsmöglichkeiten des Ressourcenkonflikts. P_5 kann

1. zum ersten vor P_3

2. zum zweiten zwischen P_3 und P_4

3. und zum dritten nach P_4 durchgeführt werden.

Für jede Auflösungsmöglichkeit ergibt sich ein Ablaufplan (eine *ressourcenkonfliktfreie* Prozeßvariante). Alle *drei* so erhaltenen ressourcenkonfliktfreien Prozeßvarianten benötigen eine Gesamtlaufzeit von 34 Zeiteinheiten.
Im Gegensatz dazu entsteht bei der Auswahl von $P_6(d, 5)$ kein Ressourcenkonflikt und die beiden Aktionen $(\mathbf{seq}\ P_3(c, 15), P_4(c, 16))$ sowie $P_6(d, 5)$ können parallel bearbeitet werden. Als Gesamtlaufzeit erhalten wir bei dieser Alternative 31 Zeiteinheiten. Der zugehörige Ablaufplan hat das Aussehen

$$[((0, P_3), c, 15), ((15, P_4), c, 16), ((0, P_6), d, 5)]$$

Dabei sind die einzelnen Tripel $((s, P), r, d)$ des Ablaufplans wie folgt zu lesen:

- s entspricht der Startzeit der Grundaktion P
- r entspricht der von P benötigten Ressource
- d entspricht der von P benötigten zeitlichen Dauer

Eine einfache Überlegung zeigt nun, daß der obige Ablaufplan gleichermaßen für die beiden Aktionen (6) und (8) eine zeitlich optimale, ressourcenkonfliktfreie Prozeßvariante darstellt.

Ausdrücke (in der Form von (8)), bei denen '**xor**'-freie Terme durch ein einzelnes '**xor**'-Operator verknüpft sind, bezeichnen wir als *normalisiert.* Bei normalisierten Ausdrücken können wir zur Bestimmung der optimalen Ablaufpläne die Argumente eines einzelnen Operators '**xor**' getrennt betrachten. Innerhalb der Argumente von '**xor**' kommen dabei *nur* die Operatoren '**seq**' und '**pll**' vor.
In Abschnitt 3.2 wird ein Verfahren vorgestellt werden, das es erlaubt, Aktionen, die wie (6) aussehen, in die *normalisierte* Form von (8) zu transformieren. Dazu verleihen wir der Sprache $\mathcal{RSV}$ eine Semantik. Die Semantik der Sprache $\mathcal{RSV}$ wird wie folgt formal definiert:

Definition 3.1.2. *Die modell-theoretische Semantik aller Aktionen aus $\mathcal{RSV}$ wird durch eine Interpretation $\mathcal{I}$ gegeben, die aus der Menge $\mathcal{D}$ (die Domäne von $\mathcal{I}$) und einer Funktion $\cdot^{\mathcal{I}}$ (die Interpretationsfunktion von $\mathcal{I}$) besteht. Die Menge $\mathcal{D}$ besteht aus allen aktiven Ablaufplänen, die aus $\mathcal{RSV}$-Aktionen hergeleitet werden können. Die Interpretationsfunktion $\cdot^{\mathcal{I}}$ bildet jede Aktion A aus $\mathcal{RSV}$ auf eine Teilmenge von $\mathcal{D}$ ab, die aus allen zu A gehörigen (aus A herleitbaren) aktiven Ablaufplänen besteht.*

Die folgenden Beispiele sollen zum besseren Verständnis der obigen Definition beitragen. Ein Ablaufplan wird hierbei wie bei dem vorherigen Beispiel mittels einer Liste von Tripeln dargestellt. Alle hierbei vorkommenden Grundaktionen seien zu $\mathcal{RSV}$ gehörig:

Beispiel 3.1.1. Gegeben sei die Aktion $(\mathbf{seq} P_1(a,2), (\mathbf{xor} Q_1(a,3), Q_2(b,4)), P_2(b,2))$. Bei der Erfüllung letzterer Aktion besteht beim zweiten Argument des Operators **seq** eine Auswahlmöglichkeit zwischen den beiden Grundaktionen Q_1 und Q_2. Als Folge können bei dieser Aktion zwei unterschiedliche Ablaufpläne hergeleitet werden. Wie dies geschieht zeigt die folgende Gleichung:

$$\begin{aligned}
&(\mathbf{seq}\, P_1(a,2), (\mathbf{xor}\, Q_1(a,3), Q_2(b,4)), P_2(b,2))^{\mathcal{I}} \\
&= (\mathbf{seq}\, P_1(a,2), Q_1(a,3), P_2(b,2))^{\mathcal{I}} \cup (\mathbf{seq}\, P_1(a,2), Q_2(b,4), P_2(b,2))^{\mathcal{I}} \\
&= \{[((0,P_1),a,2),\ ((2,Q_1),a,3),\ ((5,P_2),b,2)]\} \\
&\quad \cup \{[((0,P_1),a,2),\ ((2,Q_2),b,4),\ ((6,P_2),b,2)]\} \\
&= \{[((0,P_1),a,2),\ ((2,Q_1),a,3),\ ((5,P_2),b,2)], \\
&\quad [((0,P_1),a,2),\ ((2,Q_2),b,4),\ ((6,P_2),b,2)]\}
\end{aligned}$$

Bei obigem Beispiel war der Erhalt zweier Ablaufpläne eine Folge des Vorkommens des Operators **xor** innerhalb der gegebenen Aktion. Das nun folgende Beispiel verdeutlicht das Entstehen mehrerer Ablaufpläne als Folge eines Ressourcenkonflikts.

Beispiel 3.1.2. Gegeben sei die Aktion $(\mathbf{seq} P_1(a,2), (\mathbf{pll} P_2(b,3), P_3(b,1)), P_4(c,2))$. Bei letzterer Aktion sind die beiden Grundaktionen P_2 und P_3 an einem Ressourcenkonflikt beteiligt, der eine parallele Abarbeitung beider Grundaktionen verhindert. So können je nachdem, in welcher Anordnung die beiden Grundaktionen sequentialisiert werden, werden zwei unterschiedliche Ablaufpläne erhalten:

$$\begin{aligned}
&(\mathbf{seq}\, P_1(a,2), (\mathbf{pll}\, P_2(b,3), P_3(b,1)), P_4(c,2))^{\mathcal{I}} \\
&= \{[((0,P_1),a,2),\ ((2,P_2),b,3),\ ((5,P_3),b,1),\ ((6,P_4),c,2)], \\
&\quad [((0,P_1),a,2),\ ((2,P_3),b,1),\ ((3,P_2),b,3),\ ((6,P_4),c,2)]\}
\end{aligned}$$

3.1.3 Schedulingaxiome

Wie bereits erwähnt existieren Aktionen aus $\mathcal{RSV}$, die zwar syntaktisch unterschiedlich aussehen, aber semantisch äquivalent sind. Um die semantische Äqui-

valenz zwischen zwei Aktionen aus $\mathcal{RSV}$ darzustellen, wird außer den Operatorsymbolen '**seq**', '**xor**' und '**pll**' im Kontext der Sprache $\mathcal{RSV}$ ein weiteres strukturelles Symbol $\doteq$ definiert.
Der Ausdruck $C \doteq D$ beschreibt, daß die Interpretation der beiden Aktionen C und D identisch ist. d. h. die Menge aller zu C und D herleitbaren Ablaufpläne identisch ist.

Notation 3.1.1. *Die Äquivalenzbeziehungen (durch $\doteq$ hergestellte Beziehungen zwischen Aktionen) werden Schedulingaxiome genannt.*

Die Syntax von Schedulingaxiomen wird wie folgt formal definiert:

Definition 3.1.3. *Seien C und D Aktionen. Die Syntax der Schedulingaxiomen wird durch*

$$C \doteq D$$

definiert.

Die Semantik der Schedulingaxiomen wird wie folgt definiert:

Definition 3.1.4. *Seien C und D Aktionen. Eine Interpretation $\mathcal{I}$ von $\mathcal{RSV}$ erfüllt das Schedulingaxiom*

$$C \doteq D \qquad gdw. \qquad C^{\mathcal{I}} = D^{\mathcal{I}}$$

Beispiele für Schedulingaxiome beinhaltet der nun folgende Abschnitt.

3.2 Kalkül für die Schedulingsprache $\mathcal{RSV}$

Zwei Aktionen sind *semantisch äquivalent*, wenn die Interpretation der beiden Aktionen identisch ist. Wie in Abschnitt 3.1.2 gezeigt, ist die Interpretation der beiden Aktionen

$$(\mathbf{pll}\,(\mathbf{seq}\,P_3(c,15),P_4(c,16)),(\mathbf{xor}\,P_5(c,3),P_6(d,5))) \tag{9}$$

und

$$\begin{aligned}(\mathbf{xor}\quad &(\mathbf{pll}\,(\mathbf{seq}\,P_3(c,15),P_4(c,16)),P_5(c,3)),\\ &(\mathbf{pll}\,(\mathbf{seq}\,P_3(c,15),P_4(c,16)),P_6(d,5)))\end{aligned} \tag{10}$$

identisch, d.h. die Menge aller zu der Aktion (9) gehörigen Ablaufpläne und die Menge aller zu der Aktion (10) gehörigen Ablaufpläne sind identisch. Somit bilden die beiden Aktionen ein Schedulingaxiom, dargestellt in dem folgenden Ausdruck:

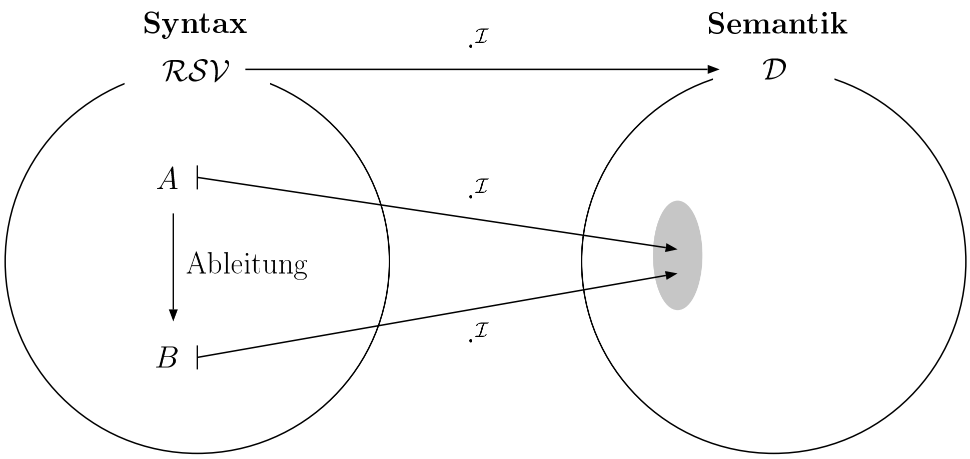


Abbildung 3.1: Grafische Darstellung eines Schedulingaxioms $A \doteq B$

$$\begin{aligned}
&(\mathbf{pll}\,(\mathbf{seq}\,P_3(c,15),P_4(c,16)),(\mathbf{xor}\,P_5(c,3),P_6(d,5)))\\
\doteq\quad &(\mathbf{xor}\quad(\mathbf{pll}\,(\mathbf{seq}\,P_3(c,15),P_4(c,16)),P_5(c,3)),\\
&\qquad\qquad(\mathbf{pll}\,(\mathbf{seq}\,P_3(c,15),P_4(c,16)),P_6(d,5)))
\end{aligned}$$

Im folgenden wird ein Kalkül vorgestellt, der eine Aktion A durch Umformungen auf Ausdrucksebene in eine normalisierte Aktion B transformiert, und bezogen auf das obige Beispielaxiom die darin enthaltene syntaktische Ableitung auf der Sprache $\mathcal{RSV}$ liefert. Im Anschluß wird für den vorgestellten Kalkül bewiesen, daß er ein korrekter Kalkül ist, d.h. nur solche Ableitungen erlaubt, die garantieren, daß alles syntaktisch ableitbare semantisch gleich bleibt. Die Korrektheit gewährleistet, daß eine Aktion A und ihre mittels des Kalküls abgeleitete normalisierte Form B stets ein Schedulingaxiom ($A \doteq B$) bilden. Graphisch zeigt diese Aussage Abbildung 3.1.

3.2.1 Der $\mathcal{RSV}$-Kalkül

Der Kalkül, den wir im folgenden $\mathcal{RSV}$-Kalkül nennen, wird wie folgt definiert:

Definition 3.2.1. *Seien $A_1, A_2, \cdots, A_k, B_1, \cdots, B_l, A_{k+2}, \cdots, A_n$ beliebige Aktionen. Der Kalkül besitzt folgende* 5 *Ableitungsregeln:*

1. *Assoziativität des Operators '**seq**' : Diese Regel beschreibt, daß ein mittels '**seq**' verbundener Subausdruck, der als ein Argument des Operators '**seq**' vorkommt, aufgelöst werden kann.*

$$\frac{(\mathbf{seq}\,A_1,A_2,\cdots,A_k,(\mathbf{seq}\,B_1,B_2,\cdots,B_l),A_{k+2},A_{k+3},\cdots,A_n)}{(\mathbf{seq}\,A_1,A_2,\cdots,A_k,B_1,B_2,\cdots,B_l,A_{k+2},A_{k+3},\cdots,A_n)}$$

2. *Assoziativität des Operators '***xor***' : Diese Regel beschreibt, daß ein mittels '***xor***' verbundener Subausdruck, der als ein Argument des Operators '***xor***' vorkommt, aufgelöst werden kann.*

$$\frac{(\mathbf{xor}\ A_1, A_2, \cdots, A_k, (\mathbf{xor}\ B_1, B_2, \cdots, B_l), A_{k+2}, A_{k+3}, \cdots, A_n)}{(\mathbf{xor}\ A_1, A_2, \cdots, A_k, B_1, B_2, \cdots, B_l, A_{k+2}, A_{k+3}, \cdots, A_n)}$$

3. *Assoziativität des Operators '***pll***' : Diese Regel beschreibt, daß ein mit '***pll***' verbundener Subausdruck, der als ein Argument des Operators '***pll***' vorkommt, aufgelöst werden kann.*

$$\frac{(\mathbf{pll}\ A_1, A_2, \cdots, A_k, (\mathbf{pll}\ B_1, B_2, \cdots, B_l), A_{k+2}, A_{k+3}, \cdots, A_n)}{(\mathbf{pll}\ A_1, A_2, \cdots, A_k, B_1, B_2, \cdots, B_l, A_{k+2}, A_{k+3}, \cdots, A_n)}$$

4. *Die folgende Regel beschreibt, daß der Operator '***xor***', sofern er als Argument einer mit '***seq***' verbundenen Aktion vorkommt, an die äußerste linke Position bewegt werden kann.*

$$\frac{(\mathbf{seq}\ A_1, A_2, \cdots, A_k, (\mathbf{xor}\ B_1, B_2, \cdots, B_l), A_{k+2}, A_{k+3}, \cdots, A_n)}{\begin{array}{r}(\mathbf{xor}\ (\mathbf{seq}\ A_1, A_2, \cdots, A_k, B_1, A_{k+2}, A_{k+3}, \cdots, A_n),\\ (\mathbf{seq}\ A_1, A_2, \cdots, A_k, B_2, A_{k+2}, A_{k+3}, \cdots, A_n),\\ \vdots\quad\quad\quad\quad\quad\\ (\mathbf{seq}\ A_1, A_2, \cdots, A_k, B_l, A_{k+2}, A_{k+3}, \cdots, A_n))\end{array}}$$

5. *Die folgende Regel beschreibt, daß der Operator '***xor***', sofern er als Argument einer mit '***pll***' verbundenen Aktion vorkommt, an die äußerste linke Position bewegt werden kann.*

$$\frac{(\mathbf{pll}\, A_1, A_2, \cdots, A_k, (\mathbf{xor}\, B_1, B_2, \cdots, B_l), A_{k+2}, A_{k+3}, \cdots, A_n)}{\begin{array}{r}\mathbf{xor}\, ((\mathbf{pll}\, A_1, A_2, \cdots, A_k, B_1, A_{k+2}, A_{k+3}, \cdots, A_n), \\ (\mathbf{pll}\, A_1, A_2, \cdots, A_k, B_2, A_{k+2}, A_{k+3}, \cdots, A_n), \\ \vdots \qquad\qquad \\ (\mathbf{pll}\, A_1, A_2, \cdots, A_k, B_l, A_{k+2}, A_{k+3}, \cdots, A_n))\end{array}}$$

Wir werden jetzt zeigen, daß jede der zuletzt definierten Regeln die Interpretation eines $\mathcal{RSV}$-Ausdruckes nicht verändert, d. h. der Kalkül bzgl. der Semantik korrekt ist.

Lemma 3.2.1. *Der Umformungskalkül der Sprache $\mathcal{RSV}$ ist ein korrekter Kalkül, d.h. er läßt nur solche Ausdrucksumformungen zu, die die semantische Äquivalenz erhalten.*

Beweis. Eine Regel der Form

$$\frac{A}{B}$$

erhält die semantische Äquivalenz, genau dann wenn die Interpretation des oberen Ausdrucks A und die Interpretation des unteren Ausdrucks B identisch sind ($A^{\mathcal{I}} = B^{\mathcal{I}}$).
Im folgenden wird für die obigen 5 Regeln gezeigt, daß sie semantikerhaltend sind, d. h. die Interpretation des oberen und unteren Ausdrucks übereinstimmt:

Regel 1: Die Menge aller aus dem oberen und unteren Ausdruck herleitbaren Ablaufpläne ist offensichtlich identisch, da die beiden Ausdrücke dieselbe sequentielle Anordnung der Aktionen $A_1, \cdots, A_k, B_1, \cdots, B_l, A_{k+2}, \cdots, A_{n-1}$ und A_{k+2} beschreiben. Ansonsten findet bei der Ausdrucksumformung keine Veränderung statt. Somit gilt die folgende Gleichung:

$$\begin{aligned}&(\mathbf{seq}\, A_1, A_2, \cdots, A_k, (\mathbf{seq}\, B_1, B_2, \cdots, B_l), A_{k+2}, A_{k+3}, \cdots, A_n)^{\mathcal{I}} \\ &\quad = \quad (\mathbf{seq}\, A_1, A_2, \cdots, A_k, B_1, B_2, \cdots, B_l, A_{k+2}, A_{k+3}, \cdots, A_n)^{\mathcal{I}}\end{aligned}$$

Regel 2 und *Regel 3* können ähnlich wie *Regel 1* bewiesen werden.

Regel 4: Bei dem $k+1$-ten Argument des Operators **seq** besteht eine Auswahlmöglichkeit. Eine der l Aktionen $B_1, \cdots B_{l-1}$ und B_l ist auszuwählen. Für jede

Auswahl von $B_i\,(i = 1, \cdots, l)$ wird eine eindeutig bestimmte Menge aller zugehörigen Ablaufpläne, diese sei M_{B_i}, hergeleitet. Die Menge aller zu dem oberen Ausdruck gehörigen Ablaufpläne entspricht dann offensichtlich der Vereinigung der Mengen $M_{B_1} \cdots M_{B_{l-1}}$ und M_{B_l}. Ferner entspricht diese Vereinigungsmenge der Menge aller zu dem unteren Ausdruck gehörigen Ablaufpläne. Somit gilt die folgende Gleichung:

$$
\begin{aligned}
&(\mathbf{seq}\, A_1, A_2, \cdots, A_k, (\mathbf{xor}\, B_1, B_2, \cdots, B_l), A_{k+2}, A_{k+3}, \cdots, A_n)^{\mathcal{I}} \\
&\quad = \quad (\mathbf{xor} \quad (\mathbf{seq}\, A_1, A_2, \cdots, A_k, B_1, A_{k+2}, A_{k+3}, \cdots, A_n), \\
&\qquad\qquad\qquad (\mathbf{seq}\, A_1, A_2, \cdots, A_k, B_2, A_{k+2}, A_{k+3}, \cdots, A_n), \\
&\qquad\qquad\qquad\qquad \vdots \\
&\qquad\qquad\qquad (\mathbf{seq}\, A_1, A_2, \cdots, A_k, B_l, A_{k+2}, A_{k+3}, \cdots, A_n))^{\mathcal{I}}
\end{aligned}
$$

Regel 5: Diese kann ähnlich wie *Regel 4* bewiesen werden.

Die Korrektheit des Kalküls erlaubt die Formulierung folgenden Satzes:

Satz 3.2.1. *Für jeden beliebigen Ausdruck (Aktion) A aus der Sprache $\mathcal{RSV}$ kann der Operator '**xor**' immer an die äußerste linke Position bewegt werden, sofern der Operator '**xor**' im Inneren der Aktion A vorkommt.*

Beweis. Zum Beweis der Hypothese reicht es zu zeigen, daß für jeden Ausdruck A alle Ableitungen maximaler Länge bei einem normalisierten Ausdruck terminieren. Eine Ableitung maximaler Länge sei dabei eine Ableitung, bei der ein Ausdruck erreicht wird, auf den sich keine der 5 Regeln des $\mathcal{RSV}$-Kalküls anwenden läßt.
Wir betrachten dazu zunächst den *am weitesten rechts* vorkommden '**xor**'-Operator bei A. Auf diesen '**xor**'-Operator wird eine der Regeln 2, 4 oder 5 angewendet. Dieser Vorgang wird wiederholt, bis alle '**xor**'-Operatoren an die äußerste linke Position bewegt worden sind. Alle im inneren der Aktion A vorkommenden '**xor**'-Operatoren können auf diese Weise offensichtlich Schritt für Schritt nach links bewegt werden, da bei den Regeln 4 und 5 der '**xor**'-Operator nach links bewegt wird und bei der Regel 2 der innere '**xor**'-Operator verschwindet.
Anschließend werden die übrigen Regeln 1 und 3 solange angewendet, bis keine der beiden Regeln mehr angewendet werden kann. □

Bemerkung: Satz 3.2.1 besagt, daß ein beliebiger $\mathcal{RSV}$-Ausdruck in einen semantisch äquivalenten, *normalisierten* Ausdruck transformiert werden kann. Ein wichtiger Sachverhalt bei dieser Transformation ist, daß die Anwendung der Regeln 2, 4 und 5 stets zuerst für *den an der weitesten rechten Stelle* vorkommenden '**xor**'-Operator geschieht. Die wiederholte Anwendung für *den am weitesten links* vorkommenden '**xor**'-Operator führt zwar auch zur Terminierung, aber der Berechnungsaufwand kann dabei sehr viel größer sein.

Die folgenden Beispiele sollen zum besseren Verständnis des Kalküls beitragen:

Beispiel 3.2.1. Der Ausdruck

$$(\mathbf{seq}\, P_1(a, t_1), (\mathbf{xor}\, P_2(b, t_2), P_3(c, t_3)), P_4(d, t_4))$$

kann durch Anwendung des Kalküls in den folgenden semantisch äquivalenten, normalisierten Ausdruck transformiert werden:

$$\begin{array}{ll} \mathbf{xor} & (\mathbf{seq}\, P_1(a, t_1), P_2(b, t_2), P_4(d, t_4)), \\ & (\mathbf{seq}\, P_1(a, t_1), P_3(c, t_3), P_4(d, t_4)) \end{array}$$

Die semantische Äquivalenz der beiden obigen Aktionen repräsentiert das folgende Schedulingaxiom

$$\begin{array}{l} (\mathbf{seq}\, P_1(a, t_1), (\mathbf{xor}\, P_2(b, t_2), P_3(c, t_3)), P_4(d, t_4)) \\ \quad \doteq \quad (\mathbf{xor}(\mathbf{seq}\, P_1(a, t_1), P_2(b, t_2), P_4(d, t_4)), \\ \qquad\qquad (\mathbf{seq}\, P_1(a, t_1), P_3(c, t_3), P_4(d, t_4))) \end{array}$$

Beispiel 3.2.2. Das folgende Schedulingaxiom soll verdeutlichen, wie der Operator '**xor**' der Subaktion $(\mathbf{xor}\ P_2(b, t_2), P_3(c, t_3))$ an die äußerste linke Position bewegt werden kann:

$$\begin{array}{l} (\mathbf{pll}\, P_1(a, t_1), (\mathbf{xor}\, P_2(b, t_2), P_3(c, t_3)), P_4(d, t_4)) \\ \quad \doteq \quad (\mathbf{xor}(\mathbf{pll}\, P_1(a, t_1), P_2(b, t_2), P_4(d, t_4)), \\ \qquad\qquad (\mathbf{pll}\, P_1(a, t_1), P_3(c, t_3), P_4(d, t_4))) \end{array}$$

Bemerkung: Wenn eine Aktion aus $\mathcal{RSV}$ in eine semantisch äquivalente normalisierte Aktion transformiert wird, kann dadurch die Anzahl der Grundaktionen bei der normalisierten Aktion exponentiell groß werden.

Bei einer normalisierten Aktion kommt der Operator '**xor**' nur einmal an der äußersten linken Position vor. Argumente dieses Operators '**xor**' sind solche Aktionen, bei denen nur die beiden Operatoren '**seq**' und '**pll**' vorkommen. Folgende begriffliche Vereinbarung wird getroffen:

Notation 3.2.1. *Eine Aktion, bei der nur die Operatoren '***seq***' oder '***pll***' vorkommen, wird eine* reduzierte Aktion *genannt.*

Wegen der Existenz von Satz 3.2.1 genügt es, ein algorithmisches Verfahren zu entwickeln, daß zur Lösung eines $\mathcal{RSV}$-Problems eine als Eingabe gegebene Aktion zuerst in die *normalisierte* Form transformiert und dann für jede einzelne reduzierte Aktion Ablaufpläne mit minimaler Gesamtbearbeitungszeit berechnet werden. Diejenigen Ablaufpläne unter all diesen berechneten Ablaufpläne mit minimaler Gesamtbearbeitungszeit, entsprechen dann den gesuchten optimalen Ablaufplänen für die Eingabe-Aktion. Im nächsten Kapitel 4 wird ein Lösungsalgorithmus vorgestellt, mit dem für eine beliebige reduzierte Aktion Ablaufpläne mit minimaler Gesamtbearbeitungszeit (optimale Ablaufpläne für eine reduzierte Aktion) berechnet werden können.

3.3 Die Schedulingsprache $\mathcal{RSVX}$

Wie zuletzt gezeigt, kann bei einer beliebigen Aktion A der Operator '**xor**' immer an die äußerste linke Position bewegt werden, sofern der Operator '**xor**' im inneren Teil der Aktion A vorkommt. Dies besagt, daß jede Aktion zu einer semantisch äquivalenten, normalisierten Aktion transformiert werden kann. Bei dieser normalisierten Aktion sind reduzierte Aktionen ('**xor**'-freie Subausdrücke) durch ein einzelnes '**xor**' verknüpft.

Wird ein Projekt unter Berücksichtigung aller alternativer Teilabläufe und Anordnungsbeziehungen als eine Aktion in der Schedulingsprache $\mathcal{RSV}$ dargestellt, so kann diese Aktion, durch Anwendung des in Abschnitt 3.2 vorgestellten Kalküls, in eine semantisch äquivalente, normalisierte Aktion umgeformt werden.

Die Teilmenge aller Aktionen aus $\mathcal{RSV}$, die nur aus normalisierten Aktionen bestehen, bilden eine eigene Sprache, die mit $\mathcal{RSVX}$ bezeichnet wird. Jede Aktion aus $\mathcal{RSV}$ kann in eine semantisch äquivalente Aktion aus $\mathcal{RSVX}$ umgewandelt werden. Die Teilmengenbeziehung zwischen $\mathcal{RSV}$ und $\mathcal{RSVX}$ sowie die Interpretationsabbildung für eine Aktion A auf eine Teilmenge von $\mathcal{D}$ zeigt die Abbildung 3.2. Die Menge aller aus $\mathcal{RSV}$-Ausdrücken ableitbaren Ablaufpläne und die Menge aller aus $\mathcal{RSVX}$-Ausdrücken ableitbaren Ablaufpläne entsprechen der gleichen Menge $\mathcal{D}$. Im folgenden wird die Sprache $\mathcal{RSVX}$ formal definiert:

3.3.1 Die Syntax der Schedulingsprache $\mathcal{RSVX}$

Definition 3.3.1. *Das Vokabular von $\mathcal{RSVX}$ besteht aus drei disjunkten Mengen von Symbolen. Diese Mengen sind:*

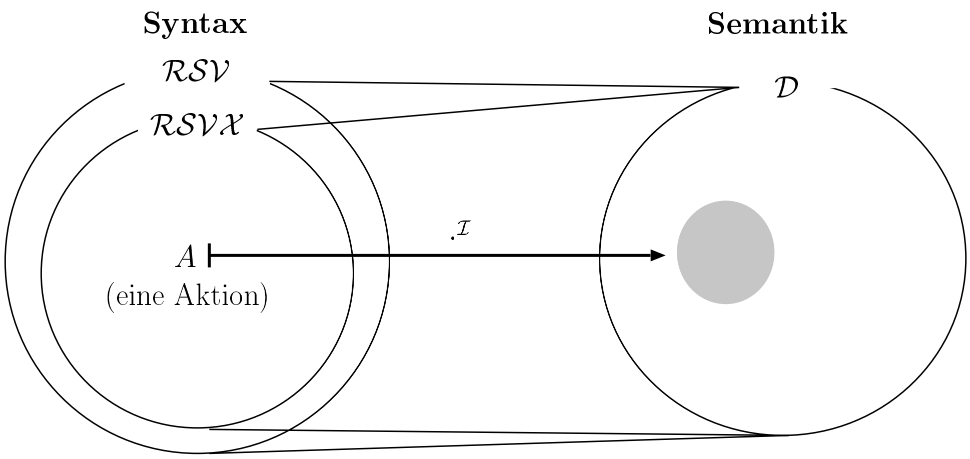


Abbildung 3.2: Die Subklassenbeziehung zwischen $\mathcal{RSV}$ und $\mathcal{RSVX}$ sowie die Interpretationsabbildung der beiden Sprachen

- *Eine Menge von Zeichen für Grundaktionen. Die Grundaktionen haben dieselbe Form wie bei Definition 3.1.1.*
- *Eine Menge von strukturellen Symbolen. Die Menge aller strukturellen Symbole enthält die Operatoren '**seq**', '**pll**' und '**xor**'. Das strukturelle Symbol '**xor**' wird ausgezeichneter Operator genannt.*
- *Die beiden Klammersymbole* (,)*.*

Bevor die Konstruktionsvorschriften für eine Aktion aus $\mathcal{RSVX}$ angegeben werden können, ist es erforderlich, die Bildungsregeln der reduzierten Aktionen aufzuzeigen.

Jede Grundaktion ist eine reduzierte Aktion. Sind A_1, A_2, ..., A_k reduzierte Aktionen, dann sind

$$(\mathbf{seq}\, A_1, A_2, \cdots, A_k),$$
$$(\mathbf{pll}\, A_1, A_2, \cdots, A_k)$$

reduzierte Aktionen.

Aktionen in $\mathcal{RSVX}$ werden mittels folgender Regeln gebildet:
Jede reduzierte Aktion ist eine Aktion. Seien B_1, B_2, $\cdots$, B_k reduzierte Aktionen, dann ist

$$(\mathbf{xor}\, B_1, B_2, \cdots, B_k)$$

eine Aktion.

Die Operatoren '**seq**' und '**pll**' haben in $\mathcal{RSVX}$ die folgende Semantik:

- '**seq**' : Dieser Operator dient zur Spezifikation der *sequentiellen* Abarbeitung einer bzw. mehrerer reduzierten Aktionen. So spezifiziert ($\mathbf{seq}\, A_1, A_2, \cdots, A_k$) aus Definition 3.3.1 die sequentielle Erfüllung der reduzierten Aktionen $A_1, A_2, \cdots, A_k$.

- '**pll**' : Dieser Operator dient zur Spezifikation der *parallelen* Abarbeitung einer bzw. mehrerer reduzierten Aktionen. So spezifiziert ($\mathbf{pll} A_1, A_2, \cdots, A_k$) aus Definition 3.3.1 die parallele Erfüllung der reduzierten Aktionen $A_1, A_2, \cdots, A_k$.

Der ausgezeichnete Operator '**xor**' besitzt folgende Semantik:

- '**xor**' : Dieser Operator dient zur Spezifikation der Wahlmöglichkeit einer reduzierten Aktion aus einer Menge alternativer reduzierten Aktionen. So bedeutet die Aktion ($\mathbf{xor}\ B_1, \cdots, B_k$) aus Definition 3.3.1 eine Auswahl einer der reduzierten Aktionen $B_1, B_2, \cdots, B_k$.

3.3.2 Die Semantik der Schedulingsprache $\mathcal{RSVX}$

Da die Sprache $\mathcal{RSVX}$ einer Subklasse der Sprache $\mathcal{RSV}$ entspricht, kann die Semantik der Sprache $\mathcal{RSVX}$ direkt aus $\mathcal{RSV}$ übernommen werden.

3.3.3 Graphische Darstellung der Schedulingsprache $\mathcal{RSVX}$

In diesem Abschnitt wird eine grafische Repräsentation für Ausdrücke aus $\mathcal{RSVX}$ vorgestellt. Die graphische Repräsentation soll vor allem eine einfache Erkennung der Struktur von reduzierten Aktionen aus $\mathcal{RSVX}$ ermöglichen.

Gemäß Tabelle 3.1 wird jedem Operator ein Symbol zugeordnet. Jeder Operator wird mittels des zugeordneten Symbols spezifiziert. Das Symbol des Operators '**pll**' verfügt über einen Punkt. An diesen Punkt werden die Argumente des Operators '**pll**' angeschlossen (siehe Beispiel 3.3.3). Die grafische Repräsentation soll an drei Beispielen verdeutlicht werden:

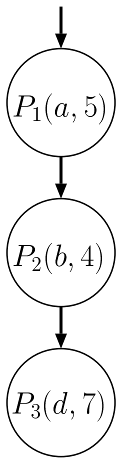
Beispiel 3.3.1. Der Ausdruck

$$(\mathbf{seq} P_1(a,5), P_2(b,4), P_3(d,7))$$

wird grafisch wie folgt repräsentiert:

Operator	Symbol
seq	$\longrightarrow$
pll	$\boxed{\|\|}$
xor	$\boxed{+}$

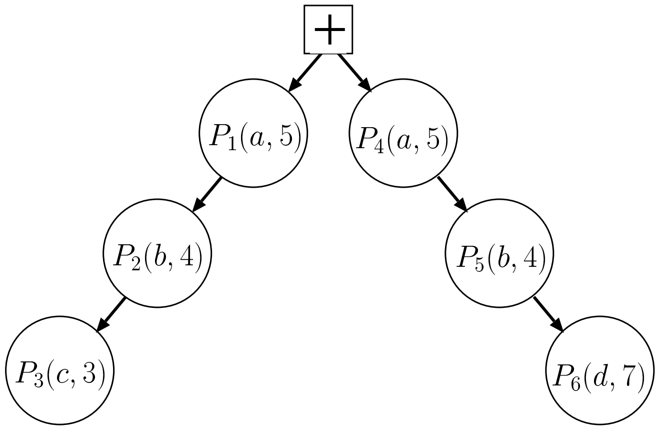
Tabelle 3.1: Die Operatoren '**seq**' '**pll**' sowie '**xor**' und die zugehörigen Symbole



Beispiel 3.3.2. Der Ausdruck

$$\begin{aligned}(\mathbf{xor} \quad & (\mathbf{seq}\, P_1(a, 5), P_2(b, 4), P_3(c, 3)),\\ & (\mathbf{seq}\, P_4(a, 4), P_5(b, 5), P_6(d, 7)))\end{aligned}$$

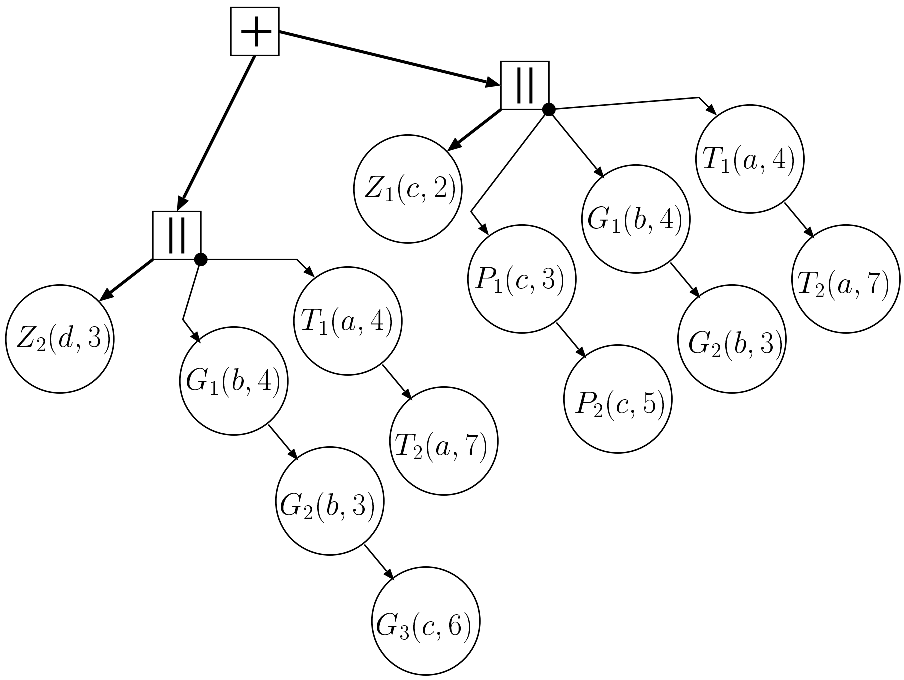
wird durch den folgenden Baum dargestellt.



Beispiel 3.3.3. Der Ausdruck

$$\begin{array}{llll}
(\mathbf{xor} & (\mathbf{seq} & (\mathbf{pll} & (\mathbf{seq}\, T_1(a,4), T_2(a,7)), (\mathbf{seq}\, G_1(b,4), G_2(b,3)), \\
 & & & (\mathbf{seq}\, P_1(c,3), P_2(c,5))), \\
 & & Z_1(c,2)), & \\
 & (\mathbf{seq} & (\mathbf{pll} & (\mathbf{seq}\, T_1(a,4), T_2(a,7)), (\mathbf{seq}\, G_1(b,4), G_2(b,3), G_3(c,6))), \\
 & & Z_2(d,3))) &
\end{array}$$

wird wie folgt grafisch dargestellt:



Kapitel 4

Lösung des $\mathcal{RSV}$-Problems mittels Diagramme

Ein $\mathcal{RSV}$-Ausdruck (Aktion) kann durch Anwendung des in Abschnitt 3.2 definierten Kalküls in einen semantisch äquivalenten $\mathcal{RSVX}$-Ausdruck transformiert werden. Bei einem $\mathcal{RSVX}$-Ausdruck sind eine oder mehrere reduzierte Aktionen, die alle alternativen Projektabläufe (Netzpläne) eines untersuchten Projektes darstellen, durch einen einzelnen Operator '**xor**' verknüpft. Eine reduzierte Aktion zeichnet sich dadurch aus, daß in ihr alleinig die beiden Operatoren '**seq**' und '**pll**' vorkommen, nicht hingegen der Operator '**xor**'.

In der vorliegenden Arbeit wird davon ausgegangen, daß die Verfügbarkeit der Ressourcen beschränkt ist (siehe Abschnitt 2.1). Diese Ressourcenbeschränkung kann Ressourcenkonflikte verursachen, die eine parallele Bearbeitung mehrerer Aktionen verhindern. Solche Ressourcenkonflikte können durch *Sequentialisierung* aller an einem Ressourcenkonflikt beteiligten Grundaktionen aufgelöst werden. Die Gesamtlaufzeit einer reduzierten Aktion kann je nachdem, in welcher Anordnung die an einem Ressourcenkonflikt beteiligten Grundaktionen sequentialisiert werden, unterschiedlich sein. Die folgende reduzierte Aktion verdeutlicht dies:

$$(\mathbf{pll}\,(\mathbf{seq}\,P_1(a,1), P_2(b,2)), P_3(a,2))$$

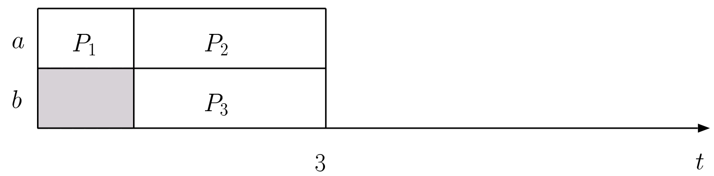
Bei der parallelen Durchführung der beiden Subaktionen $(\mathbf{seq}\,P_1(a,1), P_2(b,2))$ und $P_3(a,2)$ entsteht ein Ressourcenkonflikt zwischen den beiden Aktionen $P_1(a,1)$ und $P_3(a,2)$, da sie *gleichzeitig* dieselbe Ressource a beanspruchen. Wie bereits erwähnt kann der entstehende Ressourcenkonflikt durch Sequentialisierung aufgelöst werden. Sequentialisierung bedeutet, daß zuerst nur eine Aktion durchgeführt wird und die andere wartet, bis die Ressource wieder verfügbar ist. Im Beispiel existieren zwei alternative Sequentialisierungsmöglichkeiten:

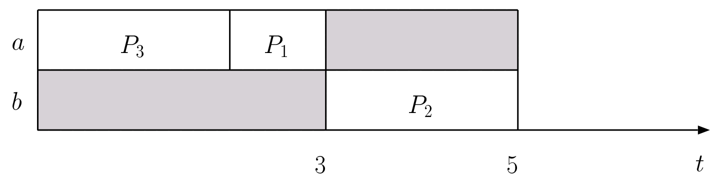
- zuerst wird die Grundaktion $P_1(a,1)$ durchgeführt, oder

- zuerst wird die Grundaktion $P_3(a, 2)$ durchgeführt.

Die zwei alternativen Auflösungsmöglichkeiten des Ressourcenkonflikts führen zu zwei alternativen Ablaufplänen bei obiger Aktion. Bei dem ersten Ablaufplan wird die Grundaktion $P_1(a, 1)$ zuerst durchgeführt, bei dem zweiten die Grundaktion $P_3(a, 2)$.

Die beiden Ablaufpläne erfüllen zwar dieselbe Aktivität, benötigen jedoch unterschiedliche Gesamtlaufzeiten. Dies zeigen die folgenden beiden zu den Ablaufplänen gehörigen Gantt-Charts:





Das Beispiel zeigt somit, daß zu einer reduzierten Aktion *mehrere (aktive) Ablaufpläne* existieren können, die unterschiedliche Bearbeitungszeiten benötigen.

Sind für alle zu einem $\mathcal{RSV}$-Ausdruck gehörigen reduzierten Aktionen die zugehörigen Ablaufpläne ermittelt worden, entsprechen diejenigen Ablaufpläne unter diesen den optimalen Lösungen, die die kleinste Gesamtbearbeitungszeit aufweisen. In diesem Kapitel wird ein Berechnungsmodell vorgestellt, mit dem für eine reduzierte Aktion alle nichtredundanten zugehörigen Ablaufpläne ermittelt werden können. "nichtredundant" heißt dabei, daß jeder Ablaufplan nur einmal berechnet wird. Dieses Lösungsmodell werden wir als $\mathcal{RSV}$-Diagrammberechnung bezeichnen, da die Rechenvorgänge mittels Diagrammen dargestellt werden können[1]. Auf diesem Wege erhalten wir zugleich ein Modell, mit dem das $\mathcal{RSV}$-Problem gelöst werden kann. Die Aufbaustruktur dieses Modells zeigt die Abbildung 4.1.

Folgende im Rahmen der $\mathcal{RSV}$-Diagrammberechnung vorausgesetzte Nebenbedingung ist dabei von wesentlicher Bedeutung:

- Eine durchzuführende Grundaktion kann nicht unterbrochen werden.

[1] Diese graphische Darstellungsweise ermöglicht, daß der Lösungsvorgang sehr leicht verstanden werden kann.

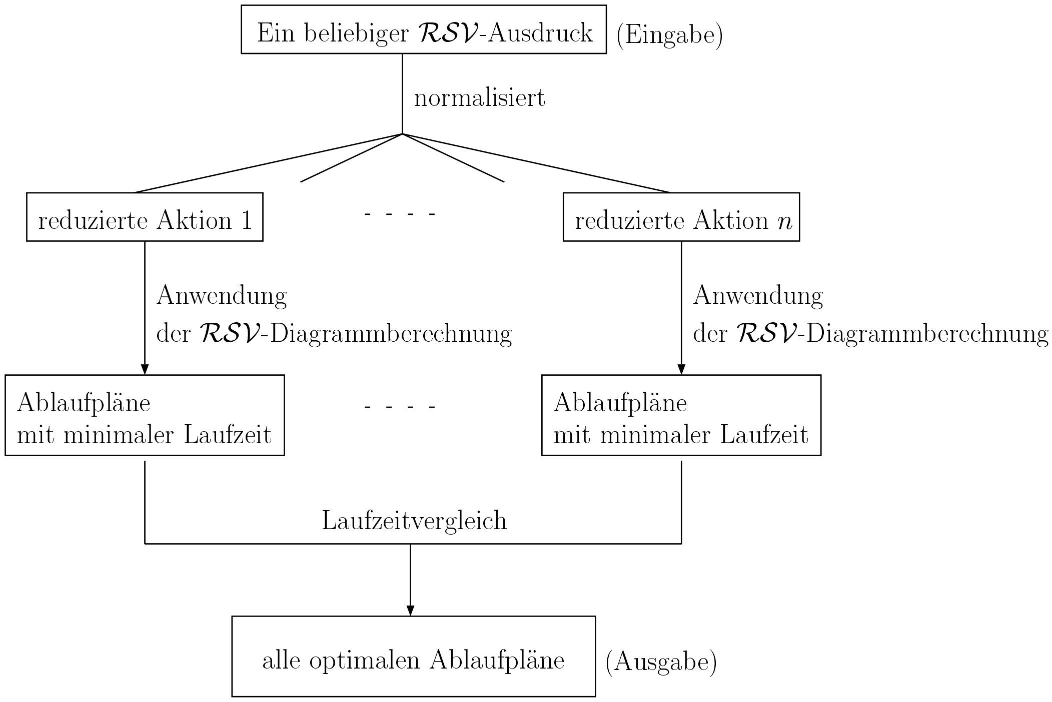


Abbildung 4.1: Übersicht zur Lösung des $\mathcal{RSV}$-Problems

In Abschnitt 4.1 wird zunächst erläutert, wie die nichtredundante Aufzählung einer einzigen Abarbeitungsfolge (Reihenfolge) für jeden Ablaufplan ermöglicht werden kann. Dies wird im Rahmen des Korrektheitsbeweises des $\mathcal{RSV}$-Berechnungsmodells benötigt, wo wir unter anderem beweisen werden, daß für eine reduzierte Aktion die $\mathcal{RSV}$-Berechnung alle nichtredundanten Ablaufpläne aufzählt. In Abschnitten 4.2 und 4.3 wird das $\mathcal{RSV}$-Berechnungsmodell vorgestellt. Im Anschluß werden wir in Abschnitt 4.4 zeigen, wie das $\mathcal{RSV}$-Berechnungsmodell konkret in einer Programmiersprache implementiert werden kann. In Abschnitt 4.5 werden wir den bereits erwähnten Korrektheitsbeweis führen.
Ablaufpläne für das $\mathcal{RSV}$-Problem haben wir bis jetzt stets mittels Gantt-Diagrammen dargestellt. In Abschnitt 4.6 werden wir zeigen, daß Ablaufpläne auch als Aktionen aus $\mathcal{RSVX}$ dargestellt werden können. Abschließend erfolgt in Abschnitt 4.7 eine Gegenüberstellung des $\mathcal{RSV}$-Berechnungsmodells und des Modells von Manne .

	Abarbeitungsfolgen	Laufzeit
1	$[P_1(a,1), P_{2_{1<}}(b,2), P_3(a,2)]$	3
2	$[P_1(a,1), P_3(a,2), P_{2_{1<}}(b,2)]$	3
3	$[P_3(a,2), P_1(a,1), P_{2_{1<}}(b,2)]$	5

Tabelle 4.1: Abarbeitungsfolgen und Laufzeiten zur Beispielaktion $(\mathbf{pll}\,(\mathbf{seq}\; P_1(a,1), P_2(b,2)), P_3(a,2))$

a	1	3	3					
b		2	2					

Abbildung 4.2: Ablaufplan der Abarbeitungsfolge 1 bzw. 2

4.1 Enumeration aller nichtredundanten Abarbeitungsfolgen

Aus einer Abarbeitungsfolge der Vorgänge eines Projektes[2] kann ein Ablaufplan des Projektes hergestellt werden, indem die Vorgänge in der Abarbeitungsfolge unter Berücksichtigung der gegebenen Präzedenzbeziehungen in ein Gantt-Diagramm eingeplant werden. Bei der in der Einführung dieses Kapitel betrachteten Aktion

$$(\mathbf{pll}\,(\mathbf{seq}\; P_1(a,1), P_2(b,2)), P_3(a,2))$$

existieren beispielsweise insgesamt 3 alternative Abarbeitungsfolgen, die in Tabelle 4.1 wiedergegeben werden. Bei diesen Abarbeitungsfolgen ist bei jeder Grundaktion der direkte Vorgänger gekennzeichnet, falls ein direkter Vorgänger existiert. Die Kennzeichnung $1<$ bei der Grundaktion $P_{2_{1<}}(b,2)$ bei allen drei Abarbeitungsfolgen besagt, daß die Grundaktion P_2 den direkten Vorgänger P_1 besitzt. Somit muß P_2 bei allen drei Alternativen hinter P_1 eingeplant werden. Wenn die Vorgänge in der Abarbeitungsfolge 1 (bzw. 2) aus Tabelle4.1 eingeplant werden, erhalten wir einen zu der betrachteten Aktion gehörigen Ablaufplan; dieser entspricht dem Ablaufplan aus Abbildung 4.2. Den Ablaufplan , der aus der Abarbeitungsfolge 3 aus Tabelle4.1 hergeleitet werden kann, zeigt die Abbildung

[2] Eine Abarbeitungsfolge wird in Fachliteraturen auch als eine Reihenfolge bezeichnet.

a	3	3	1					
b				2	2			

Abbildung 4.3: Ablaufplan der Abarbeitungsfolge 3

	Reihenfolgen	Laufzeit
1	$[(0,1),(1,2),(1,3)]$	3
2	$[(0,1),(1,3),(1,2)]$	3
3	$[(0,3),(2,1),(3,2)]$	5

Tabelle 4.2: Reihenfolgen und Laufzeiten

4.3.

Wie diese Beispielaktion zeigt, kann ein zu einer reduzierten Aktion gehöriger Ablaufplan mittels einer Abarbeitungsfolge (Reihenfolge) bzw. mehrerer Abarbeitungsfolgen erhalten werden. Die beiden Abarbeitungsfolgen 1 und 2 stellen denselben Ablaufplan aus Abbildung 4.2 dar, was dem Vorkommen von Redundanz entspricht.

Schrage [Sch70] stellt Ansätze vor, mit deren Hilfe für eine reduzierte Aktion *nur alle nichtredundanten* Abarbeitungsfolgen aufgezählt werden können. Jeder Ablaufplan wird dabei mittels *genau einer* Abarbeitungsfolge assoziiert. So wird *nur genau eine* Abarbeitungsfolge unter allen Abarbeitungsfolgen, die denselben Ablaufplan darstellen, aufgezählt.

Eine verschmelzende Betrachtung von Abarbeitungsfolgen und zugehöriger Ablaufpläne zeigt die Ursache der bei obigem Beispiel beobachteten Redundanz. Wir repräsentieren dazu die Abarbeitungsfolgen als Folgen von Tupeln der Form $(s(i), i)$, wobei i einer Grundaktion und $s(i)$ der Startzeit der Grundaktion i entspricht und diese Informationen aus der Ablaufplanung entnommen werden[3]. Solche Folgen bezeichnen wir als Reihenfolgen. Bei dem von uns betrachteten Beispiel erhalten wir auf diese Weise die drei Reihenfolgen aus Tabelle 4.2.

Ordnen wir die Tupel der ersten und zweiten Reihenfolge lexikographisch aufsteigend, so werden beide Reihenfolgen identisch und die Tatsache, daß hinter der ersten und zweiten Reihenfolge derselbe Ablaufplan steht, wird offensichtlich.
Der folgende Satz besagt, daß für eine reduzierte Aktion jeder zugehörige Ablaufplan mittels *einer eindeutigen* Reihenfolge repräsentiert werden kann.

Satz 4.1.1. *Gegeben sei ein Ablaufplan $\mathcal{P}$, wobei $1, \cdots, n$ den Prädikaten der in $\mathcal{P}$ vorkommenden Grundaktionen und $s(1), \cdots, s(n)$ den Startzeiten der Grundaktionen $1, \cdots, n$ entspreche. $[(s(J(1)), J(1)), \cdots, (s(J(n)), J(n))]$ sei die lexikographisch geordnete Reihenfolge der Tupel $(s(i), i)$ von $\mathcal{P}$, wobei $\{J(1), \cdots J(n)\}$ eine Permutation von $\{1, \cdots, n\}$ ist. Wenn alle Grundaktionen von $\mathcal{P}$ in der Reihenfolge $[(s(J(1)), J(1)), \cdots, (s(J(n)), J(n))]$ in ein Gantt-Chart eingeplant werden, wird der Ablaufplan $\mathcal{P}$ erhalten.*

[3] Hierbei werden die Grundaktionen, die wir bisher mittels $P_1, P_2, \cdots, P_n$ bezeichneten, aus Vereinfachungsgründen nur mittels ihres Index $1, 2, \cdots, n$ identifiziert.

$((0,1),a,1)$ — $((1,2_{1<}),b,2)$ — $((1,3),a,2)$ Reihenfolge 1

$((0,3),a,2)$ — $((2,1),a,1)$ — $((3,2_{1<}),b,2)$ Reihenfolge 2

Abbildung 4.4: nichtredundante Reihenfolgen

Beweis. Der Beweis befindet sich in [Sch70].

Definition 4.1.1. *Eine Reihenfolge F heißt* redundant*, wenn bei F zwei Grundaktionen i und j existieren, so daß das Tupel $(s(i),i)$ vor $(s(j),j)$ vorkommt und $s(i) > s(j)$ oder $s(i) = s(j)$* und *$i > j$ gilt.*

Satz 4.1.2. *[Sch70]Redundante Reihenfolgen brauchen nicht aufgezählt zu werden.*

Beweis. Wegen Satz 4.1.1 existiert offensichtlich eine andere lexikographisch geordnete Reihenfolge, die denselben Ablaufplan erzeugt. Jeder Ablaufplan braucht nur einmal erzeugt zu werden. □

Somit kann eine eins-zu-eins Korrespondenz zwischen einem Ablaufplan und einer nichtredundanten (lexikographisch geordneten) Reihenfolge gebildet werden. Wenn wir alle Ablaufpläne generieren wollen, genügt es, alle nichtredundanten Reihenfolgen zu erzeugen.

Die Erzeugung aller nichtredundanten Reihenfolgen kann durch einen Verzweigungsbaum illustriert werden. Der Verzweigungsbaum aus Abbildung 4.4 illustriert die Erzeugung aller nichtredundanten Reihenfolgen für die Aktion aus dem obigen Beispiel. Ein Knoten entspricht dabei einem Tripel $((s(i),i),r(i),t(i))$, wobei $s(i)$ die Startzeit der Grundaktion i, $r(i)$ die von i beanspruchte Ressource und $t(i)$ die von i benötigten zeitlichen Dauer wiedergibt. An i werden die direkten Vorgänger angefügt, wenn solche existieren. Die Schreibweise eines Tupels $(s(i),i)$ wurde zu $((s(i),i),r(i),t(i))$ erweitert, da in letzterer Darstellungsweise eine Reihenfolge vollständig einen Ablaufplan beschreibt und ein zugehöriger Ablaufplan ohne weiteres erzeugt werden kann. Der Erzeugungsvorgang soll kurz erläutert werden:

Zuerst kann entweder die Grundaktion 1 oder die Grundaktion 3 durchgeführt werden. Entsprechend wird am Anfang zu den zwei Knoten $((0,1),a,1)$ und $((0,3),a,2)$ verzweigt, wobei die beiden Grundaktionen 1 und 3 die Startzeit 0 besitzen. Ausgehend vom Knoten $((0,1),a,1)$ können im nächsten Schritt die Grundaktionen 2 und 3 parallel durchgeführt werden. Die Startzeiten der beiden Grundaktionen 2 und 3 sind dabei identisch und entsprechen dem Zeitpunkt 1.

$((0,1),a,4)$ — $((4,2),a,2)$ — $((6,3_{2<}),b,2)$ — $((8,4_{3<}),a,2)$ — $((10,5_{4<}),b,4)$

Reihenfolge 1 (Gesamtbearbeitungszeit: 14)

$((0,2),a,2)$ — $((2,1),a,4)$ — $((2,3_{2<}),b,2)$ — $((6,4_{3<}),a,2)$ — $((8,5_{4<}),b,4)$

Reihenfolge 2 (Gesamtbearbeitungszeit: 12)

$((2,3_{2<}),b,2)$ — $((4,4_{3<}),a,2)$ — $((6,1),a,4)$ — $((6,5_{4<}),b,4)$

$\Uparrow$

Reihenfolge 3 (Gesamtbearbeitungszeit: 10)

Abbildung 4.5: nichtredundante Reihenfolgen

Da beide Grundaktionen an letzter Stelle stehen, werden sie lexikographisch verglichen. Lexikographisch gilt $(1,2) < (1,3)$. So wird zuerst $(1,2)$ und dann $(1,3)$ gewählt. Vom Knoten $((0,3),a,2)$ aus kann die entsprechende Reihenfolge nach derselben Vorgehensweise erzeugt werden.

Es wird nun ein komplizierterer Fall anhand eines weiteren Beispiels betrachtet:

Beispiel 4.1.1. Bei der Aktion

$$(\mathbf{pll}\, P_1(a,4),\ (\mathbf{seq}\, P_2(a,2), P_3(b,2), P_4(a,2), P_5(b,4)))$$

werden insgesamt drei nichtredundante Reihenfolgen erzeugt, die aus dem Verzweigungsbaum von Abbildung 4.5 abgelesen werden können.

Wir betrachten die Reihenfolge 3, bei der zunächst die beiden Grundaktionen P_2 und P_3, repräsentiert durch die sequentielle Anordnung der beiden Tripel $((0,2),a,2)$, $((2,3_{2<}),b,2)$, hintereinander abgearbeitet werden. Als nächstes kann eins von den beiden Tripeln $((4,4_{3<}),a,2)$ und $((2,1),a,4)$ angeordnet werden (an der mit $\Uparrow$ markierten Stelle). Wird das Tripels $((2,1),a,4)$ an der mit $\Uparrow$ markierten Stelle angeordnet, erhalten wir zum Schluß die Reihenfolge:

$$[((0,2),a,2),((2,3_{2<}),b,2),((2,1),a,4),((6,4_{3<}),a,2),((8,5_{4<}),b,4)]$$

Diese Reihenfolge entspricht der Reihenfolge 2, was einem Vorkommen von Redundanz entspricht.

4.2 Das $\mathcal{RSV}$-Diagramm

Im letzten Abschnitt wurde gezeigt, wie für eine reduzierte Aktion alle nichtredundanten Reihenfolgen erzeugt werden können. In diesem Abschnitt wird eine neue Darstellungsmethodik für reduzierte Aktionen auf Basis von Diagrammen

Abbildung 4.6: Spezifikation des Operators '**pll**'

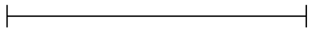


Abbildung 4.7: Spezifikation des Operators '**seq**' und der zeitlichen Dauer einer Grundaktion

vorgestellt. Die erhaltenen Diagramme werden als $\mathcal{RSV}$-*Diagramme* bezeichnet. Unter Zuhilfenahme der vorgestellten Darstellungsmethodik kann der Enumerationsvorgang aller nichtredundanten Reihenfolgen deutlich veranschaulicht werden. Auf Basis dieser Darstellungsmethodik wird später ein algorithmisches Verfahren vorgestellt, mit dem alle nichtredundanten Ablaufpläne berechnet werden können. Im Zentrum der vorgestellten Darstellungsmethodik steht eine *Scan-line*, wobei die Idee der Verwendung einer Scan-line vom Scan-line-Prinzip bei der Lösung geometrischer Algorithmen übernommen wurde. Das Scan-line-Prinzip gilt als eine der leistungsfähigsten Techniken zur Lösung geometrischer Probleme [OW90], [Woo84].

Jede reduzierte Aktion aus $\mathcal{RSV}$ kann mittels eines $\mathcal{RSV}$-Diagramms dargestellt werden. Ein $\mathcal{RSV}$-Diagramm weist eine Zeitachse auf, von der die Anfangszeit sowie Endzeit jeder Grundaktion abgelesen werden kann, die zur Durchführung der betrachteten reduzierten Aktion benötigt wird. Auf einem Zeitpunkt dieser Zeitachse befindet sich die Scan-line. Die Scan-line wird als Hilfsmittel bei der Auflösung von Ressourcenkonflikten verwendet. Ferner können mit ihrer Hilfe alle Auflösungsmöglichkeiten eines Ressourcenkonflikts ermittelt werden. Durch eine schrittweise Rechtsbewegung können so auf einfache Weise alle Ressourcenkonflikte bestimmt sowie aufgelöst werden.

Die Linie aus Abbildung 4.6 spezifiziert den Operator '**pll**'. Der Operator '**seq**' wird mittels der Linie aus Abbildung 4.7 spezifiziert. Hierbei gibt die Länge der Linie die beanspruchte zeitliche Dauer einer Grundaktion wieder.

Im folgenden werden verschiedene reduzierte Aktionen mittels eines $\mathcal{RSV}$-Diagramms dargestellt.

Beispiel 4.2.1. Die Aktion

$$\begin{aligned}&\mathbf{pll}\ (\mathbf{seq}\ P_1(a,1),\ (\mathbf{pll}\ P_2(b,1), P_3(d,2)),\ P_4(c,3)),\\ &\quad(\mathbf{seq}\ P_5(d,2), P_6(a,1))\end{aligned}$$

stellt das $\mathcal{RSV}$-Diagramm aus Abbildung 4.8 dar.

Beispiel 4.2.2. Die Aktion

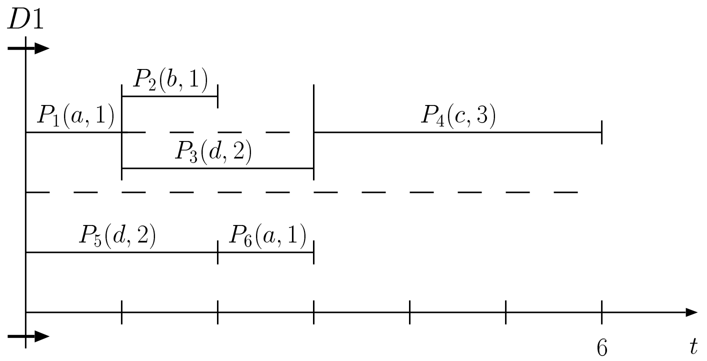


Abbildung 4.8: Ein $\mathcal{RSV}$-Diagramm mit dem Haltepunkt der Scan-line $t(H) = 0$

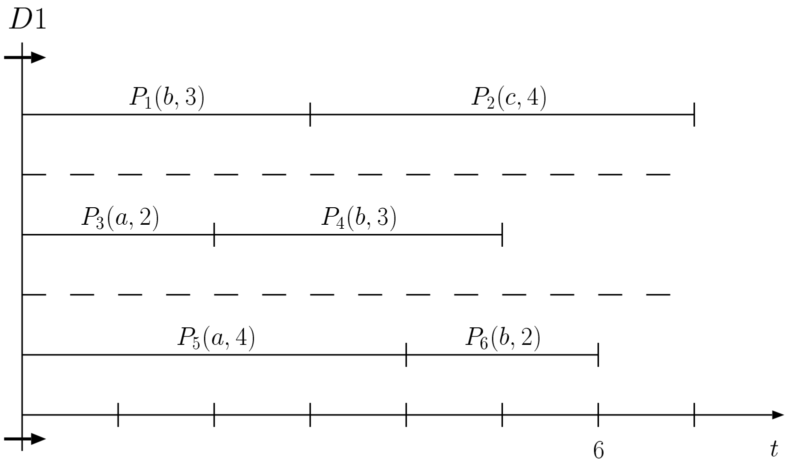


Abbildung 4.9: Ein $\mathcal{RSV}$-Diagramm mit dem Haltepunkt der Scan-line $t(H) = 0$

$$\begin{aligned}\textbf{pll } & (\textbf{seq } P_1(b,3),\ P_2(c,4)),\\ & (\textbf{seq } P_3(a,2),\ P_4(b,3)),\\ & (\textbf{seq } P_5(a,4),\ P_6(b,2))\end{aligned}$$

stellt das $\mathcal{RSV}$-Diagramm aus Abbildung 4.9 dar.

Jede Grundaktion besitzt bei einem $\mathcal{RSV}$-Diagramm *einen linken* sowie *einen rechten Endpunkt*. Wie bereits erwähnt, entspricht der linke Endpunkt einer Grundaktion der Anfangszeit der Durchführung; der rechte Endpunkt einer Grundaktion der Endzeit der Durchführung.

Notation 4.2.1. *Linke sowie rechte Endpunkte von Grundaktionen werden im folgenden als* Haltepunkte *bezeichnet. Ferner werden der linke sowie der rechte Endpunkt einer beliebigen Grundaktion $P(r,t)$ mit $LE(P(r,t))$ und $RE(P(r,t))$ bezeichnet.*

Zu Beginn steht die Scan-line auf dem Zeitpunkt $t = 0$. Statt eines kontinuierlichen Schwenks wird mit der Scan-line von Haltepunkt zu Haltepunkt "gesprungen". Wie bereits erwähnt, wird nach jedem Springen geprüft, ob Ressourcenkonflikte zwischen Grundaktionen entstehen, d.h. ob ein Zeitinterval hinter der Scan-line existiert, bei dem mehrere Grundaktionen dieselbe Ressource beanspruchen. Ist dies der Fall, werden die Grundaktionen sequentialisiert. Im folgenden Abschnitt wird ein Lösungsalgorithmus vorgestellt, bei dem der Sequentialisierungsvorgang ausführlich erläutert wird.

4.3 Der Algorithmus

Im Kontext von $\mathcal{RSV}$-Diagrammen führen wir zunächst folgende Notation ein:

Notation 4.3.1. *Mit (D, t_{SL}), wobei $t_{SL} \geq 0$, wird bezeichnet, daß sich die Scan-Line bei dem $\mathcal{RSV}$-Diagramm D auf dem Zeitpunkt t_{SL} befindet.*

Wie bereits erwähnt, springt die Scan-Line in einer Rechtsbewegung von einem Haltepunkt zum nächsten, wobei die Haltepunkte durch die Start- und Endzeiten der im Diagramm vorhandenen Grundaktionen gegeben werden. Bei jedem Haltepunkt wird für Grundaktionen, die an keinem Ressourcenkonflikt beteiligt sind (siehe Definition 4.3.5), der *Durchführungszeitraum* festgelegt. Bei einem Diagramm (D, t_{SL}) mit $t_{SL} > 0$ wird somit für eine ressourcenkonfliktfreie Grundaktion G_i, für die $LE(G_i) := t_l < t_{SL}$ und $RE(G_i) = t_{SL}$ gilt, der Zeitraum zwischen t_l und t_{SL} als ihr Durchführungszeitraum festgelegt. Dieser Zeitraum zwischen t_l und t_{SL} bei (D, t_{SL}) entspricht dem Zeitinterval, auf dem sich G_i bei (D, t_{SL}) befindet. Festlegung bzw. Festplatzierung einer Grundaktion bedeutet, daß sie nicht mehr bewegt werden darf. Grundaktionen, deren Durchführungszeitraum festgelegt wurde, werden als *eingefroren* bezeichnet.
Ressourcenkonflikte werden aufgelöst, indem bei jedem entstehenden Ressourcenkonflikt eine Grundaktion ausgewählt wird. Als Durchführungszeitraum einer derart ausgewählten Grundaktion wird der Zeitraum festgelegt, auf dem sie sich befindet. Das heißt, sie wird auf dem Platz im Diagramm, wo sie sich befindet, eingefroren. Alle sonstigen an einem Ressourcenkonflikt beteiligten Grundaktionen werden jeweils hinter die eingefrorene Grundaktion geschoben. Als Folge davon kommen auf der linken Seite der Scan-line keine Ressourcenkonflikte mehr vor. Ferner wird die durch die eingefrorene Grundaktion beanspruchte Ressource im Kontext weiterer Ressourcenkonflikte blockiert, als Folge der Nebenbedingung, daß eine eingefrorene Grundaktion nicht mehr bewegt werden darf. Alle weiteren Aktionen, die diese Ressource benötigen, müssen mindestens solange warten, bis die Ressource wieder von der eingefrorenen Grundaktion freigegeben wird (siehe Definition 4.3.4).

Die Gesamtbearbeitungszeiten der ermittelten Ablaufpläne können je nachdem, wie die an den Ressourcenkonflikten beteiligten Grundaktionen angeordnet werden, unterschiedlich sein. Um verschiedene Anordnungen der an einem Ressourcenkonflikt beteiligten Grundaktionen zu verfolgen und dazu gehörige Ablaufpläne zu ermitteln, wird das Diagramm vervielfacht. Das Diagramm wird derart vervielfacht, daß jede kombinatorisch mögliche Anordnung verfolgt wird (siehe Definition 4.3.6).
Wenn bei einem $\mathcal{RSV}$-Diagramm die Scan-line alle Grundaktionen überschritten hat, sind alle Grundaktionen eingefroren und $\mathcal{RSV}$-Diagramm stellt einen Ablaufplan dar. Im folgenden wird näher erläutert, wie unter Berücksichtigung der vorgegebenen Präzedenzbeziehungen der Grundaktionen ihre Durchführungszeiträume mit Hilfe der Scan-line ressourcenkonfliktfrei festgelegt werden können.

Dazu werden zunächst die folgenden Begriffe eingeführt.

Definition 4.3.1. *Bei einem $\mathcal{RSV}$-Diagramm (D, t_{SL}) mit $t_{SL} > 0$ heißt eine Grundaktion $P(r,t)$ t_{SL}-Zeit-Scan-line-Grundaktion gdw.*

- $LE(P(r,t)) < t_{SL}$ *und*
- $RE(P(r,t)) \geq t_{SL}$ *gilt.*

Definition 4.3.2. *Bei einem $\mathcal{RSV}$-Diagramm (D, t_{SL}) mit $t_{SL} > 0$ heißt eine Scan-line-Grundaktion $P(r,t)$ eine (t_{SL}-Zeit)-direkte Scan-line-Grundaktion gdw. bei (D, t_{SL}) für $P(r,t)$ die Eigenschaft $RE(P(r,t)) = t_{SL}$ gilt. Die von einer direkten Scan-line-Grundaktion beanspruchte Ressource r heißt eine (t_{SL}-Zeit)-direkte Scan-line-Ressource.*

Definition 4.3.3. *Bei einem $\mathcal{RSV}$-Diagramm (D, t_{SL}) mit $t_{SL} > 0$ sind die r beanspruchenden Grundaktionen $P_1(r,t_1), P_2(r,t_2), \cdots$ und $P_n(r,t_n)$ $(n \geq 2)$ an einem (t_{SL}, r)-Ressourcenkonflikt bzw. t_{SL}-Zeit Ressourcenkonflikt beteiligt gdw.*

- *r eine direkte Scan-line-Ressource ist,*
- *$P_1(r,t_1), P_2(r,t_2), \cdots$ und $P_n(r,t_n)$ $(n \geq 2)$ Scan-line-Grundaktionen sind und*
- *keine von diesen $P_1(r,t_1), P_2(r,t_2), \cdots$ und $P_n(r,t_n)$ $(n \geq 2)$ eingefroren ist.*

Die Ressource r wird als t_{SL}-Zeit-Konfliktressource bezeichnet. Ferner heißen die Grundaktionen $P_1(r,t_1), P_2(r,t_2), \cdots$ und $P_n(r,t_n)$ $(n \geq 2)$ (t_{SL}, r)-Konfliktgrundaktionen bzw. t_{SL}-Zeit Konfliktgrundaktionen.

Ist eine Scan-line-Grundaktion eingefroren, so ist ihr Durchführungszeitraum festgelegt. Wie bereits erwähnt, bleibt eine von einer eingefrorenen Grundaktion belegte Ressource solange blockiert, bis die Durchführung der zugehörigen Grundaktion abgeschlossen ist. Alle weiteren Grundaktion, die die blockierte Ressource beanspruchen, müssen soweit nach rechts geschoben werden, bis die blockierte Ressource wieder freigegeben wird[4].
Die folgenden weiteren Begriffe werden eingeführt:

Definition 4.3.4. *Bei einem $\mathcal{RSV}$-Diagramm (D, t_{SL}) mit $t_{SL} > 0$ heißt eine Ressource r t_{SL}-Zeit-blockierte Ressource gdw.*

- *eine r beanspruchende, eingefrorene (Scan-line-)Grundaktion $P(r,t)$ existiert, für die $LE(P) < t_{SL}$ und $RE(P) \geq t_{SL}$ gilt.*

Hierbei wird eine nicht eingefrorene Scan-line-Grundaktion, die eine t_{SL}-Zeit-blockierte Ressource beansprucht, als eine vs-Scan-line-Grundaktion bezeichnet.

Definition 4.3.5. *Bei einem $\mathcal{RSV}$-Diagramm (D, t_{SL}) mit $t_{SL} > 0$ ist eine Grundaktion $P(r,t)$* eine *t_{SL}-Zeit konfliktfreie Grundaktion gdw.*

- *$RE(P(r,t)) = t_{SL}$ gilt und*
- *$P(r,t)$ an keinem t_{SL}-Zeit Ressourcenkonflikt beteiligt ist.*

Zu jedem beliebigen Zeitpunkt t_{SL} können mehrere t_{SL}-Zeit Konfliktressourcen existieren. Dies ist gleichbedeutend damit, daß (t_{SL}, r)-Konfliktgrundaktionen bzgl. verschiedener Ressourcen existieren können. Für jede t_{SL}-Zeit Konfliktressource r wird jeweils eine (t_{SL}, r)-Konfliktgrundaktion ausgewählt und eingefroren. Hierbei ergeben sich verschiedene kombinatorische Möglichkeiten, je nachdem, welche t_{SL}-Zeit Konfliktgrundaktionen ausgewählt werden. Eine solche Kombination wird als *eine Konfliktkombination* bezeichnet. Formal wird dies wie folgt definiert:

Definition 4.3.6. $r_1, r_2, \cdots, r_n$ *seien bei einem $\mathcal{RSV}$-Diagramm (D, t_{SL}) mit $t_{SL} > 0$ t_{SL}-Zeit Konfliktressourcen. $P_{r_1,1}, \cdots, P_{r_1,m_1}$ seien alle* (t_{SL}, r_1)-Konfliktgrundaktionen*, $P_{r_2,1}, \cdots, P_{r_2,m_2}$ seien alle* (t_{SL}, r_2)-Konfliktgrundaktionen*, $\cdots$, und $P_{r_n,1}, \cdots, P_{r_n,m_n}$ seien alle* (t_{SL}, r_n)-Konfliktgrundaktionen. *Ein Element aus der folgenden Menge*

$$\{[P_{r_1,i_1}, P_{r_2,i_2}, \cdots, P_{r_n,i_n}] | i_1 = 1, \cdots, m_1, i_2 = 1, \cdots, m_2, \cdots, i_n = 1, \cdots, m_n\}$$

heißt eine t_{SL}-Zeit-Konfliktkombination.

Bei einem Diagramm (D, t_{SL}) mit $t_{SL} > 0$ aus der obigen Definition existieren insgesamt $m_1 \times m_2 \times \cdots \times m_n$ t_{SL}-Zeit Konfliktkombinationen, was eine einfache kombinatorische Überlegung zeigt.

[4] siehe Beispiel 4.3.1

Der Algorithmus

Nachdem die definitorischen Grundlagen geschaffen wurden, erfolgt nun die Vorstellung des Algorithmus in formaler Form. Der Algorithmus hat das folgende Aussehen:

START:

Gegeben: Eine beliebige reduzierte Aktion A.
Gesucht: Die optimale Gesamtbearbeitungszeit sowie optimale Ablaufpläne.

1. Erzeuge ein $\mathcal{RSV}$-Diagramm (D, t_{SL}) mit $t_{SL} = 0$ für A.

2. t_{SL} := `Nächster_Haltepunkt` $((D, t_{SL}))$

3. `Verschiebe_vs-Scan-line-Grundaktion`$((D, t_{SL}))$

4. IF `Berechne_Konfliktkombination` $((D, t_{SL})) = \phi$ THEN
   (D, t_{SL}) :=`Mark` $((D, t_{SL}))$
   IF `Fertig` $((D, t_{SL}))$ THEN
   (D, t_{SL}) ist ein Ablaufplan. Gebe dieses Diagramm (D, t_{SL}) als ein Ablaufplan aus. ELSE
   Gehe zu dem Schritt 2.
   ELSE
   `Vervielfache`$((D, t_{SL}))$.

   /* (D, t_{SL}) sei n mal vervielfacht. Es seien die Diagramme $(D, t_{SL})_1, \cdots, (D, t_{SL})_n$ entstanden, wobei n der Anzahl der Elemente der Menge `Berechne_Konfliktkombination`$((D, t_{SL}))$ entspricht. Jedes Diagramm $(D, t_{SL})_i (i = 1, \cdots, n)$ verfolgt eine t_{SL}-Zeit Konfliktkombination.
   Bei jedem $(D, t_{SL})_i (i = 1, \cdots, n)$ werden die folgenden Anweisungen durchgeführt. */

   (D, t_{SL}) :=`Mark`$((D, t_{SL})_i)$
   Gehe zu dem Schritt 2.

5. Vergleiche die Gesamtbearbeitungszeiten aller berechneten Ablaufpläne und gebe die minimale Gesamtbearbeitungszeit sowie die zugehörigen (optimalen) Ablaufpläne aus.

HALT

Die Teilprozeduren des obigen Algorithmus erfüllen dabei die folgenden Aufgaben:

- Die Teilprozedur `Verschiebe_vs-Scan-line-Grundaktion`$((D, t_{SL}))$ verschiebt alle vs-Scan-line-Grundaktionen hinter die zuletzt eingefrorenen Grundaktionen, wenn solche existieren (siehe Definition 4.3.4).

- Die Teilprozedur `Vervielfache`$((D, t_{SL}))$ vervielfacht das Diagramm $((D, t_{SL}))$ mit dem Haltepunkt der Scan-line $t = t_{SL}$ $(t_{SL} > 0)$ entsprechend der Anzahl der t_{SL}-Zeit Konfliktkombinationen. Jedes der so vervielfachten Diagramme entspricht einer kombinatorischen Möglichkeit unter allen kombinatorischen Möglichkeiten. Anschließend wird bei jedem Diagramm die Teilprozedur `Mark`$((D, t_{SL}))$ mit $t_{SL} > 0$ angewendet.

- Die Teilprozedur `Mark`$((D, t_{SL}))$ $(t_{SL} > 0)$ friert jede t_{SL}-Zeit konfliktfreie Grundaktion sowie alle zu der zugeordneten t_{SL}-Zeit Konfliktkombination gehörigen t_{SL}-Zeit Konfliktgrundaktionen ein, wenn solche existieren. Anschließend werden alle sonstigen t_{SL}-Zeit Konfliktgrundaktionen jeweils hinter die zugehörige, eingefrorene t_{SL}-Zeit Konfliktgrundaktion geschoben.

- Die Teilprozedur `Berechne_Konfliktkombination`$((D, t_{SL}))$ ermittelt die Menge aller t_{SL}-Zeit Konfliktkombinationen.

- Die Teilprozedur `Fertig`$((D, t_{SL}))$ überprüft, ob sich die Scan-line an der äußerst rechten Position befindet, d.h. keine rechten Endpunkte von Grundaktionen rechts von ihr vorkommen.

- Die Teilprozedur `Nächster_Haltepunkt`$((D, t_{SL}))$ läßt die Scan-line zum nächsten Haltepunkt beim Diagramm (D, t_{SL}) springen.

Im folgenden wird die Arbeitsweise des Algorithmus mittels einer Beispielaktion erläutert:

Beispiel 4.3.1. Die Aktion

$$\begin{aligned}
\mathbf{pll}\ & (\mathbf{seq}\ P_1(a,1), (\mathbf{pll}\ P_2(b,1), P_3(d,2)), P_4(c,3)),\\
& (\mathbf{seq}\ P_5(d,2), P_6(a,1)),\\
& P_7(b,3),\\
& (\mathbf{seq}\ P_8(e,2), P_9(d,2))
\end{aligned}$$

stellt das $\mathcal{RSV}$-Diagramm aus Abbildung 4.10 dar. Nach dem Springen der Scan-line zum nächsten Haltepunkt gilt, daß $P_1(a,1)$ die einzige 1-Zeit konfliktfreie Grundaktion ist und weder eine vs-Scan-line-Grundaktion noch eine 1-Zeit Konfliktressource existiert. Für die Grundaktion $P_1(a,1)$ wird der Zeitraum zwischen 0 und 1 als ihr Durchführungszeitraum festgelegt. Durch eine Schattierung wird gekennzeichnet, daß die Grundaktion $P_1(a,1)$ eingefroren ist. Das entstehende Diagramm $(D,1)$ gibt Abbildung 4.11 wieder.

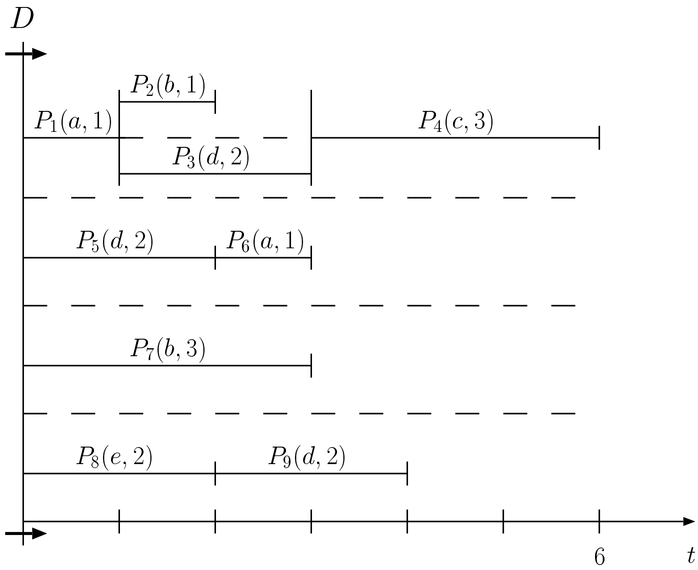


Abbildung 4.10: Ein $\mathcal{RSV}$-Diagramm $(D, 0)$

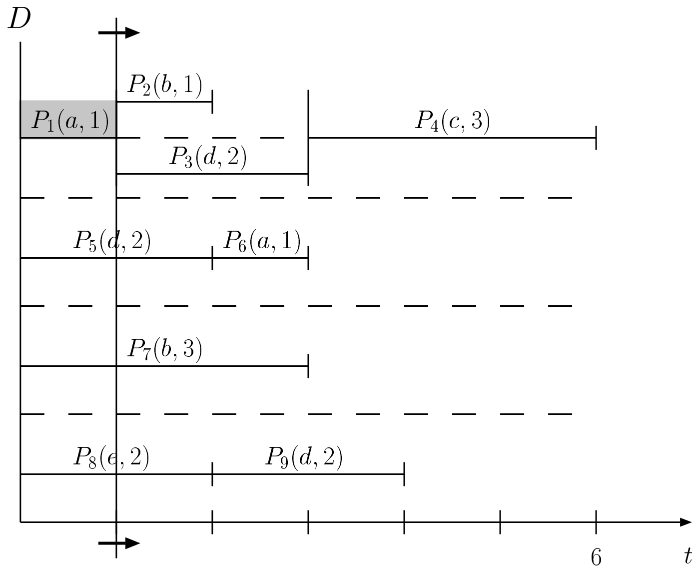


Abbildung 4.11: Ein $\mathcal{RSV}$-Diagramm $(D, 1)$

Nun wird mit der Scan-line zum nächsten Haltepunkt H mit $t(H) = 2$ gesprungen. Es existieren eine 2-Zeit-konfliktfreie Aktion P_8 sowie zwei 2-Zeit Konfliktressourcen b und d. So existieren insgesamt vier 2-Zeit Konfliktkombinationen, diese sind $[P_2, P_3], [P_2, P_5], [P_3, P_7]$ sowie $[P_5, P_7]$. Das Diagramm $(D, 2)$ wird entsprechend der 4 Konfliktkombinationen viermal vervielfältigt, die erhaltenen Diagramme seien mit $D1, \cdots, D4$, bezeichnet. Bei $D1$ werde die Kombination $[P_2, P_3]$

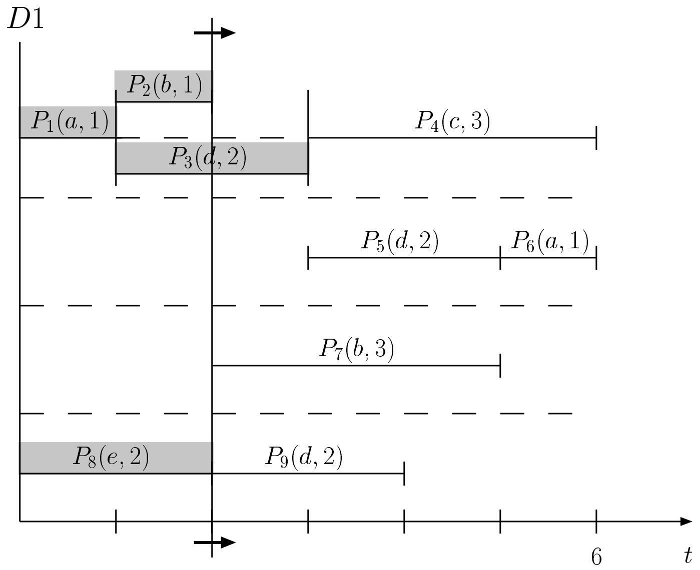


Abbildung 4.12: Ein $\mathcal{RSV}$-Diagramm $(D1, 2)$

weiterverfolgt, bei der die Grundaktionen P_2 und P_3 zuerst durchgeführt werden. P_5 muß in diesem Fall 3 Zeiteinheiten nach rechts geschoben $(RE(P_3) = LE(P_5))$ werden, da die Ressource d durch P_3 blockiert ist. P_7 wird 2 Zeiteinheiten nach rechts geschoben $(RE(P_2) = LE(P_7))$, da die Ressource b durch P_2 blockiert ist. P_2, P_3 sowie die 2-Zeit-konfliktfreie Aktion P_8 werden eingefroren. Das entstehende Diagramm gibt Abbildung 4.12 wieder.

Als nächster Schritt springt die Scan-line beim Diagramm $(D1, 2)$ zum nächsten Haltepunkt H mit $t(H) = 3$. Nun existiert eine vs-Scan-line-Aktion P_9, die die 3-Zeit-blockierte Ressource d benötigt. Daher muß die Aktion P_9 hinter die Aktion P_3 geschoben, die die Ressource d gerade blockiert. Das entstehende Diagramm gibt Abbildung 4.13 wieder.

Im Anschluß springt die Scan-line zum nächsten Haltepunkt H mit $t(H) = 5$. Es existiert eine 5-Zeit-Konfliktressource d und in der Folge insgesamt zwei 5-Zeit Konfliktkombinationen $[P_5]$ und $[P_9]$. Das Diagramm $(D1, 5)$ wird zu zwei Diagrammen, diese seien $D11$, $D12$, vervielfacht. Bei $D11$ werde P_5 zuerst durchgeführt, während bei $D12$ P_9 zuerst durchgeführt werde. Bei $D11$ wird P_9 2 Zeiteinheiten nach rechts geschoben, somit gilt $RE(P_5) = LE(P_9)$. Anschließend werden die 5-Zeit-konfliktfreie-Grundaktion P_7 sowie die ausgewählte Grundaktion P_5 eingefroren. Das entstehende Diagramm zeigt die Abbildung 4.14.

Auf den übrigen Teil der reduzierten Aktion aus Abbildung 4.14 wird derselbe Prozeß rekursiv angewendet. Bei diesen rekursiven Anwendungen entstehen keine weiteren Ressourcenkonflikte mehr. Schließlich wird von $D11$ aus ein Ablaufplan mit der Gesamtbearbeitungszeit von 7 Zeiteinheiten erhalten. Diesen Ablaufplan

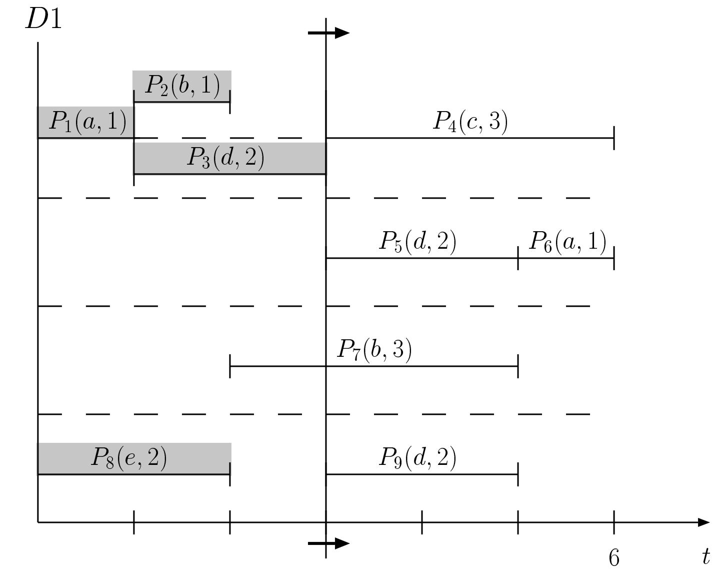


Abbildung 4.13: Ein $\mathcal{RSV}$-Diagramm $(D1, 3)$

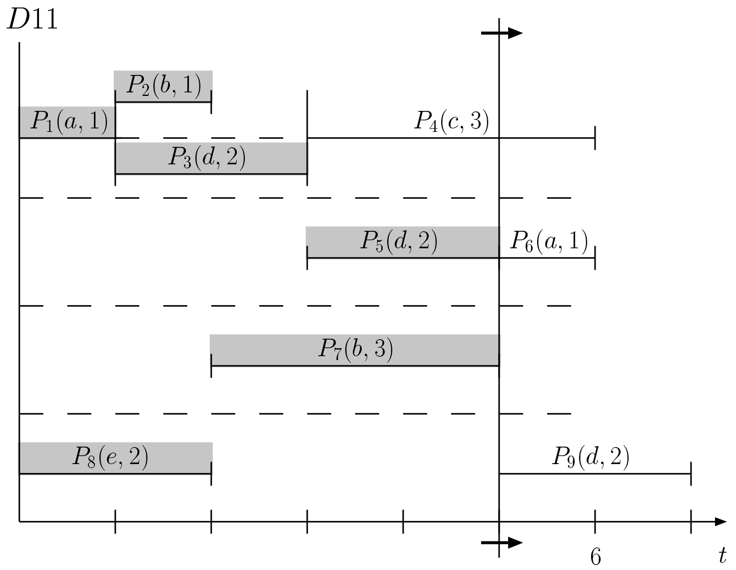


Abbildung 4.14: Ein $\mathcal{RSV}$-Diagramm $(D11, 5)$

zeigt Abbildung 4.15.
Von $D12$ aus wird der in Abbildung 4.16 wiedergegebene Ablaufplan erhalten, der eine Gesamtbearbeitungszeit von 8 Zeiteinheiten benötigt.

Bei den sonstigen Diagrammen $D2$, $D3$ und $D4$ wird obiger Prozeß genauso rekursiv angewendet, wobei jeweils eine weitere Kombination weiterverfolgt wird. So werden von $D2$ aus zwei Ablaufpläne erhalten. Diese geben die Abbildungen

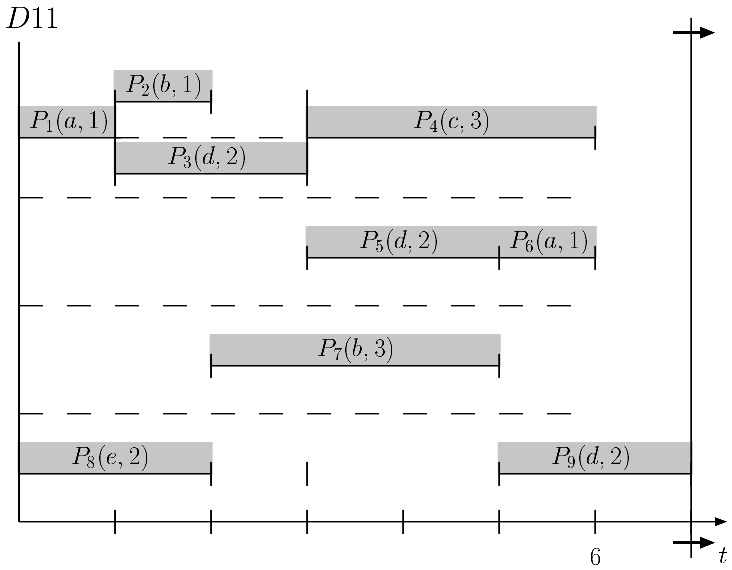


Abbildung 4.15: Ein $\mathcal{RSV}$-Enddiagramm

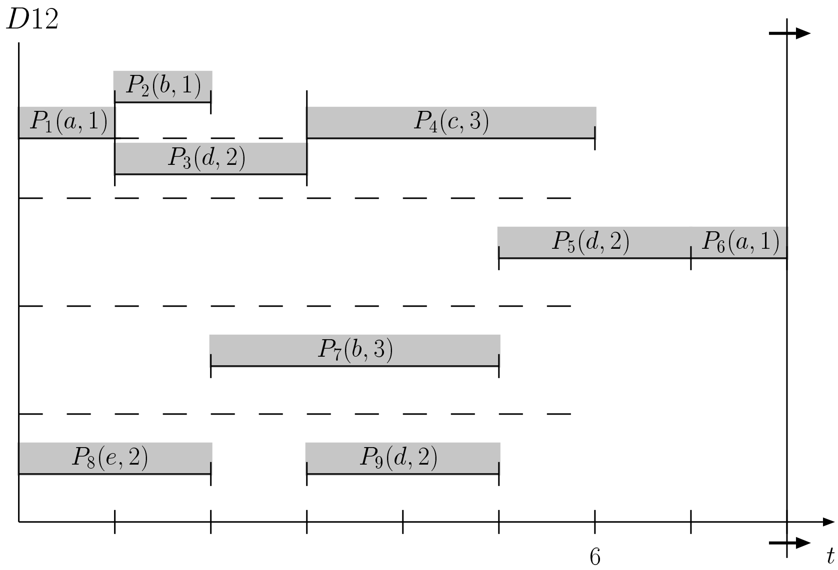


Abbildung 4.16: Ein $\mathcal{RSV}$-Enddiagramm

4.17 und 4.18 wieder.

Insgesamt werden somit 8 verschiedene Ablaufpläne erhalten. Vier davon benötigen eine Gesamtbearbeitungszeit von 7 Zeiteinheiten und sind damit optimal. Die Abbildungen 4.19 und 4.20 zeigen die zwei verbleibenden optimalen Ablaufpläne, die ausgehend von $D3$ und $D4$ aus erhalten werden.

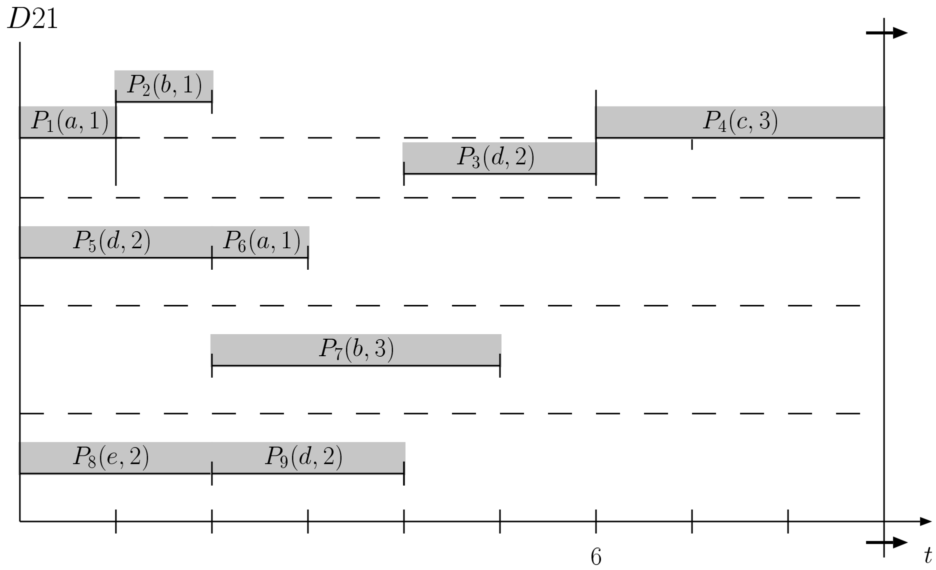


Abbildung 4.17: Ein $\mathcal{RSV}$-Enddiagramm (Ein Ablaufplan)

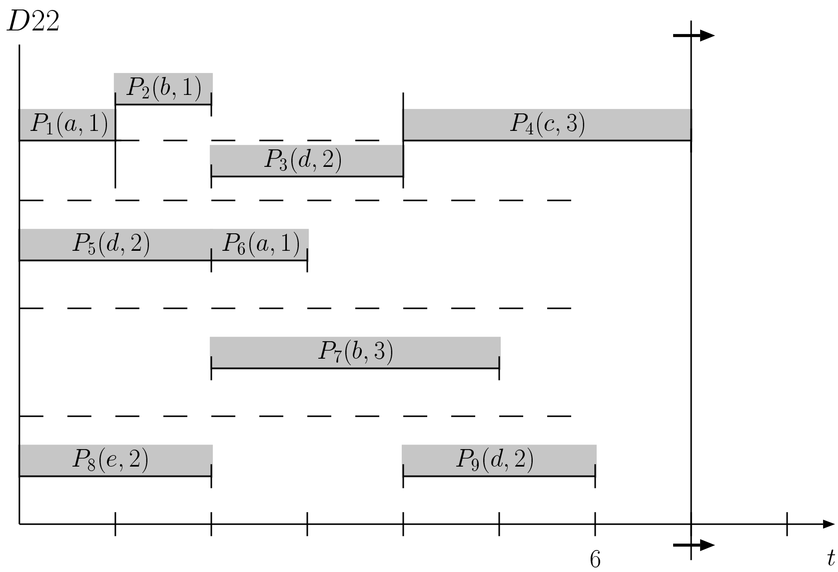


Abbildung 4.18: Ein $\mathcal{RSV}$-Enddiagramm (Ein Ablaufplan)

4.4 Implementierung des Algorithmus

In diesem Abschnitt wird eine konkrete Implementierung des im letzten Abschnitt eingeführten Algorithmus in der funktionalen Programmiersprache 'Haskell' [HAB⁺96], [PHA⁺97] vorgestellt[5].

[5] Das komplette Programm wird am Ende dieser Arbeit beim Anhang angeheftet.

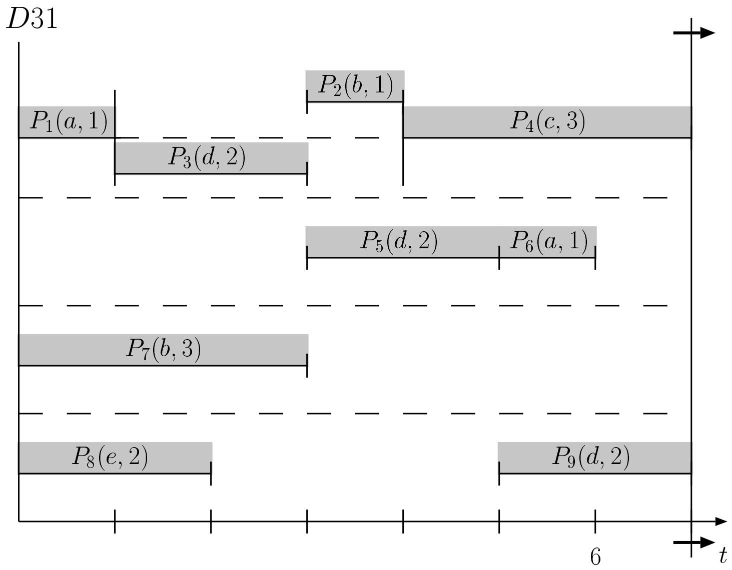


Abbildung 4.19: Ein $\mathcal{RSV}$-Enddiagramm

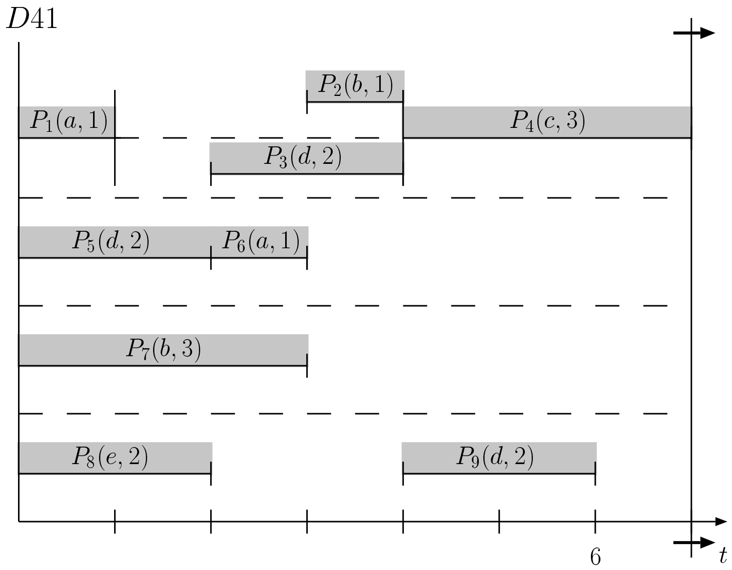


Abbildung 4.20: Ein $\mathcal{RSV}$-Enddiagramm

Bei der Implementierung wird der in Abschnitt 4.3 vorgestellte Algorithmus aus implementatorischen Vereinfachungsgründen leicht variiert. Die Veränderung ist folgender Natur:

> Statt am Anfang für eine als Eingabe gegebene reduzierte Aktion ein $\mathcal{RSV}$-Diagramm zu erzeugen und nach jeder Auflösung von Ressourcenkonflikten Grundaktionen nach rechts zu schieben, wie dies im

Algorithmus geschieht, wird bei der Implementation eine Grundaktion erst dann in ein $\mathcal{RSV}$-Diagramm (hinter den direkten Vorgänger) eingefügt, wenn die Durchführungszeiträume aller ihrer Vorgänger festgelegt worden sind und die von ihr benötigte Ressource zu dem Zeitpunkt nicht blockiert ist (siehe Definition 4.3.4).

Bei dieser Variante wird der überflüssige Berechnungsaufwand für die Verschiebungen der Grundaktionen gespart.

Zunächst definieren wir eine algebraische Datenstruktur zur Repräsentation von Aktionen:

```
>type Name = [Char]
>type Resource = Char

>data
>   Action =  GroundAction Name Resource Int
>           | SEQ [Action]
>           | PLL [Action]
```

Die Aktion

$$\texttt{action:=}\ \mathbf{pll}\ (\mathbf{seq}\ P_1(a,1), P_2(b,2)),\\ P_3(a,2)$$

wird mittels obiger Datenstruktur wie folgt dargestellt:

```
>gA1    = GroundAction "P1" 'a' 1
>gA2    = GroundAction "P2" 'b' 2
>gA3    = GroundAction "P3" 'a' 2
>action = PLL [SEQ [gA1, gA2], gA3]
```

Der Algorithmus besteht aus mehreren Berechnungsschritten. Im Rahmen des ersten Berechnungsschrittes geschieht folgendes:

Anfügen aller Startaktionen bzw. weiterer Aktionen an die Scan-line und Scan-line bewegen

Zunächst werden alle Grundaktionen der *aktuellen* Eingabeaktion, die an die Scan-line angefügt werden können, ermittelt und an die Scan-line angefügt. "An die Scan-line anfügen" bedeutet, daß bei allen anzufügenden Grundaktionen die Durchführungsstartzeit `l`, auf den Zeitpunkt gesetzt wird, an dem sich die Scan-line befindet. Eine Grundaktion P_i kann an die Scan-line angefügt werden gdw. sie die folgenden Bedingungen erfüllt:

1. P_i besitzt bei der aktuellen Eingabeaktion keinen Vorgänger.

2. P_i ist noch nicht in das Diagramm aufgenommen worden.

3. P_i beansprucht *keine* `l`*-Zeit blockierte Ressource*[6] , wobei `l` dem Zeitpunkt entspricht, auf dem sich die Scan-line befindet.

Anschließend springt die Scan-line zum nächsten Haltepunkt. Die Funktion, die diese Aufgaben erfüllt, trägt die Bezeichnung `evalStep1` und wird wie folgt definiert:

```
>evalStep1 l as cs a = evalStep2 l' as' cs' a
>                       where
>                         l'  = nextScanline l as'
>                         ps  = getParallel a
>                         qs  = foldl fRemove ps cs
>                         as' = foldl (addAction l) as qs
>                         cs' = cs \\ [x| x <- cs, (snd x) <= l']
```

- Das erste Argument `l` der Funktion `evalStep1` bezeichnet den Zeitpunkt, auf dem sich die Scan-line befindet.

- Das zweite Argument, die Liste `as`, beschreibt mit der Scan-line `l` zusammen ein $\mathcal{RSV}$-Diagramm, aus dem die festgelegten Zeiträume von Grundaktionen abgelesen werden können. Diese Liste bezeichnen wir als `ScheduleInfo` und besitzt die folgende Typdeklaration:

  ```
  ScheduleInfo :: [(Name, Resource, BeginTime, EndTime)]
  BeginTime    :: Int
  EndTime     :: Int
  ```

- Die Liste `cs :: [(Resource, EndTime)]` besteht aus Tupeln. Die erste Komponente eines Tupels beschreibt dabei eine `l`-Zeit-blockierte Ressource und die zweite Komponente den rechten Endpunkt der zugehörigen eingefrorenen Grundaktion. Bei einer solchen Grundaktion P_i gilt bei `as` immer die Eigenschaft $LE(P_i) < \mathtt{l} < RE(P_i)$, da sie an einem Ressourcenkonflikt beteiligt war und eingefroren wurde. Die zugehörige Ressource bleibt bis zum Zeitpunkt $RE(P_i)$ blockiert.

- `a` stellt die aktuelle Eingabeaktion dar.

[6] siehe Definition 4.3.4

Das Anfügen aller Startaktionen bzw. weiterer Aktionen an die Scan-line geschieht wie folgt:

Zunächst wird die Funktion `getParallel` auf die Aktion `a` angewendet (`ps = getParallel a`). Die Funktion `getParallel` ermittelt alle Grundaktionen, die bei `a` keinen Vorgänger besitzen. Die Definition der Funktion `getParallel` lautet:

```
getParallel :: Action -> [(Name, Resource, Int)]
getParallel (GroundAction n r l) = [(n, r, l)]
getParallel (SEQ (x:xs))         = (getParallel x)
getParallel (PLL xs    )         = concat (map getParallel xs)
```

Die Aufgabe der Funktion `getParallel` verdeutlicht das folgende Beispiel:

Beispiel 4.4.1. Wenn die Funktion `getParallel` auf die Aktion `action1` aus Beispiel 4.3.1

$$\begin{aligned}\texttt{action1}:=\textbf{pll}\ &(\textbf{seq}\ P_1(a,1), (\textbf{pll}\ P_2(b,1), P_3(d,2)), P_4(c,3)),\\ &(\textbf{seq}\ P_5(d,2), P_6(a,1)),\\ &P_7(b,3)\end{aligned}$$

angewendet wird, erhalten wir die folgende Ausgabe:

```
> [("P1", 'a', 1), ("P5", 'd', 2), ("P7", 'b', 3)]
```

Anschließend werden aus `ps` alle Grundaktionen entfernt, die eine `l`-Zeit blockierte Ressource (Elemente aus der Liste `cs`) beanspruchen. Darauf basierend wird die Liste `qs`

```
qs           = foldl fRemove ps cs
fRemove ps c = ps \\ [x|x<-ps, (snd3 x)==(fst3 c)]
```

berechnet.
Mit diesem Programmteil wird die Teilprozedur

`Verschiebe_vs-Scan-line-Grundaktion` ((D,`l`))

mit dem Haltepunkt der Scan-line $t(H) =$`l` realisiert.

Als nächster Schritt werden alle Grundaktionen aus `qs`, die noch nicht in `as` aufgenommen worden sind, beim Diagramm (`as, l`) an die Scan-line angefügt. Das Ergebnis wird in der Liste `as'` festgehalten, die wie folgt definiert ist:

```
as' = foldl (addAction l) as qs
```

Die Hilfsfunktion `addAction` verfügt dabei über folgende Definition:

```
addAction :: Int -> ScheduleInfo -> (Name, Resource, Int)
                                 -> ScheduleInfo
addAction l as (n, r, t)
     | (elem n (map getName as)) = as
     | otherwise                 = (n, r, l, l + t):as
```

Wir greifen nun das zuvor betrachtete Beispiel wieder auf, um die Arbeitsweise der oben vorgestellten Teile der Implementation zu verdeutlichen.

Beispiel 4.4.2. Am Anfang sind die Listen `as` sowie `cs` leer und die Scan-line steht auf dem Haltepunkt $t(H) = 0$. `qs` entspricht zu diesem Zeitpunkt der Liste `ps` und damit dem Ergebnis von `getparallel action1`. Nach dem Anfügen aller Grundaktionen aus `qs` an die Scan-line wird die folgende Liste `as'` erhalten. Wie bereits erwähnt, wird als Startzeit jeder Grundaktion der gegenwärtige Haltepunkt der Scan-line `l = 0` gesetzt und wir erhalten bei unserem Beispiel für `as'`:

```
> [("P7", 'b', 0, 3), ("P5", 'd', 0, 2), ("P1", 'a', 0, 1)]
```

Nun springt die Scan-line mittels der Funktion `nextScanline` vom Haltepunkt `l` zum nächsten Haltepunkt, der durch die in der Liste `as'` mitbeinhalteten Endpunkte bestimmt wird:

```
l' = nextScanline l as'.
```

Die Hilfsfunktion `nextScanline` wird wie folgt definiert:

```
nextScanline l as = minimum0 (filter (\t -> l < t)
                                (map getEndTime as))
```

Der soeben vorgestellte Programmteil entspricht der Umsetzung der Teilprozedur

`Nächst_Haltepunkt` $((D,\mathtt{l}))$

aus der Beschreibung des Algorithmus im vorherigen Abschnitt. Wir greifen wiederum unser Beispiel auf und führen es fort:

Beispiel 4.4.3. Die Scan-line befindet sich auf dem Haltepunkt `l = 0`. Somit liefert die Auswertung des Ausdrucks `nextScanline l as'` bei unserem Beispiel zu diesem Zeitpunkt den Wert `1`.

Mit der Scan-line wurde im letzten Schritt zum nächsten Haltepunkt gesprungen, der in `l'` festgehalten wird. Alle Elemente `x` aus `cs`, für die `(snd x)`$\leq$ `l'` gilt, werden aus `cs` entfernt. Das Ergebnis wird in der Liste `cs'` mit der Definition

```
cs' = cs \\ [x| x <- cs, (snd x) <= l']
```

festgehalten. Die Entfernung der Elemente aus `cs` ist erforderlich, da die Scan-line die zugehörigen Grundaktionen überschritten hat (siehe Definition der Funktion `evalStep1`) und die zugehörigen Ressourcen wieder freigegeben werden müssen.

Im weiteren wird eine Funktion `evalStep2` vorgestellt, die weiteren Berechnungsschritt realisiert. Die Funktion `evalStep2` verfügt über 4 Argumenten, die semantisch äquivalent zu den 4 Argumenten der Funktion `evalStep1` sind.

Ressourcenkonflikte bestimmen
Es wird geprüft, ob `l`-Zeit-Ressourcenkonflikte entstehen.

```
>evalStep2 l as cs a
>       = evalStep3 l as bs cs a ccs
>         where
>           bs  = scanLineAct l as
>           ccs = rcombine (map
>                   (\r -> map getName
>                   (filter (\g -> r == getResource g) bs)
>                    )
>                   (confResr (scanLineRes l bs) bs)
>                           )
```

Die Hilfsfunktion `scanLineAct l as` ermittelt eine Liste `bs`, die aus allen `l`-Zeit-Scan-line-Grundaktionen P_i der Liste `as` besteht, d.h. bei der Liste `as` gilt für alle P_i die Eigenschaft $LE(P_i) <$ `l` $\leq RE(P_i)$. Die Hilfsfunktion `scanLineAct` verfügt über folgende Definition:

```
>scanLineAct :: Int -> WorkingInfo -> WorkingInfo
>scanLineAct l as
>   = filter (\g -> (l == getEndTime g) ||
>            (((getStartTime g) < l) && (l < (getEndTime g)))
>             ) as
```

Die weitere Hilfsfunktion `rcombine` berechnet eine Liste `ccs`, die aus allen `l`-Zeit-Konfliktkombinationen besteht. Das Argument der Funktion `rcombine` entspricht einer Liste von Mengen. Jede Menge besteht dabei aus `(l, r)`-Konfliktgrundaktionen. Die Berechnung dieser Mengen leistet die Hilfsfunktion `confResr`, die alle `l`-Zeit-Konfliktressourcen berechnet, und wie folgt definiert ist:

```
>confResr :: [Resource] -> WorkingInfo -> [Resource]
>confResr [] _    = []
>confResr (r:rs) bs
>      | 1 < (length (filter (\g -> r == getResource g) bs))
>                      = r:(confResr rs bs)
>      | otherwise     = confResr rs bs
```

Die Funktion `rcombine` verfügt über folgende Definition:

```
>ccs = rcombine :: [[Name]] -> [[Name]]
>rcombine [] = []
>rcombine (xs:xxs)
>   = case xxs of
>         [] -> ts
>         _  -> concat (map (\z -> (map (\x -> x ++ z) ts))
>                           (rcombine xxs)
>                       )
>                    where
>                      ts = (transpose [xs])
```

Alle `l`-Zeit-direkten Scan-line-Ressourcen werden mittels der Hilfsfunktion `scanLineRes` berechnet, deren Definition dem Anhang zu entnehmen ist. Der soeben vorgestellte Programmteil entspricht der Umsetzung der Teilprozedur

`Berechne_Konfliktkombination` $((D,\mathtt{l}))$

Die Wirkungsweise der beiden Funktionen `scanLineAct` und `rcombine` soll das folgende Beispiel verdeutlichen, das einer Fortsetzung von Beispiel 4.4.3 entspricht:

Beispiel 4.4.4. Die Werte von `l` und `as` seien entsprechend Beispiel 4.4.3 gesetzt. Dann liefert die Auswertung von `scanLineAct l as` als Ergebnis die Liste:

```
[("P7", 'b', 0, 3), ("P5", 'd', 0, 2),
 ("P1", 'a', 0, 1)]
```

Der Wert vom `bs` entspreche im weiteren der obigen Liste. Den liefert die Auswertung von `confResr (scanLineRes l bs) bs` als Ergebnis die leere Liste. Es entsteht somit kein Ressourcenkonflikt zwischen dem letzten Haltepunkt `l = 0` und dem gegenwärtigen Haltepunkt der Scan-line `l = 1`. Entsprechend wäre die Liste `ccs` zu diesem Zeitpunkt leer.

Ressourcenkonflikte auflösen

Die Auflösung der `l`-Zeit-Ressourcenkonflikte leistet eine Funktion `evalStep3`, die über 5 Argumente verfügt, deren Semantik aus ihren Bezeichnungen und dem zuvor ausgesagten abgelesen werden kann (siehe Definition der Funktion `evalStep2` usw.). Die Funktion `evalStep3` verfügt über die folgende Definition:

```
>evalStep3 l as bs cs a ccs = case ccs of
>                  [] -> evalStep4 l as cs a
>                  _  -> concat (map execBranch ccs)
>                     where
>                       execBranch ns = evalStep4 l as' cs' a
>                               where
>                                 as' = transAction as bs ns
>                                 cs' = cs ++ (contRK l bs ns)
```

Ist die als Argument gegebene Liste `ccs` leer, wird die Funktion `evalStep4` mit den Werten von `l, as, cs, a` aufgerufen. Ist letzteres nicht der Fall, wird die Hilfsfunktion `execBranch` auf *jede Komponente* der Liste `ccs` angewendet, was einer Modellierung der Teilprozedur

`Vervielfache` ((D,`l`))

entspricht. Jede Komponente `ns` der Liste `ccs` entspricht einer `l`-Zeit-Konfliktkombination. Die Funktion `execBranch` ruft weitere Hilfsfunktionen wie z. B. `transAction` und `contRK` auf, um die folgenden zwei Aufgaben zu erfüllen, die im Rahmen der Vorstellung der Teilprozedur `Mark`$((D, t_{SL}))$ beschrieben wurden:

- Mittels der Funktion `transAction` werden bei `as` alle sonstigen `l`-Zeit-Konfliktgrundaktionen jeweilig hinter die ausgewählte (eine zu `ns` gehörige) `l`-Zeit-Konfliktgrundaktion geschoben. Das Ergebnis wird in der Liste `as'` mit der Definition

  `as' = transAction as bs ns`

  festgehalten.

- Unter Zuhilfenahme der Funktion `contRK` werden alle aktuellen `l`-Zeit-blockierten Ressourcen berechnet. Gilt für eine `l`-Zeit-Konfliktgrundaktion P_i aus `ns` die Eigenschaft $RE(P_i) > \texttt{l}$, so wird ein Tupel $(r, RE(P_i))$ gebildet. Das Tupel besagt, daß die Ressource r bis zum Zeitpunkt $RE(P_i)$ durch die eingefrorene Grundaktion P_i blockiert wird. Jedes so gebildete Tupel $(r, RE(P_i))$ wird zur Liste `cs` hinzugefügt und das Ergebnis in der Liste `cs'` festgehalten:

```
cs' = cs ++ (contRK l bs ns)
```

Die genauen Definitionen der Funktionen `transAction` sowie `contRK` können dem Anhang entnommen werden. Die Werte `l, a` sowie die beiden ermittelten Listen `as'` und `cs'` werden im Anschluß als Argumente an die Funktion `evalStep4` übergeben.

Löschen der `l`-Zeit-direkten Scan-line-Grundaktionen von der Eingabeaktion

Wir betrachten nun den letzten Berechnungsschritt modelliert in der Funktion `evalStep4`, bei dem alle `l`-Zeit-direkten Scan-line-Grundaktionen aus der Eingabeaktion `a` gelöscht werden:

```
>evalStep4 l as cs a
>       = case a' of
>            NOTHING       -> [as]
>            (SOMETHING a) -> evalStep1 l as cs a
>        where
>         a' = foldl (\a n -> delGAExt n a) (SOMETHING a) ds
>              where
>                ds = (map getName
>                     (filter (\a -> (getEndTime a) <= l) as)
>                      )
```

- `a'` bezeichnet die von `a` reduzierte Aktion.

Liefert `a'` den Wert `NOTHING` (dies entspricht im Algorithmus, daß der Test `Fertig` $((D, \texttt{l})$ als Ergebnis den Wert WAHR liefert), dann ist dies gleichbedeutend damit, daß die Scan-line alle Grundaktionen überschritten hat. In diesem Fall wird ein Ablaufplan `[as]` zurückgegeben. Im anderen Fall, werden die Werte `l, as, cs` und `a` (letzteres wird durch ein "entpacken" von `a'` erhalten) als Argumente der Funktion `evalStep1` übergeben. Der ganze Prozeß wird somit auf `a` *rekursiv* angewendet.

Die Betrachtung von Beispiel 4.4.4 soll nun fortgesetzt werden:

Beispiel 4.4.5. Da bei unserem Beispiel die Liste `ccs` zu dem zuletzt betrachteten Zeitpunkt leer ist, werden die Werte `l, as, cs` und `a` von `evalStep3` zu `evalStep4` als Argumente weitergeleitet. Durch `evalStep4` wird die `l`-Zeit konfliktfreie Grundaktion P_1 aus der Eingabeaktion `action1` gelöscht. So erhalten wir die folgende reduzierte Eingabeaktion `a'`:

$$\begin{aligned} \texttt{a'}= \ \mathbf{pll}\ & (\mathbf{seq}\,(\mathbf{pll}\ P_2(b,1), P_3(d,2)),\ P_4(c,3)), \\ & (\mathbf{seq}\ P_5(d,2), P_6(a,1)), \\ & P_7(b,3) \end{aligned}$$

Durch weitere rekursive Anwendung der Funktion `evalStep1` auf `a'` werden folgende Werte erhalten:

```
l'  == 2
as' == [("P3", 'd', 1, 3), ("P2", 'b', 1, 2),
        ("P7", 'b', 0, 3), ("P5", 'd', 0, 2),
                           ("P1", 'a', 0, 1)]
cs' == []
```

Im Rahmen der Funktion `evalStep2` werden die folgenden beiden Listen berechnet:

```
bs  == [("P3", 'd', 1, 3), ("P2", 'b', 1, 2),
        ("P7", 'b', 0, 3), ("P5", 'd', 0, 2)]
ccs == [["P3", "P2"], ["P5", "P2"], ["P3", "P7"], ["P5", "P7"]]
```

`ccs` ist zu diesem Zeitpunkt somit keine leere Liste mehr. Daher wird bei der Funktion `evalStep3` auf jede Komponente der Liste `ccs` die Funktion `execBranch` angewendet. Dies ist gleichbedeutend damit, daß 4 weitere Berechnungswege (vervielfachte Diagramme) verfolgt werden. Nun verfolgen wir die Berechnung, bei der die beiden Grundaktionen `P3` und `P2` eingefroren werden.
Durch Anwendung der Funktion `execBranch` auf die Liste `["P3", "P2"]` werden die beiden folgenden Listen erhalten:

```
as' == [("P3", 'd', 1, 3), ("P2", 'b', 1, 2),
        ("P1", 'a', 0, 1), ("P5", 'd', 3, 5),
                           ("P7", 'b', 2, 5)]
cs' == [('d', 3)]
```

Im Anschluß wird die Eingabeaktion verkleinert, indem die von der Scan-line überschrittene Grundaktion P_2 aus der Eingabeaktion entfernt wird. Als aktuelle Eingabeaktion erhalten wir damit:

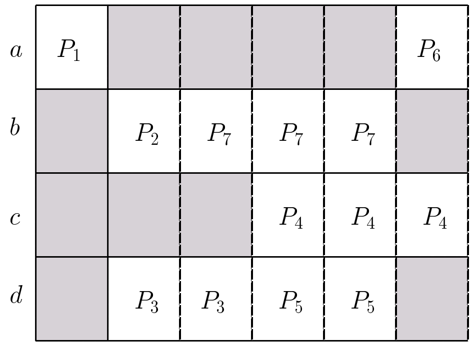


Abbildung 4.21: Ein ausgegebener Ablaufplan

$$\begin{aligned}\texttt{a'=}\ \mathbf{pll}\ & (\mathbf{seq}\, P_3(d,2),\ P_4(c,3)),\\ & (\mathbf{seq}\, P_5(d,2), P_6(a,1)),\\ & P_7(b,3)\end{aligned}$$

Als Ergebnis der weiteren Berechnung erhalten wir folgende Liste `as`, die einen Ablaufplan darstellt:

```
as == [("P1", 'a', 0, 1), ("P2", 'b', 1, 2), ("P3", 'd', 1, 3),
       ("P5", 'd', 3, 5), ("P7", 'b', 2, 5), ("P4", 'c', 3, 6),
                                              ("P6", 'a', 5, 6)]
```

Auf diese Liste `as` werden die Funktionen, die zum Zweck der graphischen Ergebnisdarstellung am Bildschirm definiert sind, angewendet (siehe unten aufgeführte Erläuterungen). Den entsprechenden ausgegebenen Ablaufplan zeigt Abbildung 4.21.

Um die Ablaufpläne darstellenden Listen am Bildschirm ausgeben zu können, werden drei Funktionen `matrix1, matrix2` sowie `evalP´`[7] definiert und die zwei Prelude-Funktionen von Haskell `putStr` und `sequence` benutzt[8]. Die prelude-Funktion `putStr` dient dazu, einen String (Liste von Buchstaben) am Bildschirm auszugeben und entspricht einem monadischen Ausdruck. Um mehrere monadische Ausdrücke, die als Ergebnis der Anwendung der Funktion `map` entstehen (siehe unten aufgeführte Definition), aneinander zu hängen, wird die Prelude-Funktion `sequence` angewendet. Die Funktionsdefinition, die für eine Eingabe-Aktion '`a`' die Ergebnisausgabe leistet, sieht wie folgt aus:

[7] Die Definition der Funktionen `matrix1, matrix2` und `evalP´` kann dem Anhang entnommen werden.

[8] Prelude-Funktionen sind Standarddefinitionen, die in Haskell beinhaltet sind. Das Standard-Prelude von Haskell kann [PHA+97] entnommen werden.

```
>evalP a = sequence (map putStr (evalP´ a))
```

Im folgenden Abschnitt führen wir einen Korrektheitsbeweis für die $\mathcal{RSV}$-Diagrammberechnung.

4.5 Korrektheitsbeweis der $\mathcal{RSV}$-Diagrammberechnung

Es wird nun auf Basis des im letzten Abschnitt vorgestellten Haskell-Programms formal bewiesen, daß die $\mathcal{RSV}$-Diagrammberechnung für eine beliebige reduzierte Aktion alle nichtredundanten Ablaufpläne berechnet und die optimale Gesamtbearbeitungszeit sowie alle zugehörigen optimalen Ablaufpläne ermittelt. Dazu wird zunächst bewiesen, daß die Funktion '`getParallel`' alle Grundaktionen ermittelt, die bei einer gegebenen Eingabeaktion keinen Vorgänger besitzen. Zur Vereinfachung der folgenden Beweise wird folgende Notation eingeführt:

Notation 4.5.1. *Eine mit '**pll**' ('**seq**') verbundene reduzierte Aktion wird als* **pll**-Aktion *(***seq**-Aktion*) bezeichnet.*

Lemma 4.5.1. *Die Funktion '`getParallel`' ermittelt für eine beliebige reduzierte Aktion alle Grundaktionen, die keinen Vorgänger besitzen.*

Beweis. Wir zeigen durch strukturelle Induktion über den Formelaufbau (Aktionsaufbau) von einer beliebigen reduzierten Aktion A, daß `getParallel` alle Grundaktionen ermittelt, die keinen Vorgänger besitzen.

Induktionsanfang: Falls A eine Grundaktion ist, besitzt A keinen Vorgänger. Nach Definition der Funktion `getParallel` wird die Grundaktion A zurückgegeben.

Induktionsschritt: Es sind zwei Fälle, eine **seq**-Aktion (*Fall* 1) und eine **pll**-Aktion (*Fall* 2), zu unterscheiden.
Fall 1: A hat die Form $(\mathbf{seq} F_1, \cdots, F_m)$, wobei $F_1, \cdots F_m$ die Argumente des Operators '**seq**' sind. '**seq**' spezifiziert die sequentielle Abarbeitung von $F_1, \cdots$ bis F_m. In der Folge sind alle Grundaktionen, die bei A keinen Vorgänger besitzen, diejenigen, die bei F_1 keinen Vorgänger besitzen. Nach der Definition der Funktion `getParallel` für eine **seq**-Aktion

getParallel (**seq** x:xs) = (getParallel x)

wird `getParallel` auf F_1 angewendet, wobei F_1 strukturell kleiner als A ist. Nach Induktionsannahme ermittelt `getParallel` alle Grundaktionen, die bei F_1 keinen Vorgänger besitzen.
Fall 2: A hat die Form $(\mathbf{pll}\ F_1, \cdots, F_m)$, wobei $F_1, \cdots F_m$ die Argumente des

Operators '**pll**' sind. '**pll**' spezifiziert die parallele Abarbeitung von $F_1, \cdots, F_m$. Alle Grundaktionen, die bei A keinen Vorgänger besitzen, müssen der Vereinigung aller Grundaktionen entsprechen, die jeweils bei $F_1, \cdots$ und F_m keinen Vorgänger besitzen. Nach der Definition der Funktion `getParallel` für eine **pll**-Aktion

```
getParallel (pll xs) = concat (map getParallel xs)
```

wird die Funktion `getParallel` einzeln auf F_i für $i = 1, \cdots, m$ angewendet (die Funktion `map`) und die so entstehenden Mengen mittels der Funktion `concat` vereinigt.
Jedes Argument $F_i (i = 1, \cdots, m)$ ist offensichtlich strukturell kleiner als A. So ermittelt `getParallel` nach Induktionsannahme für $F_i (i = 1, \cdots, m)$ alle Grundaktionen, die keinen Vorgänger besitzen. □

Das folgende Lemma beweist, daß die $\mathcal{RSV}$-Diagrammberechnung stets terminiert.

Lemma 4.5.2. *Die $\mathcal{RSV}$-Diagrammberechnung terminiert angewendet auf eine beliebige reduzierte Aktion.*

Beweis. Die $\mathcal{RSV}$-Diagrammberechnung terminiert, da bei dem rekursiven Aufruf der Funktion `evalStep1` die strukturelle Größe der berechneten Aktion immer kleiner wird. Wir zeigen dies durch strukturelle Induktion über den Formelaufbau (Aktionsaufbau) von einer beliebigen reduzierten Aktion A.

Induktionsanfang: Falls A eine Grundaktion ist, terminiert die $\mathcal{RSV}$-Diagrammberechnung trivialerweise.

Induktionsschritt: Am Anfang befindet sich die Scan-line auf dem Zeitpunkt 0. Zunächst werden alle Startgrundaktionen mittels der Funktion `getParallel` ermittelt (siehe Lemma 4.5.1) und diese anschließend an die Scan-line angefügt. Danach springt die Scan-line vom Zeitpunkt 0 zum nächsten Haltepunkt l. $G_1, G_2, \cdots, G_n$ seien alle Startgrundaktionen, deren rechter Endpunkt zu diesem Zeitpunkt l entspricht, sich also auf der Scan-line befindet ($G_1, G_2, \cdots, G_n$ seien zu diesem Zeitpunkt alle l-Zeit direkten Scan-line-Grundaktionen). Es sind nun folgende drei Fälle zu unterscheiden:
Fall 1: Alle Grundaktionen $G_1, G_2, \cdots, G_n$ sind l-Zeit-konfliktfreie Aktionen. Da in diesem Fall kein l-Zeit-Ressourcenkonflikt entsteht, wird zum nächsten Berechnungsschritt übergegangen, bei dem alle Aktionen $G_1, G_2, \cdots, G_n$ aus der Eingabeaktion A gelöscht werden. Somit wird die Eingabeaktion strukturell kleiner. Nach Induktionsannahme terminiert die $\mathcal{RSV}$-Diagrammberechnung, da im weiteren die $\mathcal{RSV}$-Diagrammberechnung auf diese verkleinerte Aktion rekursiv angewendet wird.
Fall 2: Es entstehen einer oder mehrere Ressourcenkonflikte, d.h. es existieren eine oder mehrere l-Zeit-Konfliktressourcen. Ferner existiert mindestens eine Grundaktion G_i, die einer l-Zeit-konfliktfreien Grundaktion entspricht. Zunächst werden

die Ressourcenkonflikte aufgelöst. Hierbei werde das Diagramm zu k Diagrammen $D_1, \cdots, D_k$ vervielfacht, so daß jeder l-Zeit-Konfliktkombination ein eigenes Diagramm zugeordnet wird. Nach der Auflösung der Ressourcenkonflikte werden bei jedem Diagramm D_i alle l-Zeit-direkten-Scan-line-Grundaktionen aus der Eingabeaktion A gelöscht. Die Eingabeaktion wird hierbei strukturell kleiner, da mindestens die l-Zeit-konfliktfreien Grundaktionen G_i aus A gelöscht werden. Im nächsten Schritt wird die $\mathcal{RSV}$-Diagrammberechnung auf die verkleinerte Eingabeaktion rekursiv angewendet und so terminiert nach Induktionsannahme jede $\mathcal{RSV}$-Diagrammberechnung D_i.
Fall 3: Jede Grundaktion G_i ist an einem Ressourcenkonflikt beteiligt, d.h. es existiert keine l-Zeit-konfliktfreie Grundaktion. Zunächst werden in diesem Fall die Ressourcenkonflikte aufgelöst. Das Diagramm werde dabei zu k Diagrammen $D_1, \cdots, D_k$ vervielfacht, so daß jeder l-Zeit-Konfliktkombination ein eigenes Diagramm zugeordnet wird. Hierbei sind zwei Subfälle zu unterscheiden:
Fall 31: Mindestens eine l-Zeit direkte Scan-line-Grundaktion G_i wird eingefroren. Dieser Fall ist dem zweiten Fall sehr ähnlich und kann entsprechend gezeigt werden.
Fall 32: Keine l-Zeit direkte Scan-line-Grundaktion G_i wird eingefroren. Bei diesem Fall müssen weitere l-Zeit-Konfliktgrundaktionen existieren, die einzufrieren sind. Diese Aktionen seien $H_1 \cdots, H_m$, wobei $RE(H_j) > l$ für jede j gelten muß. Früher oder später werden die Grundaktionen $H_1 \cdots, H_m$ aus der Eingabeaktion gelöscht. Die in der Folge erhaltenen Eingabeaktionen sind strukturell kleiner, daraus ergibt sich in Verbindung mit der Induktionsannahme direkt die Terminierung bei diesem Fall. □

Lemma 4.5.3. *Jeder mittels der $\mathcal{RSV}$-Diagrammberechnung erhaltene Ablaufplan erfüllt die vorgegebenen Präzedenzbedingungen.*

Beweis. Jede Grundaktion G wird aus der Eingabeaktion gelöscht, nachdem die Scan-line den rechten Endpunkt von G überschritten hat. Ferner wird jede Grundaktion an die Scan-line angefügt, wenn sie bei der aktuellen Eingabeaktion keinen Vorgänger mehr besitzt. So müssen alle Vorgänger einer Grundaktion bei der aktuellen Eingabeaktion gelöscht sein, damit diese an die Scan-line angefügt werden kann. Dies heißt, daß bei einem $\mathcal{RSV}$-Diagramm jede Grundaktion nur hinter ihrem (direkten) Vorgänger vorkommen kann. □

Lemma 4.5.4. *Jeder mittels der $\mathcal{RSV}$-Diagrammberechnung erhaltene Ablaufplan ist ressourcenkonfliktfrei.*

Beweis. Für jeden Zeitraum, bei dem mehr als eine Grundaktion dieselbe Ressource benötigt, wird genau eine Grundaktion ausgewählt und eingefroren. Alle sonstigen Grundaktionen werden hinter die eingefrorene Grundaktion geschoben.

Ferner wird die Ressource solange blockiert, bis die Ressource durch die Grundaktion wieder freigegeben wird. Alle weiteren eine blockierte Ressource benötigenden Grundaktionen müssen warten, bis die Ressource wieder freigegeben wird. □

Lemma 4.5.5. *Jeder mittels der $\mathcal{RSV}$-Diagrammberechnung erhaltene Ablaufplan ist ein aktiver Ablaufplan.*

Beweis. In Lemmata 4.5.3, 4.5.4 wurde die Erfüllung der Präzedenzbedingungen sowie der Ressourcenkonfliktfreiheit gezeigt. Nun wird induktiv gezeigt, daß bei einem beliebigen mittels der $\mathcal{RSV}$-Diagrammberechnung erhaltenen Ablaufplan $\mathcal{P}$ jede Grundaktion *frühestmöglich* bei der zugehörigen Ressource eingeplant wird.

Die Grundaktion, die bei der Berechnung von $\mathcal{P}$ zuerst aus der Eingabeaktion gelöscht wird, erhält die Startzeit 0 und ist damit frühestmöglich eingeplant. Angenommen, alle Grundaktionen bis zur k-ten seien frühestmöglich eingeplant. Die Einplanung der $k+1$-ten Grundaktion wird nur solange verzögert, bis die Scan-line alle Vorgänger von ihr überschritten hat und die von ihr benötigte Ressource verfügbar ist. Sobald die Ressource verfügbar ist und die Scan-line alle ihre Vorgänger überschritten hat, wird die $k+1$-te Grundaktion an die Scan-line angefügt. Somit wird die $k+1$-te Grundaktion frühestmöglich eingeplant. □

Bei der (rekursiven) $\mathcal{RSV}$-Diagrammberechnung werden Grundaktionen nacheinander gelöscht, so daß die aktuelle Eingabeaktion strukturell immer kleiner wird. Das folgende Lemma zeigt, daß ein mittels der $\mathcal{RSV}$-Diagrammberechnung erhaltener Ablaufplan für eine reduzierte Aktion auch unter Zuhilfenahme der Reihenfolge, in der die Grundaktionen im Verlauf der Diagrammberechnung gelöscht werden, dargestellt werden kann.

Lemma 4.5.6. *A sei eine beliebige reduzierte Aktion. $\mathcal{P}$ sei ein beliebiger zu A gehöriger Ablaufplan, der durch die $\mathcal{RSV}$-Diagrammberechnung erhalten sei. $[J(1), J(2), \cdots, J(n)]$ sei eine Permutation der in A vorkommenden Grundaktionen $\{1, \cdots, n\}$, entsprechend der Reihenfolge, in der sie bei der Berechnung von $\mathcal{P}$ gelöscht wurden. Bei $[(s(J(1)), J(1)), \cdots, (s(J(n)), J(n))]$ bezeichne $s(J(i))$ die Startzeit der Grundaktion $J(i) (i = 1, \cdots, n)$ bei $\mathcal{P}$. Wenn die Grundaktionen in der Reihenfolge $[(s(J(1)), J(1)), \cdots, (s(J(n)), J(n))]$ eingeplant werden, wird der Ablaufplan $\mathcal{P}$ erhalten, d.h. der Ablaufplan kann auch mittels der Reihenfolge $[(s(J(1)), J(1)), \cdots, (s(J(n)), J(n))]$ dargestellt werden.*

Beweis. Dieser Satz kann ähnlich wie Satz 4.1.1 bewiesen werden.

Notation 4.5.2. *Bei der $\mathcal{RSV}$-Diagrammberechnung eines Ablaufplans $\mathcal{P}$ seien die Grundaktionen $J(1), J(2), \cdots, J(n)$ in der Reihenfolge $[J(1), J(2), \cdots, J(n)]$*

aus der Eingabeaktion gelöscht worden. Die zu $\mathcal{P}$ gehörige Reihenfolge $[(s(J(1)), J(1)), \cdots, (s(J(n)), J(n))]$, *wobei* $s(J(i))$ *die Startzeit der Grundaktion* $J(i)(i = 1, \cdots, n)$ *bei* $\mathcal{P}$ *bezeichne, wird als* elim-*Reihenfolge bezeichnet.*

Wie in Abschnitt 4.1 gesehen, können mehrere *elim*-Reihenfolgen existieren, die denselben Ablaufplan darstellen. Im folgenden Satz wird bewiesen, daß im Rahmen der $\mathcal{RSV}$-Diagrammberechnung alle Ablaufpläne einer reduzierten Aktion berechnet werden und für jeden Ablaufplan *nur eine einzige elim-Reihenfolge* aufgezählt wird.

Lemma 4.5.7. *Für eine beliebige reduzierte Aktion A werden bei der $\mathcal{RSV}$-Diagrammberechnung alle nichtredundanten* elim-*Reihenfolgen aufgezählt.*

Beweis. Dies kann durch strukturelle Induktion über den Formelaufbau einer beliebigen reduzierten Aktion A gezeigt werden. Der Beweis verläuft ähnlich wie der Beweis von Lemma 4.5.2, und soll daher nur kurz skizziert werden. Beim Beweis des Induktionsschritts sind drei Fälle zu unterscheiden, entsprechend dem Vorkommen von Ressourcenkonflikten bei l-Zeit Scan-line Grundaktionen (siehe Lemma 4.5.2). In allen Fällen wird eine strukturell kleinere aktuelle Eingabeaktion erreicht und die Induktionsvoraussetzung angewendet. □

Aus den Lemmata 4.5.3, 4.5.4, 4.5.6 und 4.5.7 ergibt sich der folgende Satz:

Satz 4.5.1. *Für eine beliebige reduzierte Aktion A berechnet die $\mathcal{RSV}$-Diagrammberechnung alle nichtredundanten Ablaufpläne von A.*

Beweis. Jede *elim*-Reihenfolge stellt gemäß Lemma 4.5.6 einen Ablaufplan dar. Ferner ermittelt die $\mathcal{RSV}$-Diagrammberechnung gemäß Lemma 4.5.7 für jede reduzierte Aktion A alle nichtredundanten *elim*-Reihenfolgen. Somit berechnet die $\mathcal{RSV}$-Diagrammberechnung für eine beliebige Aktion A alle nichtredundanten Ablaufpläne. □

Satz 4.5.2. *Für eine beliebige Aktion A ermittelt die $\mathcal{RSV}$-Diagrammberechnung alle nichtredundanten optimalen Ablaufpläne sowie die optimale Gesamtbearbeitungszeit von A.*

Beweis. Nach Satz 4.5.1 berechnet die $\mathcal{RSV}$-Diagrammberechnung alle nichtredundanten Ablaufpläne von A. Nach der Ermittlung aller Ablaufpläne vergleicht die $\mathcal{RSV}$-Diagrammberechnung die erhaltenen Laufzeiten. In der Folge gibt die $\mathcal{RSV}$-Diagrammberechnung die optimalen Ablaufpläne mit optimaler Laufzeit zurück. □

4.6 Repräsentation der Ablaufpläne als Aktionen aus $\mathcal{RSVX}$

Wie in Abschnitt 4.3 gesehen, können für eine reduzierte Aktion mehrere Ablaufpläne existieren, die unterschiedliche Laufzeiten benötigen. Ablaufpläne wurden bis jetzt mittels eines $\mathcal{RSV}$-Diagramms bzw. eines Gantt-Charts dargestellt. In diesem Abschnitt wird vorgestellt, daß Ablaufpläne auch als *Aktionen aus* $\mathcal{RSVX}$ dargestellt werden können. Wir führen in diesem Zusammenhang folgende Notation ein:

Notation 4.6.1. *Aktive Ablaufpläne einer reduzierten Aktion werden* Prozeßaktionen *genannt.*

Dummy-Ressource, Dummy-Grundaktionen

Bei der Auflösung von Ressourcenkonflikten werden, wie zuvor gesehen, die zuerst durchzuführenden Konfliktgrundaktionen eingefroren und alle sonstigen Konfliktgrundaktionen nach rechts verschoben. Um bei der Repräsentation einer Prozeßaktion als eine Aktion aus $\mathcal{RSVX}$ diese Rechtsverschiebung zu berücksichtigen, wird eine ausgezeichnete Ressource benötigt, die nun eingeführt wird.

Definition 4.6.1. *eu bezeichnet eine Dummy-Ressource, die unbeschränkt verfügbar ist. Eine Grundaktion, die die Ressource eu beansprucht, heißt eine Dummy-Grundaktion. Beanspruchen mehrere Grundaktionen die Dummy-Ressource eu gleichzeitig, entsteht kein Ressourcenkonflikt.*

Die Dummy-Ressource eu unterscheidet sich von allen anderen Ressourcen, da sie unbeschränkt verfügbar ist. (Bisher wurde davon ausgegangen, daß von jeder Ressource jeweils nur eine Einheit während der gesamten Projektdauer zur Verfügung gestellt werden kann.) Wird bei der Auflösung von Ressourcenkonflikten eine Grundaktion hinter eine andere Grundaktion geschoben, kann das dadurch entstehende Zeitinterval (Pufferzeit) stets mit einer Dummy-Grundaktion belegt werden.

Zur Verdeutlichung der Bedeutung der Dummy-Ressource eu greifen wir an dieser Stelle Beispiel 4.3.1 nochmalig auf. Bei dem Diagramm $D1$ aus Abbildung 4.12 können die bei Auflösung der Ressourcenkonflikte entstehenden Zeitintervalle mittels der beiden Dummy-Grundaktionen $P_d(eu\,,3)$ und , $P_d(eu\,,2)$ belegt werden. Das entstehende Diagramm zeigt Abbildung 4.22.

Die Prozeßaktion

$$\begin{aligned}\textbf{pll}\ &(\textbf{seq}\ P_1(a,1),\ (\textbf{pll}\ P_2(b,1), P_3(d,2)),\ P_4(c,3)),\\ &(\textbf{seq}\ P_d(eu,3), P_5(d,2), P_6(a,1))\\ &(\textbf{seq}\ P_d(eu,2), P_7(b,3))\\ &(\textbf{seq}\ P_8(e,2), P_d(eu,3), P_9(d,2))\end{aligned}$$

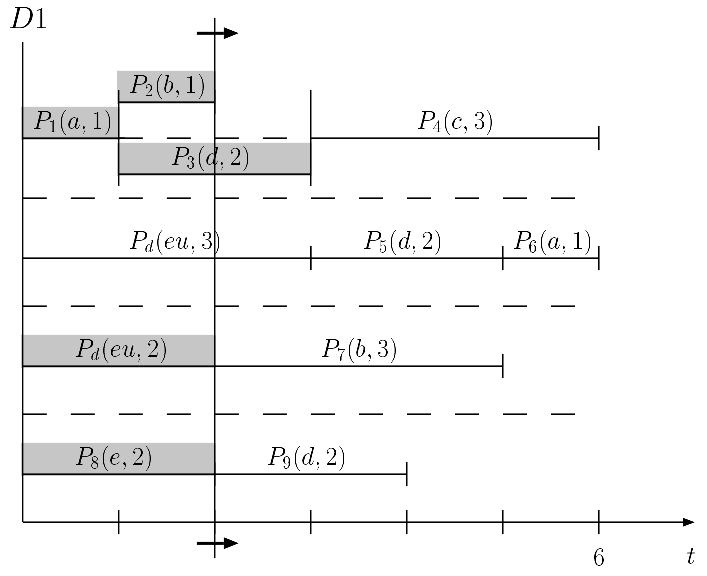


Abbildung 4.22: Belegung der bei Auflösung der Ressourcenkonflikte entstehenden Zeitintervalle mittels Dummy-Grundaktionen

gibt das Diagramm aus Abbildung 4.23 in Form einer reduzierten Aktion mit Dummy-Grundaktionen wieder. Es handelt sich damit bei obiger Prozeßaktion um einen Ausdruck aus $\mathcal{RSVX}$, bei dem Dummy-Grundaktionen vorkommen.

4.7 Gegenüberstellung der Lösungsansätze

In diesem Abschnitt wird die $\mathcal{RSV}$-Diagrammberechnung mit einem herkömmlichen Modell gegenüberstellt. Mittels dieser Gegenüberstellung können einige besonders vorteilhafte Merkmale der $\mathcal{RSV}$-Diagrammberechnung herausgestellt werden. Auf diese Merkmale wird am Schluß des Abschnitts nochmalig gesondert eingegangen.

4.7.1 Das Modell von Manne

Wie bereits in Kapitel 2 erwähnt, wurde 1960 von Allan S. Manne [Man60] ein Modell zur Darstellung und Lösung des allgemeinen Job Shop-Problems auf der Grundlage der linearen Programmierung vorgestellt. In diesem Abschnitt wird die Darstellung des Job Shop-Problems mittels des Modells von Manne vorstellt. Die in dieser Arbeit vorgestellte Version der Lösung des Job Shop-Problems mittels des Modells von Manne orientiert sich an Fürst [Für97].

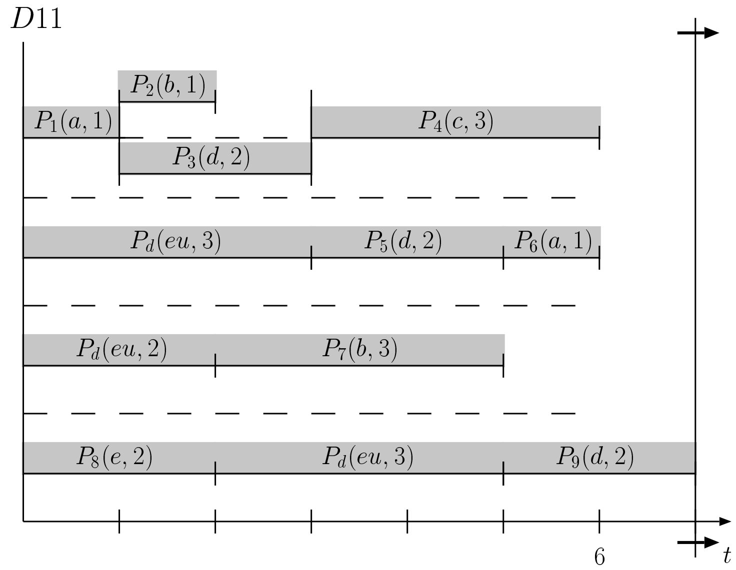


Abbildung 4.23: Eine Prozeßaktion mit der Gesamtbearbeitungszeit 7

Was ein Job Shop-Problem ist, wurde bereits in Abschnitt 2.1.1 ausgesagt. Als zusätzliche Annahme treffen wir in diesem Abschnitt, daß ein Auftrag auf einer Maschine maximal einmal zu bearbeiten ist.

Zur Darstellung eines Job Shop-Problems werden die folgenden Symbole eingeführt:

Indizes und Daten:

$I \quad = \quad \{i \mid i = 1, \cdots, n,\ i \text{ ist ein Auftrag}\}$

$J \quad = \quad \{j \mid j = 1, \cdots, m,\ j \text{ ist eine Maschine}\}$

$\mathit{\Pi} \quad = \quad \{\pi_i \mid \pi_i = (f_i(1), f_i(2), \cdots, f_i(i_l))$ ist Maschinenfolge des Auftrages i, i_l $(i_l \leq m)$ ist die letzte Bearbeitung (Arbeitsgang) des Auftrages i und $f_i(i_l)$ ist die zugehörige Maschine$\}$

d_{ij} gegebene Bearbeitungsdauer des Auftrages i auf Maschine j

M eine große Zahl

Variablen:

$D \quad = \quad$ Gesamtbearbeitungszeit

$h_{ij} \quad = \quad$ Bearbeitungsbeginn von Auftrag i auf der Maschine j, d.h. Bearbeitungsbeginn vom Arbeitsgang ij

$$u_{ikj} = \begin{cases} 1, & \text{wenn Auftrag } i \text{ vor Auftrag } k \text{ auf Maschine } j \text{ bearbeitet wird} \\ 0, & \text{sonst} \end{cases}$$

Die Zielfunktion lautet:

$$D = min!$$

Die zu erfüllenden Nebenbedingungen sind:

1. $h_{ij} + d_{ij} \leq h_{ij'}$ für alle $i \in I$; für alle $j = f_i(r)$ und $j' = f_i(r+1)$ mit $r = 1, \cdots, i_l - 1$
2. $h_{if_i(i_l)} + d_{if_i(i_l)} \leq D$ für alle $i \in I$
3. $h_{ij} + d_{ij} \leq h_{kj} + M \cdot (1 - u_{ikj})$ für alle $j \in J$ und $i \neq k$ mit $i, k \in \{e | \exists r$, so daß $f_e(r) = j$ in $\pi_e\}$
4. $h_{kj} + d_{kj} \leq h_{ij} + M \cdot u_{ikj}$ für alle $j \in J$ und $i \neq k$ mit $i, k \in \{e | \exists r$, so daß $f_e(r) = j$ in $\pi_e\}$
5. $h_{ij} \in \mathbb{N}_0$ für alle $i \in I$ und $j \in J$
6. $D \geq 0$
7. $u_{ikj} \in \{0, 1\}$ für alle $j \in J$ und $i \neq k$ mit $i, k \in \{e | \exists r$, so daß $f_e(r) = j$ in $\pi_e\}$

Durch die Restriktionen aus der ersten Nebenbedingung wird sicher gestellt, daß die vorgegebenen Maschinenfolgen der einzelnen Aufträge, d.h. die vorgegebene, sequentielle Bearbeitung der Arbeitsgänge für jeden Auftrag, eingehalten werden. Die zweite Nebenbedingung dient der Bestimmung der Gesamtbearbeitungszeit. Die dritte und vierte Nebenbedingung dienen der Festlegung der zu bestimmenden Auftragsreihenfolgen der Maschinen. Die fünfte bis siebte Nebenbedingung sind Ganzzahligkeits- bzw. Nichtnegativitätsbedingungen.

4.7.2 Gegenüberstellung der $\mathcal{RSV}$-Diagrammberechnung zu dem Modell von Manne anhand eines Beispiels

Wir wollen anhand eines einfachen Beispiels die zwei Ansätze gegenüberstellen. Zuerst wird ein Job Shop-Problem mittels des Modells Manne dargestellt und anschließend wird dasselbe Problem mittels der Sprache $\mathcal{RSV}$ formuliert.
Gegeben sei folgendes Job Shop-Problem:

4 Aufträge ($i = 1, 2, 3, 4$) seien auf 4 Maschinen ($j = 1, 2, 3, 4$) zu durchlaufen. Die Matrix, die die Bearbeitungsdauer jedes Auftrags i auf jeder Maschine j, bezeichnet durch d_{ij}, wiedergibt, enthält Tabelle 4.3.

Desweiteren seien die folgenden Maschinenfolgen π_i der einzelnen Aufträge i wie folgt vorgegeben:

		j 1	2	3	4
	1			4	2
i	2	3			3
	3		2	1	2
	4	4	1	2	

Tabelle 4.3: Die Bearbeitungsdauer d_{ij}

$\pi_1 = (4,3)$
$\pi_2 = (1,4)$
$\pi_3 = (2,3,4)$
$\pi_4 = (3,1,2)$

Das obige Beispielproblem wird nach dem Modell von Manne wie folgt formuliert: Die Zielfunktion, bei der die Gesamtbearbeitungszeit D minimiert wird, lautet:

$$D = min!$$

Hierbei sind die folgenden Nebenbedingungen zu beachten:

1. Einhaltung der vorgegebenen Maschinenfolgen der Aufträge:

   für Auftrag 1: $h_{14} + 2 \leq h_{13}$
   für Auftrag 2: $h_{21} + 3 \leq h_{24}$
   für Auftrag 3: $h_{32} + 2 \leq h_{33}$
   $h_{33} + 1 \leq h_{34}$
   für Auftrag 4: $h_{43} + 2 \leq h_{41}$
   $h_{41} + 4 \leq h_{42}$

2. Bestimmung der Gesamtdurchlaufzeit:

   für Auftrag 1: $h_{13} + 4 \leq D$
   für Auftrag 2: $h_{24} + 3 \leq D$
   für Auftrag 3: $h_{34} + 2 \leq D$
   für Auftrag 4: $h_{42} + 1 \leq D$

3. Entscheidung über die Reihenfolgen der einzelnen Aufträge:

$$
\begin{array}{ll}
\text{für Maschine 1:} & h_{21} + 3 \leq h_{41} + M(1 - u_{241}) \\
& h_{41} + 4 \leq h_{21} + M \cdot u_{241} \\
\text{für Maschine 2:} & h_{32} + 2 \leq h_{42} + M(1 - u_{342}) \\
& h_{42} + 1 \leq h_{32} + M \cdot u_{342} \\
\text{für Maschine 3:} & h_{13} + 4 \leq h_{33} + M(1 - u_{133}) \\
& h_{33} + 1 \leq h_{13} + M \cdot u_{133} \\
& h_{13} + 4 \leq h_{43} + M(1 - u_{143}) \\
& h_{43} + 2 \leq h_{13} + M \cdot u_{143} \\
& h_{33} + 1 \leq h_{43} + M(1 - u_{343}) \\
& h_{43} + 2 \leq h_{33} + M \cdot u_{343} \\
\text{für Maschine 4:} & h_{14} + 2 \leq h_{24} + M(1 - u_{124}) \\
& h_{24} + 3 \leq h_{14} + M \cdot u_{124} \\
& h_{14} + 2 \leq h_{34} + M(1 - u_{134}) \\
& h_{34} + 2 \leq h_{14} + M \cdot u_{134} \\
& h_{24} + 3 \leq h_{34} + M(1 - u_{234}) \\
& h_{34} + 2 \leq h_{24} + M \cdot u_{234}
\end{array}
$$

4. Nichtnegativitätsbedingung:

$$
\begin{array}{ll}
h_{ij} \in \mathbb{N}_0 & \text{für alle relevanten } i, j \\
D_i \geq 0 & \text{für } i = 1, \cdots, 4 \\
u_{ikj} \in \{0, 1\} & \text{für alle relevanten } i, k, j
\end{array}
$$

Bevor dieses Problem mittels der Sprache $\mathcal{RSV}$ formuliert wird, ist es erforderlich, die mit $j = 1, 2, 3, 4$ bezeichneten Maschinen in a, b, c, d umzubenennen, da bei bei der Sprache $\mathcal{RSV}$ Maschinen (Ressourcen) stets mittels Buchstaben bezeichnet werden. Aus Tabelle 4.3 kann abgelesen werden, daß die Aufträge 1 und 2 aus 2 Arbeitsgängen bestehen und die Aufträge 3 sowie 4 aus 3 Arbeitsgängen bestehen. Die in Tabelle 4.3 festgehaltenen Bearbeitungsdauer sowie die Maschinenfolgen können wie in Tabelle 4.4 gezeigt zusammengefaßt werden.

Im weiteren wird der j-te Arbeitsgang des i-ten Auftrags mit P_{ij} bezeichnet. Das betrachtete Beispielproblem kann mittels des folgenden Ausdruckes aus $\mathcal{RSV}$ repräsentiert werden:

Auftrag	Maschinenfolge Bearbeitungszeit
1	d c 2 4
2	a d 3 3
3	b c d 2 1 2
4	c a b 2 4 1

Tabelle 4.4: Beispiel zur Maschinenbelegungsplanung

$$\begin{array}{ll}\textbf{pll} & (\textbf{seq}\ P_{11}(d,2),\ P_{12}(c,4)),\\ & (\textbf{seq}\ P_{21}(a,3),\ P_{22}(d,3)),\\ & (\textbf{seq}\ P_{31}(b,2),\ P_{32}(c,1),\ P_{33}(d,2))\\ & (\textbf{seq}\ P_{41}(c,2),\ P_{42}(a,4),\ P_{43}(b,1))\end{array}$$

Obiger $\mathcal{RSV}$-Ausdruck kann dazu benutzt werden, um das als Beispiel betrachtete Job Shop-Problem mittels des in diesem Kapitel vorgestellten Algorithmus ($\mathcal{RSV}$-Diagrammberechnung) zu lösen, d.h. alle optimalen Ablaufpläne zu ermitteln. Abschließend soll eine stichpunktartige Gegenüberstellung des Modells von Manne und der $\mathcal{RSV}$-Diagrammberechnung erfolgen:

- Das $\mathcal{RSV}$-Modell ist mächtiger als das Modell von Manne, d.h. alle mittels des Modells von Manne lösbaren Probleme können auch mittels des $\mathcal{RSV}$-Modells gelöst werden, die Umkehrung gilt jedoch nicht.

- Häufig bewerten Wissenschaftler die auf der Grundlage der linearen Programmierung basierenden Modelle als Modelle, die eine sehr gute und elegante Formulierung des Job Shop-Problems geben[9] . Offensichtlich bietet das System $\mathcal{RSV}$ eine noch elegantere und leicht verständlichere Formulierung des Problems. Desweiteren kann von einem $\mathcal{RSV}$-Ausdruck, der ein Job Shop-Problem beschreibt, die Ablaufstruktur der Aufträge direkt und deutlich abgelesen werden.

- Wird ein Job Shop-Problem mittels der zuletzt vorgestellten Repräsentationsmethodik als ein $\mathcal{RSV}$-Ausdruck dargestellt, entfällt die aufwendige Repräsentation mittels zahlreicher Ungleichungen, wie sie beim Modell von Manne erforderlich ist.

- Die unschöne Eigenschaft, daß sich die Zahl der Variablen sowie Restriktionen (Ungleichungen) in Abhängigkeit der Problemgröße beim Modell von

Manne stark erhöht, wird bei der Repräsentation als $\mathcal{RSV}$-Ausdruck vermieden.

4.8 Abschließende Bemerkungen

Wie bereits erwähnt, kann ein $\mathcal{RSV}$-Ausdruck A durch Anwendung des in Abschnitt 3.2 definierten Kalküls in einen semantisch äquivalenten Ausdruck B transformiert werden, bei dem alle in dem Ausdruck A enthaltenen *nichtredundanten* Netzpläne (alle reduzierten Aktionen) durch einen einzelnen Operator '**xor**' verknüpft sind. *"nichtredundant"* heißt dabei, daß jeder enthaltene Netzplan bei der Transformation vom Ausdruck A zum Ausdruck B genau einmal erhalten wird. Ferner konnten für jeden solchen Netzplan mittels des in diesem Kapitel vorgestellten Algorithmus *alle herleitbaren nichtredundanten* Ablaufpläne berechnet werden. Somit wurde ein Berechnungsmodell erhalten, mit dem für ein beliebiges $\mathcal{RSV}$-Problem alle *nichtredundanten optimalen* Ablaufpläne berechnet werden können. Dies illustriert die Abbildung 4.24.

[9] vgl. [KS90]

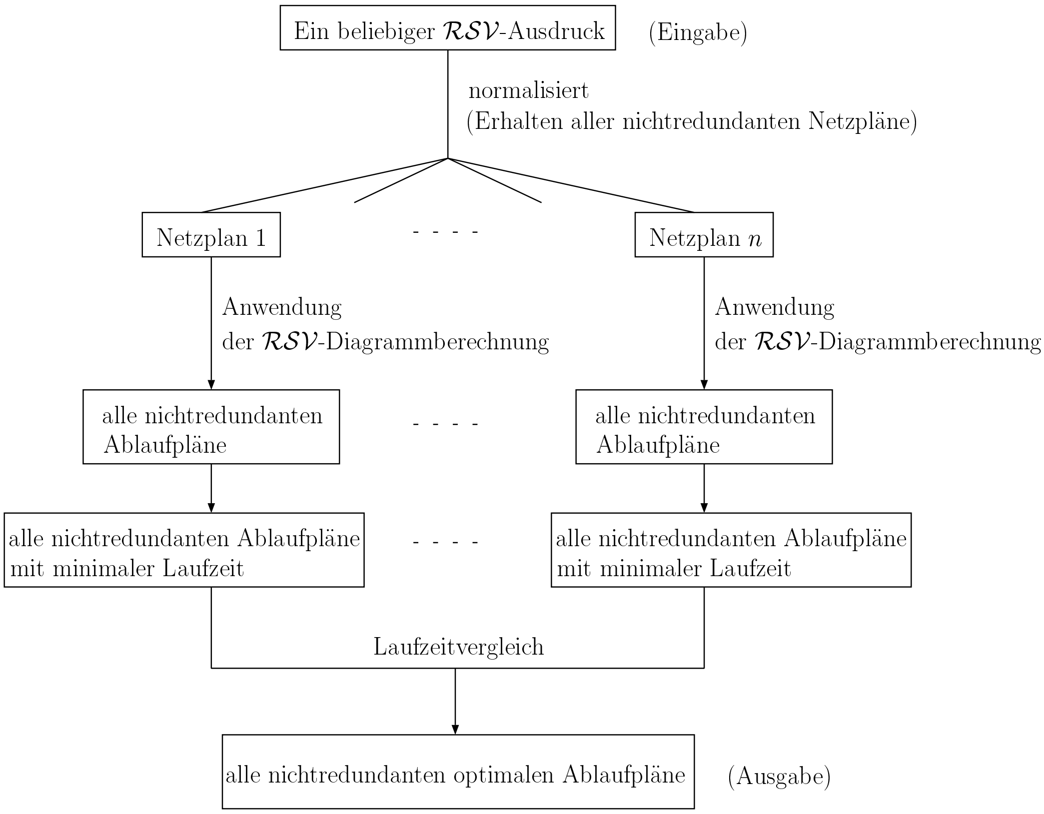


Abbildung 4.24: Berechnung aller nichtredundanten optimalen Ablaufpläne für ein $\mathcal{RSV}$-Problem

Kapitel 5

Weitere algebraische Eigenschaften

In diesem Kapitel werden weitere algebraische Eigenschaften der Sprache $\mathcal{RSV}$ vorgestellt. Der Operator '**seq**' spezifiziert wie bereits erwähnt die sequentielle Abarbeitung seiner Argumente (Aktionen). Als Folge davon können die Argumente des Operators '**seq**' bei der Berechnung aller optimalen Ablaufpläne einzeln betrachtet werden, d.h. für jedes Argument können alle optimalen Ablaufpläne getrennt berechnet und anschließend aneinander gefügt werden. In diesem Kapitel wird eine algebraische Betrachtung dieser Eigenschaft des Operators '**seq**' gegeben.

5.1 Die Funktionen $evalS$ und $evalSL$

Wie in Abschnitt 4.1 gesehen, stellt eine Abarbeitungsfolge von Grundaktionen für eine beliebige reduzierte Aktion eine zu der Aktion gehörige Prozeßaktion (einen aktiven Ablaufplan) dar. Die Funktion $evalS$ ordnet einer reduzierten Aktion die zugehörigen optimalen Abarbeitungsfolgen zu. Die Funktion $evalSL$ ordnet einer reduzierten Aktion die minimale Gesamtbearbeitungszeit zu. Im folgenden Abschnitt wird gezeigt, wie die Funktionswerte bei den Funktionen $evalS$ sowie $evalSL$ wie folgt berechnet werden können.

5.1.1 Berechnung der Funktion $evalS$

Die Funktionswerte der Funktion $evalS$ können wie folgt berechnet werden.

Berechnung für eine Grundaktion

Bei der Durchführung einer Grundaktion kann kein Ressourcenkonflikt entstehen. Die Menge der optimalen Abarbeitungsfolgen für eine Grundaktion besteht aus der einzigen Grundaktion. Somit gilt für eine Grundaktion $P(r,t)$ die folgende Gleichung:

$$evalS(P(r,t)) = \{P(r,t)\}$$

Berechnung für eine pll-Aktion

Wie in Abschnitt 4.5 gesehen, kann die Menge aller nichtredundanten optimalen Abarbeitungsfolgen für eine **pll**-Aktion unter Zuhilfenahme der $\mathcal{RSV}$-Diagrammberechnung ermittelt werden. Ein Beispiel hierfür befindet sich in Abschnitt 4.1. Bei diesem Beispiel gilt die folgende Gleichung:

$$\begin{aligned} evalS(&\mathbf{pll}\ (\mathbf{seq}\ P_1(a,1), P_2(b,2)), \\ &\quad P_3(a,2)) \\ &= \{[P_1(a,1), P_{2_{1<}}(b,2), P_3(a,2)]\} \end{aligned}$$

Berechnung für eine seq-Aktion

Der Operator '**seq**' spezifiziert die sequentielle Durchführung einer bzw. mehrerer Aktionen. Um den Funktionwert bei der Funktion *evalS* für eine **seq**-Aktion zu berechnen, existieren zwei alternative Möglichkeiten. Die Möglichkeiten sind eine Berechnung

- mittels Anwendung der rekursiven Eigenschaft des Operators '**seq**' oder
- mittels Anwendung der Assoziativitätsregel des Operators '**seq**'.

Anwendung der rekursiven Eigenschaft des Operators '**seq**':

Die rekursive Berechnungsvorschrift der Funktion *evalS* für den Operator '**seq**' wird wie folgt gegeben:

Gegeben sei eine **seq**-Aktion $(\mathbf{seq}\, A_1, A_2, \cdots, A_n)$, wobei A_1, A_2, $\cdots$, A_n beliebige Subaktionen (reduzierte Aktionen) seien. Die Funktion *evalS*, die zu einer als Eingabe gegebenen reduzierten Aktion alle zugehörigen optimalen Abarbeitungsfolgen ermittelt, kann wie folgt rekursiv für eine **seq**-Aktion angewendet werden:

$$evalS(\mathbf{seq}\, A_1, A_2, \cdots, A_n) = evalS(A_1) \times evalS(A_2) \times \cdots \times evalS(A_n)$$

Nach der rekursiven Anwendung der Funktion *evalS* auf eine **seq**-Aktion erhält *evalS* zuletzt als Argument nur eine Grundaktion oder eine **pll**-Aktion. Es wurde schon gezeigt, wie *evalS* für eine Grundaktion sowie für eine **pll**-Aktion berechnet werden kann.
Der Operator $\times$ beschreibt dabei das Kreuzprodukt zweier Mengen. Ferner werden alle Aktionen aus $evalS(A_{i-1})$, die bei $evalS(A_{i-1})$ zuletzt durchgeführt werden, als direkte Vorgänger aller Aktionen aus $evalS(A_i)$ angenommen, die bei $evalS(A_i)$ zuerst durchgeführt werden. Im folgenden wird dies mittels eines Beispiels verdeutlicht:

Beispiel 5.1.1. Die Berechnung des Wertes der Funktion *evalS* für die Beispielaktion

$$\begin{aligned}&\mathbf{seq}\, P_1(a,3),\\ &\quad(\mathbf{seq}\ P_2(b,2),\ (\mathbf{pll}\ P_3(a,1), P_4(a,2)))\end{aligned}$$

zeigt die folgende Entwicklung:

$$\begin{aligned}
&evalS(\mathbf{seq}\, P_1(a,3),\\
&\qquad(\mathbf{seq}\ P_2(b,2),\ (\mathbf{pll}\ P_3(a,1), P_4(a,2))))\\
&\quad= evalS(P_1(a,3))\\
&\qquad\times evalS(\mathbf{seq}\ P_2(b,2),(\mathbf{pll}\ P_3(a,1), P_4(a,2)))\\
&\quad= evalS(P_1(a,3))\\
&\qquad\times evalS(P_2(b,2)) \times evalS(\mathbf{pll} P_3(a,1), P_4(a,2))\\
&\quad= \{P_1(a,3)\}\\
&\qquad\times\{(P_2(b,2)\} \times \{[P_3(a,1), P_4(a,2)], [P_4(a,2), P_3(a,1)]\}\\
&\quad= \{[P_1(a,3), P_{2_{1<}}(b,2), P_{3_{2<}}(a,1), P_{4_{2<}}(a,2)],\\
&\qquad[P_1(a,3), P_{2_{1<}}(b,2), P_{4_{2<}}(a,2), P_{3_{2<}}(a,1)]\}
\end{aligned}$$

Wie bereits in Abschnitt 4.1 erläutert, besagt die Kennzeichnung $1 <$ bei der Grundaktion $P_{2_{1<}}(b,2)$, daß die Grundaktion P_2 den direkten Vorgänger P_1 besitzt.

Anwendung der Assoziativitätsregel des Operators '**seq**':

Wegen der Assoziativitätsregel des Operators '**seq**' (siehe Abschnitt 3.2.1) kann der Operator '**seq**', sofern er als Argument einer **seq**-Aktion vorkommt, aufgelöst werden. Im weiteren sei eine **seq**-Aktion A gegeben. Durch Anwendung der Assoziativitätsregel auf die Aktion A, kann A in einen semantisch äquivalenten Ausdruck A' transformiert werden, bei dem nur Grundaktionen oder **pll**-Aktionen als Argumente vorkommen. Nach dieser Transformation von A zu A' wird dann die Funktion *evalS* auf A' angewendet. Sei A' gleich $(\mathbf{seq}\ T_1, T_2, \cdots, T_n)$. Für die Eingabeaktion A kann die Funktion *evalS* wie folgt berechnet werden:

$$\begin{aligned}evalS(A) &= evalS(\mathbf{seq}\ T_1, T_2, \cdots, T_n)\\ &= evalS(T_1) \times evalS(T_2) \times \cdots \times evalS(T_n)\end{aligned}$$

5.1.2 Berechnung der Funktion $evalSL$

Die Funktion $evalSL$, die einer beliebigen reduzierten Aktion die optimale Gesamtbearbeitungszeit zuordnet, kann wie folgt berechnet werden.

Berechnung für eine Grundaktion

Jede Grundaktion besitzt eine Ressource und eine zeitliche Dauer. Die Ressource und zeitliche Dauer werden zur Erfüllung der Grundaktion benötigt. Die zu einer Grundaktion gehörige zeitliche Dauer entspricht eindeutig der minimalen Gesamtbearbeitungszeit der Grundaktion. Somit gilt für jede Grundaktion $P(r,t)$ die folgende Gleichung:

$$evalSL(P(r,t)) = t$$

Berechnung für eine pll-Aktion

In Kapitel 4 wurde ein Modell entwickelt, mit dessen Hilfe für eine **pll**-Aktion die minimale Gesamtbearbeitungszeit ermittelt werden kann. Unter Zuhilfenahme eines solchen Modells soll für eine **pll**-Aktion die minimale Gesamtbearbeitungszeit berechnet werden. Ein Beispiel hierfür befindet sich in Abschnitt 4.1. Bei diesem Beispiel gilt die folgende Gleichung:

$$\begin{gathered} evalSL(\mathbf{pll}(\mathbf{seq}\, P_1(a,1), P_2(b,2)), \\ P_3(a,2)) = 3 \end{gathered}$$

Berechnung für eine seq-Aktion

Genauso wie bei der Funktion *evalS* existieren zwei alternative Möglichkeiten, die Funktionswerte bei der Funktion $evalSL$ für eine **seq**-Aktion zu berechnen. Die Möglichkeiten sind eine Berechnung

- mittels Anwendung der rekursiven Eigenschaft des Operators '**seq**' oder
- mittels Anwendung der Assoziativitätsregel des Operators '**seq**'.

Anwendung der rekursiven Eigenschaft des Operators '**seq**':

Die rekursive Berechnungsvorschrift der Funktion $evalSL$ für den Operator '**seq**' wird wie folgt gegeben:

Gegeben sei eine **seq**-Aktion $(\mathbf{seq}\, A_1, A_2, \cdots, A_n)$, wobei A_1, A_2, $\cdots$, A_n beliebige Subaktionen (reduzierte Aktionen) seien. Die Funktion $evalSL$ kann wie folgt rekursiv auf die **seq**-Aktion angewendet werden:

$$evalSL(\mathbf{seq}\, A_1, A_2, \cdots, A_n)) = evalSL(A_1) + evalSL(A_2) + \cdots + evalSL(A_n)$$

Nach derartigen rekursiven Anwendungen auf eine **seq**-Aktion erhält $evalSL$ zuletzt als Argument nur eine Grundaktion oder eine **pll**-Aktion. In diesem Abschnitt wurde schon gezeigt, wie der Funktionswert bei der Funktion $evalSL$ für eine Grundaktion bzw. für eine **pll**-Aktion berechnet werden kann. Im folgenden wird die Berechnung der Funktion $evalSL$ für eine **seq**-Aktion anhand eines Beispiels verdeutlicht:

Beispiel 5.1.2. Die rekursive Berechnung der Funktion *evalSL* für die Beispielaktion

$$\begin{aligned}&\mathbf{seq}\,P_1(a,3),\\&\quad(\mathbf{seq}\ P_2(b,2),\ (\mathbf{pll}\ P_3(a,1),P_4(a,2)))\end{aligned}$$

zeigt die folgende Entwicklung.

$$\begin{aligned}&evalSL(\mathbf{seq}\,P_1(a,3),\\&\qquad(\mathbf{seq}\ P_2(b,2),\ (\mathbf{pll}\ P_3(a,1),P_4(a,2))))\\&\quad= evalSL(P_1(a,3))\\&\qquad+evalSL(\mathbf{seq}\ P_2(b,2),\ (\mathbf{pll}\ P_3(a,1),P_4(a,2)))\\&\quad= evalSL(P_1(a,3))\\&\qquad+evalSL(P_2(b,2))+evalSL(\mathbf{pll}\ P_3(a,1),P_4(a,2))\\&\quad=3+2+3\\&\quad=8\end{aligned}$$

Anwendung der Assoziativitätsregel des Operators '**seq**':

Nach der wiederholten Anwendung der Assoziativitätsregel des Operators '**seq**' auf eine **seq**-Aktion A kann A in einen semantisch äquivalenten Ausdruck A' transformiert werden, bei dem nur Grundaktionen oder **pll**-Aktionen als Argumente vorkommen. Nach dieser Transformation von A zu A' wird dann die Funktion *evalSL* auf A' angewendet. Sei $A\prime$ gleich $(\mathbf{seq}T_1,T_2,\cdots,T_n)$. Für die Eingabeaktion A kann die Funktion *evalSL* wie folgt berechnet werden:

$$\begin{aligned}evalSL(A)&= evalSL(\mathbf{seq}\ T_1,T_2,\cdots,T_n)\\&= evalSL(T_1)+evalSL(T_2)+\cdots+evalSL(T_n)\end{aligned}$$

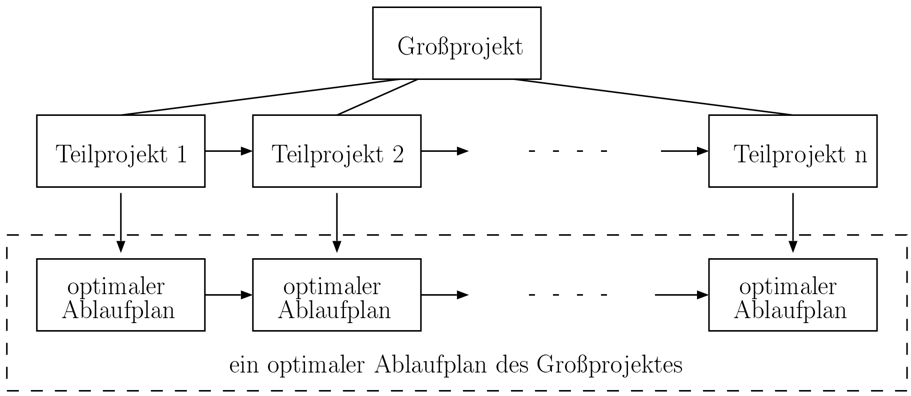
In diesem Abschnitt wurde bereits gezeigt, wie der Funktionwert bei der Funktion *evalSL* für eine Grundaktion bzw. eine **pll**-Aktion berechnet werden kann.

5.2 Abschließende Bemerkungen

Die in diesem Abschnitt vorgestellten Eigenschaften sind für große Projekte besonders interessant, die in mehrere sequentiell durchzuführende Teilprojekte zerlegt werden können. Ein solches Großprojekt kann dann mittels eines $\mathcal{RSV}$-Ausdruckes in der Form

$$\mathbf{seq}\,T_1,\cdots,T_n$$

modelliert werden, wobei $T_i (i = 1, \cdots, n)$ einer Grundaktion oder einer **pll**-Aktion entspricht. Die Subausdrücke $T_1, \cdots, T_n$ können zur Berechnung aller optimalen Ablaufpläne sowie der minimalen Gesamtbearbeitungszeit einzeln betrachtet werden. Dies illustriert die folgende Abbildung:



Kapitel 6

Die Schedulingsprache $\mathcal{RCPSV}$

In diesem Kapitel wird eine terminologische Sprache entwickelt, die wie zuvor beim $\mathcal{RSV}$-Problem *zur Darstellung und Lösung* des $\mathcal{RCPSV}$-Problems geeignet ist. Mittels der hier vorgestellten Syntax der Sprache $\mathcal{RCPSV}$ kann jedes $\mathcal{RCPSV}$-Problem als ein terminologischer Ausdruck dargestellt werden. Als Semantik eines Ausdruckes aus $\mathcal{RCPSV}$ werden wie beim $\mathcal{RSV}$-Problem alle aus dem Ausdruck herleitbaren Ablaufpläne angenommen. Auf Basis dieser Semantik wird ein Kalkül für die Sprache $\mathcal{RCPSV}$ vorgestellt, mit dem ein beliebiger Ausdruck aus $\mathcal{RCPSV}$ *in einen semantisch äquivalenten normalisierten Ausdruck* transformiert werden kann. Ein normalisierter Ausdruck besteht aus einer Menge von Ausdrücken, in denen der Operator '**xor**' nicht vorkommt, die alle durch einen einzelnen '**xor**'-Operator miteinander verknüpft sind. Die '**xor**'-freien Terme stellen dabei alle möglichen Netzpläne eines Projektes dar, die einzeln getrennt zur Bestimmung der optimalen Ablaufpläne herangezogen werden können. So ermöglicht die Existenz eines solchen Kalküls die Entwicklung eines Algorithmus, mittels dessen für einen beliebigen $\mathcal{RCPSV}$-Ausdruck alle optimalen Ablaufpläne berechnet werden können.

6.1 Die Schedulingsprache $\mathcal{RCPSV}$

Wie bereits in den Abschnitten 2.1.3 sowie 2.2.3 aufgeführt, kann ein beliebiges $\mathcal{RCPSV}$-Problem mittels eines hierarchischen Netzplans dargestellt werden. Desweiteren kann bei einem hierarchischen Netzplan für eine beliebige Menge von Knoten T, die durch AND-Kanten mit demselben Vater verbunden sind, eine Relation $E \subset T \times T$ erhalten werden, wobei E aus Präzedenzkanten von T besteht. Bei einem ein $\mathcal{RCPSV}$-Problem darstellenden hierarchischen Netzplan wird verlangt, daß jedes solcher Tupel (T, E) einen gerichteten schlichten azyklischen Graph repräsentiert.

Die nun folgenden Definitionen führen wir ein, um bei der terminologischen Repräsentation des $\mathcal{RCPSV}$-Problems die Formulierung oben beschriebener struktureller Eigenschaften zu ermöglichen.

Definition 6.1.1. *Ein* gerichteter Graph (T, E) *besteht aus einer Menge* $T = \{1, 2, \cdots, |T|\}$ *von* Knoten *und einer Menge* $E \subseteq T \times T$ *von* Kanten. *Ein Paar* $(t, t') \in E$ *heißt* Kante von t nach t'. *Weil* E *eine Menge ist, kann in einem (in dieser Arbeit betrachteten) Graph* (T, E) *jede Kante höchstens einmal auftreten (es werden keine parallelen Kanten erlaubt.) Eine Kante* (i, i) *wird Schlinge genannt.*

Ein gerichteter Graph (T, E) *ohne Schlingen wird als* ein schlichter gerichteter Graph *bezeichnet.*

Definition 6.1.2. T *sei eine nichtleere Knotenmenge.* $E \subset T \times T$ *sei eine Kantenmenge. Wenn* (T, E) *ein schlichter azyklischer gerichteter Graph ist, heißt* (T, E) *ein Schedulingnetzwerk auf* T.
Ein Element (t_i, t_j) *aus* E *beschreibt eine Präzedenzbeziehung zwischen* t_i *sowie* t_j. t_i *heißt ein direkter Vorgänger von* t_j.

Definition 6.1.3. (T, E) *sei ein Schedulingnetzwerk auf* T. *Ein Element* $t \in T$ *heißt ein isolierter Knoten von* (T, E), *wenn in* E *kein Element* (t_i, t_j) *existiert, so daß* $t = t_i$ *oder* $t = t_j$ *gilt.*

Wenn bei einem Schedulingnetzwerk (T, E) mit $E = \{(t_1, t_2), (t_3, t_4), \cdots, (t_{n-1}, t_n)\}$ kein isolierter Knoten existiert, kann (T, E) alleinig durch die Spezifikation von E beschrieben werden, da die Menge T aus E hergeleitet werden kann ($T = \{t_1, t_2\} \cup \{t_3, t_4\} \cup \cdots \cup \{t_{n-1}, t_n\}$). Falls ein isolierter Knoten existiert, kann die Herleitung durch das Hinzufügen eines dummy-Knotens 0 ermöglicht werden. Dazu wird der hinzugefügte dummy-Knoten 0 mit allen isolierten Knoten aus T verknüpft, d.h. für jeden isolierten Knoten t_i erfolgt die Bildung eines Tupels $(0, t_i)$, das als ein Element zur Menge E hinzugefügt wird (siehe Beispiel 6.1.1). Auf diese Weise erhalten wir eine Obermenge E' von E, von der aus die Knotenmenge T berechnet werden kann. Somit kann ein beliebiges Schedulingnetzwerk (T, E) auf T statt in der Form (T, E) nur mittels einer Menge von Präzedenzbeziehungen E bzw. E' beschrieben werden. Diese stets existente Vereinfachungsmöglichkeit der Schedulingnetzwerk-Beschreibung ist von elementarer Bedeutung im Kontext der Möglichkeit, ein $\mathcal{RCPSV}$ Schedulingproblem mittels der nun vorgestellten terminologischen Schedulingsprache zu spezifizieren.

Im folgenden führen wir eine terminologische Schedulingsprache ein, die wir genau wie das Problem $\mathcal{RCPSV}$ nennen werden.

6.1.1 Die Syntax der Schedulingsprache $\mathcal{RCPSV}$

Die Syntax der Sprache $\mathcal{RCPSV}$, mit der ein beliebiges Problem $\mathcal{RCPSV}$ modelliert werden kann, verfügt über folgende Definition:

Definition 6.1.4. *Das Vokabular von $\mathcal{RCPSV}$ besteht aus zwei disjunkten Mengen von Symbolen. Diese Mengen sind:*

- *Eine Menge von h-Grundaktionen* $\{(0, eu, 0)\} \cup \{(i, r(i), d(i)) | i = 1, \cdots, n, r(i) \in R, d(i) \in \mathbb{N}\}$, *wobei* $(0, eu, 0)$ *einer ausgezeichneten h-Grundaktion entspricht und R eine endliche Menge von Ressourcen ist. $eu \in R$ entspricht dabei der in Abschnitt 4.6 definierten Dummy-Ressource. h-Grundaktionen sind atomar.*
- *Eine Menge von strukturellen Symbolen. Die Menge aller strukturellen Symbole enthält die 2 Symbole '**xor**' und '**hnet**'. Diese Symbole heißen Operatoren.*
- *Die Klammersymbole* $(,), [,]$.

Die Beschreibung aller h-Aktionen (Ausdrücke) aus $\mathcal{RCPSV}$ wird wie folgt induktiv gegeben:

1. *Jede h-Grundaktion ist eine h-Aktion.*
2. *Seien $N_1, N_2, \cdots, N_k$ h-Aktionen, dann sind alle Ausdrücke*

$$(\mathbf{xor}\, N_1, N_2, \cdots, N_k)$$

und

$$\mathbf{hnet}[let\, t_1 = N_1, \cdots, t_k = N_k; (t_{11}, t_{12}), \cdots, (t_{j1}, t_{j2})]$$

h-Aktionen, wobei $t_1, t_2, \cdots, t_k$ unterschiedlichen Namen (Konstanten) und $[(t_{11}, t_{12}), \cdots, (t_{j1}, t_{j2})]$ einem Schedulingnetzwerk auf $\{t_1, t_2, \cdots, t_k\} = \{N_1, N_2, \cdots, N_k\}$ entspricht.

Die ausgezeichnete h-Grundaktion $(0, eu, 0)$ korrespondiert mit dem oben vorgestellten Dummy-Knoten und wird in Angleichung daran als h-Dummy-Grundaktion bezeichnet. Das unten aufgeführte Beispiel 6.1.1 verdeutlicht, wie und aus welchem Grund eine ausgezeichnete h-Dummy-Grundaktion $(0, eu, 0)$ verwendet wird.

Die Operatoren '**xor**' und '**hnet**' sind h-Aktionen-konstruierend auf h-Aktionen und haben die folgenden Bedeutungen:

- '**xor**': Dieser Operator dient zur Spezifikation der Abarbeitung einer beliebig *ausgewählten* h-Aktion aus einer Menge von h-Aktionen. So spezifiziert ($\mathbf{xor}\ N_1, N_2, \cdots, N_k$) aus Definition 6.1.4, daß genau eine der h-Aktionen $N_1, N_2 \cdots, N_k$ durchgeführt werden muß.

- '**hnet**': Dieser Operator dient zur Spezifikation der Anordnung von Aktivitäten, entsprechend vorgegebener Präzedenzbeziehungen. Die strukturelle Anordnung der Argumente (Aktivitäten) des Operators '**hnet**' muß stets ein Schedulingnetzwerk beschreiben (siehe auch Definition 6.1.2). So spezifiziert $[(t_{11}, t_{12}), \cdots, (t_{j1}, t_{j2})]$ in $\mathbf{hnet}[let\, t_1 = N_1, \cdots, t_k = N_k; (t_{11}, t_{12}), \cdots, (t_{j1}, t_{j2})]$ aus Definition 6.1.4 den gerichteten schlichten und azyklischen Graph, der aus der Menge der Aktivitäten $\{t_{11}, t_{12}\} \cup \{t_{21}, t_{22}\} \cup \cdots \cup \{t_{j1}, t_{j2}\}$ in Kombination mit den Präzedenzbeziehungen $[(t_{11}, t_{12}), \cdots, (t_{j1}, t_{j2})]$ gebildet wird.

Wie beim $\mathcal{RSV}$-Problem beinhaltet ein Ausdruck aus $\mathcal{RCPSV}$ implizit folgende semantische Eigenschaften:

- Jede h-Grundaktion ist mit einer bestimmten Ressource und einer bestimmten Zeit verbunden. Ressource und zeitliche Dauer werden zur Erfüllung der h-Grundaktion benötigt.

- Es wird vorausgesetzt, daß von jeder Ressource jeweils nur *eine* Einheit während der gesamten Projektdauer zur Verfügung gestellt werden kann und jede Ressource zu einem Zeitpunkt von maximal einer h-Grundaktion belegt werden kann. Sie wird erst nach Beendigung der h-Grundaktion wieder freigegeben. Diese Ressourcenbeschränkung kann Ressourcenkonflikte verursachen, die eine parallele Bearbeitung mehrerer h-Aktionen verhindern.

- Eine durchzuführende h-Grundaktion kann nicht unterbrochen werden.

Obige Ausführungen werden nun anhand eines Beispiels näher erläutert:

Beispiel 6.1.1. Gegeben sei die folgende h-Aktion aus $\mathcal{RCPSV}$:

$$\begin{aligned}(\mathbf{xor}\ \ &\mathbf{hnet}[let\, t_1 = (1, a, 2), t_2 = (\mathbf{xor}(2, b, 1), (3, c, 2)); (t_1, t_2)], \\ &\mathbf{hnet}[let\, k_1 = (0, eu, 0), k_2 = (4, d, 2), k_3 = (5, d, 2); (k_1, k_2), (k_1, k_3)])\end{aligned} \quad (1)$$

Obige h-Aktion kann mittels des hierarchischen Netzplans aus Abbildung 6.1 dargestellt werden. Die Beziehung zwischen dem hierarchischen Netzplan und der h-Aktion soll nun näher erläutert werden.
Zur Durchführung des Projektes P existieren zwei Handlungsalternativen P_1 und P_2. Die Alternative P_1 beschreibt der Ausdruck (h-Aktion)

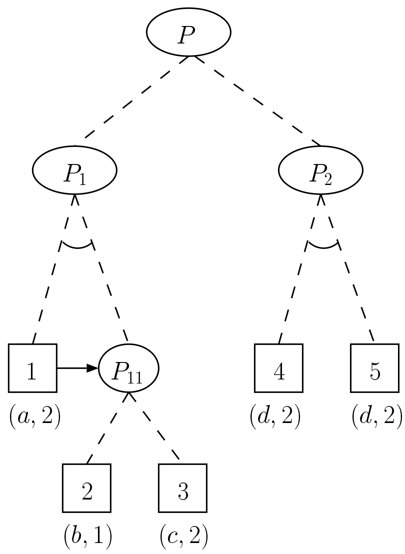


Abbildung 6.1: Ein hierarchischer Netzplan

$$\mathbf{hnet}[let\, t_1 = (1, a, 2), t_2 = (\mathbf{xor}(2, b, 1), (3, c, 2)); (t_1, t_2)]$$

Die Alternative P_2 beschreibt der Ausdruck

$$\mathbf{hnet}[let\, k_1 = (0, eu, 0), k_2 = (4, d, 2), k_3 = (5, d, 2); (k_1, k_2), (k_1, k_3)]$$

Bei der Durchführung von P_1 wiederum besteht eine Wahlmöglichkeit zwischen der Durchführung von Vorgang 2 oder 3 nach der Durchführung von Vorgang 1. Beim P_2 beschreibenden Ausdruck entspricht $(0, eu\,, 0)$ einer h-Dummy-Grundaktion. Die beiden Vorgänge (Grundaktivitäten) 4 und 5 müssen zur Erfüllung der Aktivität P_2 durchgeführt werden. Da zwischen diesen durch AND-Kanten mit demselben Vater verbundenen Knoten 4 und 5 keine Präzedenzbeziehung besteht, sind sie bei dem zu P_2 gehörigen Graph (Netzplan) isolierte Knoten (siehe Definition 6.1.3)[1]. Deshalb wird bei der terminologischen Darstellung die h-Dummy-Grundaktion verwendet.

Beispiel 6.1.2. Die h-Aktion

$$\begin{aligned}\mathbf{hnet}\quad & [t_1 = \mathbf{hnet}[let \cdots; (1, 3), (2, 3), (2, 4), (3, 4)],\\ & t_2 = (\mathbf{xor}\,\mathbf{hnet}[let \cdots; (5, 6)], \mathbf{hnet}[let \cdots; (7, 9), (8, 9)]); (t_1, t_2)]\end{aligned}$$

wobei jede Zahl $i \in \mathbb{N}$ eine h-Grundaktion repräsentiert (jeder h-Grundaktion i seien eine bestimmte Ressource $r(i)$ und eine bestimmte zeitliche Dauer $t(i)$ zugeordnet), kann mittels des hierarchischen Netzplans aus Abbildung 6.2 dargestellt

[1] Da die Aktionen 4 und 5 nicht durch eine Präzedenzkante verbunden sind, könnten sie prinzipiell parallel bearbeitet werden, was aber durch einen entstehenden Ressourcenkonflikt verhindert wird, da sie dieselbe Ressource d beanspruchen.

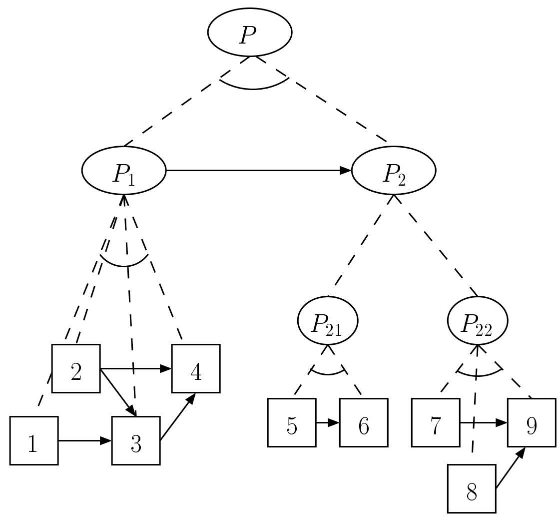


Abbildung 6.2: Ein hierarchischer Netzplan

werden. Obige h-Aktion bzw. der zugehörige hierarchische Netzplan beschreibt, daß zur Durchführung des Teilablaufs P_2, dessen terminologischer Ausdruck

$$(\mathbf{xor}\,\mathbf{hnet}[let \cdots ; (5,6)], \mathbf{hnet}[let \cdots ; (7,9), (8,9)])$$

entspricht, eine Wahlmöglichkeit zwischen $P_{21}(:= \mathbf{hnet}[let \cdots ; (5,6)])$ und $P_{22}(:= \mathbf{hnet}[let \cdots ; (7,9), (8,9)])$ besteht.

6.1.2 Die Semantik der Schedulingsprache $\mathcal{RCPSV}$

Wie in Abschnitt 3.1 erläutert, sind in einem beliebigen Ausdruck aus $\mathcal{RSV}$ eine bzw. mehrere reduzierten Aktionen zusammengefaßt. Als Semantik für einen Ausdruck aus $\mathcal{RSV}$ haben wir eine Menge angenommen, die aus allen aus dem Ausdruck herleitbaren Ablaufplänen besteht. Die Einführung dieser Semantik ermöglichte eine Entwicklung eines Lösungsverfahrens des $\mathcal{RSV}$-Schedulingproblems, wobei dies auf folgendem Sachverhalt beruhte:

Auf Basis der oben erwähnten Semantik kann eine beliebige $\mathcal{RSV}$-Aktion stets in eine semantisch äquivalente normalisierte Form, bei der alle zugehörigen alternativen reduzierten Aktionen durch einen einzelnen Operator '**xor**' verknüpft sind, transformiert werden. In Folge dessen genügte es einen Algorithmus zu entwickeln, der für eine reduzierte Aktion Ablaufpläne mit minimaler Gesamtbearbeitungszeit ermittelt. Für jede reduzierte Aktion können dann durch Anwendung eines solchen Algorithmus Ablaufpläne mit minimaler Gesamtbearbeitungszeit ermittelt werden. Diejenigen Ablaufpläne unter allen diesen ermittelten Ablaufplänen, die den kleinsten Wert (Gesamtbearbeitungszeit) aufweisen, entsprechen offensichtlich den gesuchten optimalen Ablaufplänen für eine $\mathcal{RSV}$-Aktion.

Diese Methode übernehmen wir zur Lösung des $\mathcal{RCPSV}$-Problems. Entsprechend definieren wir daher im folgenden eine Semantik für die Sprache $\mathcal{RCPSV}$. Wir betrachten dazu zunächst ein Beispiel in Form der h-Aktion

$$(\textbf{hnet}\ [let\, t_1 = (1, a, 3), t_2 = (2, b, 6), t_3 = (\textbf{xor}(3, c, 3), (4, d, 5)); (t_1, t_2), (t_2, t_3)]) \tag{2}$$

In obiger h-Aktion sind die beiden alternativen Ausdrücke

$$\begin{array}{l} \textbf{hnet}\ [let\, t_1 = (1, a, 3), t_2 = (2, b, 6), t_3 = (3, c, 3); (t_1, t_2), (t_2, t_3)] \text{ und} \\ \textbf{hnet}\ [let\, t_1 = (1, a, 3), t_2 = (2, b, 6), t_3 = (4, d, 5); (t_1, t_2), (t_2, t_3)] \end{array} \tag{3}$$

beschrieben, wobei jeder Ausdruck einen möglichen Ablauf bzw. einen zugehörigen Netzplan des untersuchten Projektes darstellt. Die h-Aktion, bei der diese beiden alternativen Ausdrücke durch den Operator **xor** verknüpft sind, sieht wie folgt aus:

$$\begin{array}{ll} (\textbf{xor} & \textbf{hnet}\ [let\, t_1 = (1, a, 3), t_2 = (2, b, 6), t_3 = (3, c, 3); (t_1, t_2), (t_2, t_3)], \\ & \textbf{hnet}\ [let\, t_1 = (1, a, 3), t_2 = (2, b, 6), t_3 = (4, d, 5); (t_1, t_2), (t_2, t_3)]) \end{array} \tag{4}$$

Als Semantik einer h-Aktion aus $\mathcal{RCPSV}$ nehmen wir die Menge an, die aus allen aus der h-Aktion herleitbaren Ablaufplänen besteht. Somit sind die beiden h-Aktionen (2) und (4) semantisch äquivalent, da die Menge aller aus der h-Aktion (2) herleitbaren Ablaufpläne sowie die Menge aller aus der h-Aktion (4) herleitbaren Ablaufpläne offensichtlich identisch sind (siehe unten stehende Beispiele 6.1.3 und 6.1.4).

Aus der semantischen Äquivalenz zweier h-Aktionen folgt, daß ein Ablaufplan, der für eine h-Aktion zeitlich optimal ist, auch für die andere semantisch äquivalente h-Aktion optimal ist und umgekehrt. Wir können daher generell zunächst eine h-Aktion in die *normalisierte* Form transformieren, bevor wir beginnen, optimale Ablaufpläne zu berechnen. Bei den normalisierten Ausdrücken können wir die Argumente (Netzpläne) des Operators '**xor**' einzeln getrennt auswerten und durch anschließenden Vergleich alle optimalen Ablaufpläne bestimmen. So ermöglicht die Einführung einer Semantik die Entwicklung eines Verfahrens zur Lösung des $\mathcal{RCPSV}$-Schedulingproblems. In Kapitel 7 werden wir einen entsprechenden Lösungsalgorithmus vorstellen.

Die Semantik der Sprache $\mathcal{RCPSV}$ wird nun definiert.

Definition 6.1.5. *Die modell-theoretische Semantik aller Aktionen aus* $\mathcal{RCPSV}$ *wird durch eine Interpretation* $\mathcal{I}$ *gegeben, die aus der Menge* $\mathcal{D}$ *(die Domäne von* $\mathcal{I}$*) und einer Funktion* $\cdot^{\mathcal{I}}$ *(die Interpretationsfunktion von* $\mathcal{I}$*) besteht. Die Menge*

$\mathcal{D}$ besteht aus allen Ablaufplänen, die aus $\mathcal{RCPSV}$-Aktionen hergeleitet werden können. Die Interpretationsfunktion $\cdot^{\mathcal{I}}$ bildet jede h-Aktion N aus $\mathcal{RCPSV}$ auf eine Teilmenge von $\mathcal{D}$ ab, die aus allen zu N gehörigen (aus N herleitbaren) Ablaufplänen besteht.

Die folgenden Beispiele sollen zum besseren Verständnis der Definition der Semantik beitragen. Ein Ablaufplan wird hierbei mittels einer Liste von Tripeln $((s(i), i), r(i), d(i))$ dargestellt, wobei $s(i)$ der Startzeit einer h-Grundaktion i, $r(i)$ der von i benötigten Ressource und $d(i)$ der von i benötigten zeitlichen Dauer entspricht. Alle hierbei vorkommenden h-Grundaktionen seien zu $\mathcal{RCPSV}$ gehörig:

Beispiel 6.1.3. Bei der h-Aktion

$$(\mathbf{hnet}\,[\mathit{let}\, t_1 = (1, a, 3), t_2 = (2, b, 6), t_3 = (\mathbf{xor}(3, c, 3), (4, d, 5)); (t_1, t_2), (t_2, t_3)])$$

ist eine der beiden h-Aktionen $(3, c, 3)$ und $(4, d, 5)$ auszuwählen. Durch obige h-Aktion werden zwei alternative Netzpläne beschrieben, bei denen kein Ressourcenkonflikt vorkommt. Deshalb können aus obiger h-Aktion zwei Ablaufpläne hergeleitet werden. Dies gibt die unten stehende Gleichung wieder, die die Anwendung der Interpretationsfunktion $\cdot^{\mathcal{I}}$ auf die h-Aktion unseres Beispiels zeigt:

$$\begin{aligned}&(\mathbf{hnet}\,[\mathit{let}\, t_1 = (1, a, 3), t_2 = (2, b, 6), t_3 = (\mathbf{xor}(3, c, 3), (4, d, 5)); (t_1, t_2), (t_2, t_3)])^{\mathcal{I}}\\ &= \{[((0, 1), a, 3), ((3, 2), b, 6), ((9, 3), c, 3)], [((0, 1), a, 3), ((3, 2), b, 6), ((9, 4), d, 5)]\}\end{aligned}$$

Der folgenden Gleichung ist zu entnehmen, daß dieselben Ablaufpläne aus dem normalisierten Äquivalent obiger h-Aktion hergeleitet werden können:

$$\begin{aligned}&(\mathbf{xor}\,\mathbf{hnet}\,[\mathit{let}\, t_1 = (1, a, 3), t_2 = (2, b, 6), t_3 = (3, c, 3); (t_1, t_2), (t_2, t_3)],\\ &\qquad \mathbf{hnet}\,[\mathit{let}\, t_1 = (1, a, 3), t_2 = (2, b, 6), t_3 = (4, d, 5); (t_1, t_2), (t_2, t_3)])^{\mathcal{I}}\\ &\quad = \{[((0, 1), a, 3), ((3, 2), b, 6), ((9, 3), c, 3)], [((0, 1), a, 3), ((3, 2), b, 6), ((9, 4), d, 5)]\}\end{aligned}$$

Beispiel 6.1.4. Der hierarchische Netzplan aus Abbildung 6.3 zeigt das grafische Aussehen der h-Aktion

$$(\mathbf{hnet}\,[\mathit{let}\, t_1 = (1, a, 2), t_2 = (\mathbf{xor}(2, b, 5), (3, c, 7)), t_3 = (4, b, 3); (t_1, t_2), (t_1, t_3)])$$

Bei der Auswahl der h-Grundaktion 2 entsteht ein Ressourcenkonflikt, da die beiden h-Grundaktionen 4 und 2 dieselbe Ressource b benötigen. So entstehen hierbei zwei alternative Ablaufpläne je nachdem, in welcher Reihenfolge 4 und 2 angeordnet werden. Bei dem Ablaufplan, bei dem die h-Grundaktion 3 ausgewählt wird, starten die beiden h-Grundaktionen 4 und 3 zu demselben Zeitpunkt 2, da die beiden ressourcenkonfliktfrei bearbeitet werden können. Alle drei aus der obigen h-Aktion herleitbaren Ablaufpläne zeigt die folgende Gleichung:

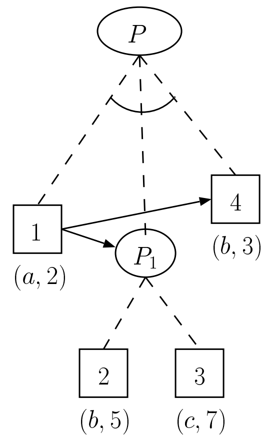


Abbildung 6.3: Ein hierarchischer Netzplan

$$\begin{aligned}
&(\mathbf{hnet} \quad [let\, t_1 = (1, a, 2), t_2 = (\mathbf{xor}(2, b, 5), (3, c, 7)), t_3 = (4, b, 3); (t_1, t_2), (t_1, t_3)])^{\mathcal{I}} \\
&= \{ \quad [((0, 1), a, 2), ((2, 2), b, 5), ((7, 4), b, 3)], \\
&\qquad [((0, 1), a, 2), ((2, 4), b, 3), ((5, 2), b, 5)], [((0, 1), a, 2), ((2, 3), c, 7), ((2, 4), b, 3)]\}
\end{aligned}$$

6.1.3 Schedulingaxiome

Ähnlich der Sprache $\mathcal{RSV}$ haben wir als Semantik für einen Ausdruck aus $\mathcal{RCPSV}$ eine Menge angenommen, die aus allen aus einem Ausdruck herleitbaren Ablaufplänen besteht. So gibt es wie bei $\mathcal{RSV}$ h-Aktionen aus $\mathcal{RCPSV}$, die zwar syntaktisch unterschiedlich aussehen, aber semantisch äquivalent sind. Um die semantische Äquivalenz zwischen h-Aktionen darstellen zu können, wird außer den Operatorsymbolen '**xor**' und '**hnet**' im Kontext der Sprache $\mathcal{RCPSV}$ ein weiteres strukturelles Symbol $\doteq$ definiert.
Der Ausdruck $C \doteq D$ beschreibt, daß die Interpretation der beiden h-Aktionen C und D identisch ist. D. h. die Menge der Ablaufpläne, die aus C ableitbar sind, und die Menge der Ablaufpläne, die aus D ableitbar sind, sind identisch.

Notation 6.1.1. *Die Äquivalenzbeziehungen (durch $\doteq$ hergestellte Beziehungen zwischen h-Aktionen) werden h-Schedulingaxiome genannt.*

Die Syntax von h-Schedulingaxiomen wird wie folgt formal definiert:

Definition 6.1.6. *Seien C und D h-Aktionen. Die Syntax der h-Schedulingaxiome wird durch*

$$C \doteq D$$

definiert.

Die Semantik der h-Schedulingaxiome wird wie folgt definiert:

Definition 6.1.7. *Seien C und D h-Aktionen. Eine Interpretation $\mathcal{I}$ von $\mathcal{RCPSV}$ erfüllt das h-Schedulingaxiom*

$$C \doteq D \qquad gdw. \qquad C^{\mathcal{I}} = D^{\mathcal{I}}$$

6.2 Der $\mathcal{RCPSV}$-Kalkül

In diesem Abschnitt wird ein Kalkül vorgestellt, der auf der im letzten Abschnitt definierten Semantik der Sprache $\mathcal{RCPSV}$ basiert. Mittels des Kalküls kann jeder beliebige Ausdruck aus $\mathcal{RCPSV}$ in einen semantisch äquivalenten Ausdruck transformiert werden, bei dem '**xor**'-freie Ausdrücke durch einen einzelnen an der äußersten linken Position vorkommenden '**xor**'-Operator verknüpft sind. Alle '**xor**'-freien Ausdrücke stellen hierbei Netzpläne dar, die unterschiedlich aussehen. Dabei beschreibt jeder einzelne Netzplan eine Ablaufstruktur zur Realisierung des untersuchten Projektes. Der Kalkül, den wir im folgenden $\mathcal{RCPSV}$-Kalkül nennen, wird wie folgt definiert:

Definition 6.2.1. *$N_1, N_2, \cdots, N_k, M_1, \cdots, M_l, N_{k+2}, \cdots, N_n$ seien beliebige h-Aktionen. Der Kalkül besitzt folgende 2 Ableitungsregeln:*

1. *Assoziativität des Operators '**xor**': Diese Regel beschreibt, daß ein mit '**xor**' verbundener Subausdruck, der als ein Argument des Operators '**xor**' vorkommt, aufgelöst werden kann.*

$$\frac{(\mathbf{xor}\ N_1, N_2, \cdots, N_k, (\mathbf{xor}\ M_1, M_2, \cdots, M_l), N_{k+2}, N_{k+3}, \cdots, N_n)}{(\mathbf{xor}\ N_1, N_2, \cdots, N_k, M_1, M_2, \cdots, M_l, N_{k+2}, N_{k+3}, \cdots, N_n)}$$

2. *Die folgenden beiden Regeln beschreiben, daß der Operator '**xor**' an die äußerste linke Position bewegt werden kann, wenn bei einer mit '**hnet**' verbundenen h-Aktion ein mit '**xor**' verknüpfter Subausdruck vorkommt:*

$$\frac{(\textbf{hnet}[let\, t_1 = N_1, \cdots, t_{k+1} = (\textbf{xor} M_1, \cdots, M_l), \cdots, t_n = N_n; (t_{11}, t_{12}), \cdots, (t_{h1}, t_{k+1}), \cdots, (t_{j1}, t_{j2})])}{\begin{array}{r}(\textbf{xor}\ \textbf{hnet}[let\, t_1 = N_1, \cdots, t_{k+1} = M_1, \cdots, t_n = N_n; (t_{11}, t_{12}), \cdots, \\ (t_{h1}, t_{k+1}), \cdots, (t_{j1}, t_{j2})], \\ \textbf{hnet}[let\, t_1 = N_1, \cdots, t_{k+1} = M_2, \cdots, t_n = N_n; (t_{11}, t_{12}), \cdots, \\ (t_{h1}, t_{k+1}), \cdots, (t_{j1}, t_{j2})], \\ \vdots \\ \textbf{hnet}[let\, t_1 = N_1, \cdots, t_{k+1} = M_l, \cdots, t_n = N_n; (t_{11}, t_{12}), \cdots, \\ (t_{h1}, t_{k+1}), \cdots, (t_{j1}, t_{j2})])\end{array}}$$

$$\frac{(\textbf{hnet}[let\, t_1 = N_1, \cdots, t_{k+1} = (\textbf{xor} M_1, \cdots, M_l), \cdots, t_n = N_n; (t_{11}, t_{12}), \cdots, (t_{k+1}, t_{h2}), \cdots, (t_{j1}, t_{j2})])}{\begin{array}{r}(\textbf{xor}\ \textbf{hnet}[let\, t_1 = N_1, \cdots, t_{k+1} = M_1, \cdots, t_n = N_n; (t_{11}, t_{12}), \cdots, \\ (t_{k+1}, t_{h2}), \cdots, (t_{j1}, t_{j2})], \\ \textbf{hnet}[let\, t_1 = N_1, \cdots, t_{k+1} = M_2, \cdots, t_n = t_n; (t_{11}, t_{12}), \cdots, \\ (t_{k+1}, t_{h2}), \cdots, (t_{j1}, t_{j2})], \\ \vdots \\ \textbf{hnet}[let\, t_1 = N_1, \cdots, t_{k+1} = M_l, \cdots, t_n = N_n; (t_{11}, t_{12}), \cdots, \\ (t_{k+1}, t_{h2}), \cdots, (t_{j1}, t_{j2})],\end{array}}$$

Wir werden jetzt zeigen, daß jede der zuletzt definierten Regeln die Interpretation eines $\mathcal{RCPSV}$-Ausdruckes nicht verändert, d. h. der Kalkül bzgl. der Semantik korrekt ist.

Satz 6.2.1. *Der Termumformungskalkül der Sprache $\mathcal{RCPSV}$ ist ein korrekter Kalkül, d.h. er läßt nur solche Termumformungen zu, die die semantische Äquivalenz erhalten.*

Beweis. Eine Regel der Form

$$\frac{N}{M}$$

erhält die semantische Äquivalenz, genau dann wenn die Interpretation des oberen Terms N und die Interpretation des unteren Terms M identisch sind ($N^{\mathcal{I}} = M^{\mathcal{I}}$). Im folgenden wird für die obigen 2 Regeln gezeigt, daß sie semantikerhaltend sind, d. h. die Interpretation des oberen und unteren Terms übereinstimmt:

Regel 1: Die Menge aller aus dem oberen und unteren Ausdruck herleitbaren Ablaufpläne ist offensichtlich identisch, da die beiden Ausdrücke dieselbe Auswahlmöglichkeit beschreiben. Eine der h-Aktionen $N_1, \cdots, N_k, M_1, \cdots, M_l, N_{k+2}, \cdots,$ N_{n-1}, N_n ist dabei auszuwählen.
$S_{N_i}(S_{M_j})$ sei die Menge aller aus $N_i(M_j)$ herleitbaren Ablaufpläne. Die Interpretation des oberen sowie unteren Ausdruckes entspricht der Vereinigung der Mengen $S_{N_1} \cdots S_{N_k}, S_{M_1} \cdots S_{M_l}, S_{N_{k+2}}, \cdots S_{N_{n-1}}, S_{N_n}$. Somit gilt die folgende Gleichung:

$$\begin{aligned}
&(\textbf{xor}\ N_1, N_2, \cdots, N_k, (\textbf{xor}\ M_1, M_2, \cdots, M_l), N_{k+2}, N_{k+3}, \cdots, N_n)^{\mathcal{I}} \\
&\quad = \quad (\textbf{xor}\ N_1, N_2, \cdots, N_k, M_1, M_2, \cdots, M_l, N_{k+2}, N_{k+3}, \cdots, N_n)^{\mathcal{I}}
\end{aligned}$$

Regel 2: Bei der zweiten Komponente des k-ten Arguments des Operators '**hnet**' besteht eine Auswahlmöglichkeit dahingehend, daß eine der l h-Aktionen $M_1, M_2,$ $\cdots, M_l$ auszuwählen ist. Für jede Auswahl von $M_i\ (i = 1, \cdots, l)$ kann aus dem Ausdruck

$$\begin{aligned}
&(\textbf{hnet}[let\, t_1 = N_1, \cdots, t_{k+1} = M_i, \cdots, t_n = N_n; (t_{11}, t_{12}), \cdots, (t_{h1}, t_{k+1}), \\
&\qquad \cdots, (t_{j1}, t_{j2})])
\end{aligned}$$

eine eindeutig bestimmte Menge von Ablaufplänen hergeleitet werden. Im weiteren werde diese Menge mit S_{M_i}bezeichnet. Die Menge aller aus dem oberen Ausdruck herleitbaren Ablaufpläne entspricht offensichtlich der Vereinigung $\cup_{i=1}^{l} S_{M_i}$. Dieselbe Menge wird erhalten, wenn beim unteren Ausdruck alle herleitbaren Ablaufpläne bestimmt werden.

Dieselbe Argumentation kann bei der zweiten Regel angewendet werden. □

Aus der Korrektheit des Kalküls ergibt sich der folgende Satz.

Satz 6.2.2. *Für jeden beliebigen Ausdruck (h-Aktion) N aus der Sprache $\mathcal{RCPSV}$ kann der Operator '**xor**' immer an die äußerste linke Position bewegt werden, sofern der Operator '**xor**' im Inneren der h-Aktion N vorkommt.*

Beweis. Dieser Satz kann ähnlich wie Satz 3.2.1 bewiesen werden. Auf den '**xor**'-Operator, der bei N *an der am weitesten rechten Stelle* vorkommt, wird wiederholt eine der Regeln 1 oder 2 angewendet. Auf diese Weise wird der '**xor**'-Operator schrittweise nach links bewegt. □

Satz 6.2.2 besagt, daß ein beliebiger $\mathcal{RCPSV}$-Ausdruck in einen semantisch äquivalenten, normalisierten Ausdruck transformiert werden kann. In einem normalisierten Ausdruck sind eine Menge '**xor**'-freier Ausdrücke, die alle ableitbaren Netzpläne darstellen, durch den Operator '**xor**' verbunden. Die folgenden Beispiele sollen zum besseren Verständnis des Kalküls beitragen. Hierbei werden die in Abschnitt 6.1.1 vorgestellten Beispiele fortsetzend betrachtet.

Beispiel 6.2.1. Die h-Aktion

$$\begin{array}{l}(\textbf{xor}\ \ \textbf{hnet}[let\, t_1 = (1, a, 2), t_2 = (\textbf{xor}(2, b, 1), (3, c, 2)); (t_1, t_2)], \\ \qquad \textbf{hnet}[let\, k_1 = (0, eu, 0), k_2 = (4, d, 2), k_3 = (5, d, 2); (k_1, k_2), (k_1, k_3)])\end{array} \tag{5}$$

aus Beispiel 6.1.1 kann durch Anwendung des Kalküls in den folgenden semantisch äquivalenten, normalisierten Ausdruck transformiert werden:

$$\begin{array}{l}(\textbf{xor}\ \ (\textbf{hnet}\ [let\, t_1 = (1, a, 2), t_2 = (2, b, 1); (t_1, t_2)]), \\ \qquad (\textbf{hnet}\ [let\, t_1 = (1, a, 2), t_2 = (3, c, 2); (t_1, t_2)]), \\ \qquad (\textbf{hnet}\ [let\, t_1 = (s, eu, 0), t_2 = (4, d, 2), t_3 = (5, d, 2); (t_1, t_2),\ (t_1, t_3)]))\end{array} \tag{6}$$

Die ableitbaren Netzpläne, die diese 3 '**xor**'-freie Ausdrücke repräsentieren, befinden sich in Abbildung 6.4, wobei das Symbol 'S' Start und das Symbol 'Z' Ziel bedeutet.

Jeder Netzplan stellt einen möglichen Ablauf zur Realisierung des untersuchten Projektes dar.

Beispiel 6.2.2. Die h-Aktion

$$\textbf{hnet}[(\textbf{hnet}[(1, 3), (2, 3), (2, 4), (3, 4)], (\textbf{xor}\,\textbf{hnet}[(5, 6)], \textbf{hnet}[(7, 9), (8, 9)]))]$$

aus Beispiel 6.1.2, wobei jede Zahl $i \in \mathbb{N}$ eine h-Grundaktion repräsentiert, kann durch Anwendung des Kalküls in den folgenden semantisch äquivalenten, normalisierten Ausdruck transformiert werden:

$$\begin{array}{l}(\textbf{xor}\ \ (\textbf{hnet}\ [(\textbf{hnet}\ [(1, 3), (2, 3), (2, 4), (3, 4)],\ \textbf{hnet}[(5, 6)])]), \\ \qquad (\textbf{hnet}\ [(\textbf{hnet}\ [(1, 3), (2, 3), (2, 4), (3, 4)],\ \textbf{hnet}[(7, 9), (8, 9)])]))\end{array}$$

Die abgeleiteten Netzpläne, die die beiden '**xor**'-freien Terme bei dem obigen normalisierten Ausdruck darstellen, zeigt Abbildung 6.5.

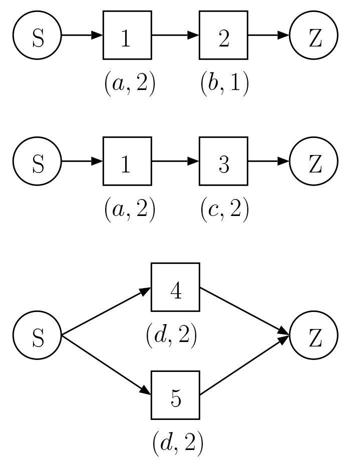


Abbildung 6.4: Die drei beinhalteten Netzpläne

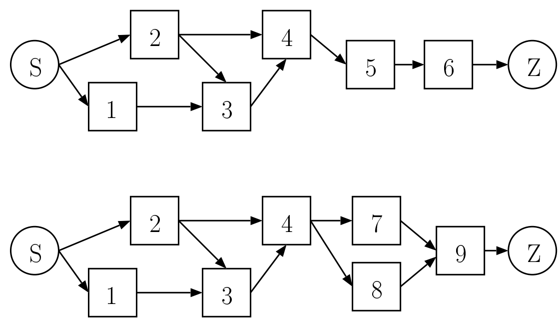


Abbildung 6.5: Die abgeleiteten Netzpläne

Bei dem ersten Netzplan aus Abbildung 6.4, bei dem die mit 1 und 2 gekennzeichneten Vorgänge aus Beispiel 6.1.1 durchgeführt werden (siehe den zugehörigen hierarchischen Netzplan aus Abbildung 6.1), entsteht kein Ressourcenkonflikt. Daraus ergibt sich ein einziger (aktiver) Ablaufplan $\{s(1) = 0,\ s(2) = 2\}$ mit der Gesamtbearbeitungszeit 3. Entsprechend kann aus dem zweiten Netzplan aus Abbildung 6.4 (bei diesem werden die mit 1 und 3 gekennzeichneten Vorgänge durchgeführt) ein einzelner Ablaufplan $\{s(1) = 0,\ s(3) = 2\}$ mit der Gesamtbearbeitungszeit 4 abgeleitet werden. Bei dem dritten Netzplan aus Abbildung 6.4 entsteht ein Ressourcenkonflikt zwischen den beiden Vorgängen 4 und 5. So entstehen hierbei zwei alternative Ablaufpläne je nachdem, in welcher Reihenfolge 4 und 5 angeordnet werden. Die beiden Ablaufpläne benötigen jedoch dieselbe Gesamtbearbeitungszeit 4. Somit entspricht der zu dem ersten Netzplan gehörige Ablaufplan $\{s(1) = 0,\ s(2) = 2\}$ mit der Gesamtbearbeitungszeit 3 dem optimalen Ablaufplan des untersuchten Projektes.

Wie in den vorgestellten Beispielen gesehen, können aus einer h-Aktion aus $\mathcal{RCPSV}$ *exponentiell viele zugehörige Netzpläne* abgeleitet werden. Desweiteren können für einen Netzplan als Folge von Ressourcenkonflikten mehrere *Ablaufpläne* (ressourcenkonfliktfreie Prozeßvarianten) existieren.

Nach Satz 6.2.2 kann eine beliebige h-Aktion aus $\mathcal{RCPSV}$ stets in eine normalisierte Form transformiert werden, bei der alle zugehörigen alternativen Netzpläne durch einen einzelnen Operator '**xor**' verknüpft sind. So kann ein algorithmisches Verfahren entwickelt werden, in dessen Rahmen eine als Eingabe gegebene h-Aktion zuerst in die *normalisierte* Form transformiert wird und dann für jeden zugehörigen Netzplan Ablaufpläne berechnet werden. Diejenigen Ablaufpläne unter all diesen berechneten Ablaufplänen, die den kleinsten Wert (die minimale Gesamtbearbeitungszeit) aufweisen, entsprechen dann den gesuchten optimalen Ablaufplänen für die gegebene h-Aktion.

Alle aus einer beliebigen h-Aktion abgeleiteten Netzpläne spezifizieren stets $\mathcal{RCPS}$-Probleme. Einen Algorithmus, der für ein beliebiges $\mathcal{RCPS}$-Problem optimale Ablaufpläne ermittelt, wird im nächsten Kapitel 7 vorgestellt.

Kapitel 7

Lösung des $\mathcal{RCPSV}$-Problems

Ein beliebiger $\mathcal{RCPSV}$-Ausdruck (ein beliebiges $\mathcal{RCPSV}$-Problem) M kann mittels des in Abschnitt 6.2 definierten Kalküls *in einen semantisch äquivalenten normalisierten Ausdruck* transformiert werden. Wie bereits erwähnt, sind bei dem normalisierten Ausdruck mehrere **xor**-freie Ausdrücke, die alle aus M ableitbaren Netzpläne ($\mathcal{RCPS}$-Probleme) darstellen, durch einen einzelnen Operator '**xor**' verknüpft. Unter Anwendung der semantischen Äquivalenz kann ein beliebiger $\mathcal{RCPSV}$-Ausdruck zuerst in einen normalisierten Ausdruck transformiert werden, bevor die Berechnung der optimalen Ablaufpläne beginnt. Anschließend können zur Bestimmung der Ablaufpläne mit minimaler Laufzeit die zugehörigen Netzpläne einzeln getrennt ausgewertet werden. Diejenigen Ablaufpläne, die hierbei unter allen die geringste Bearbeitungszeit benötigen, entsprechen dann den gesuchten optimalen Ablaufplänen. Diesen Zusammenhang zeigt die Abbildung 7.1.

In diesem Kapitel wird zunächst ein Algorithmus vorgestellt, mit dem für ein beliebiges $\mathcal{RCPS}$-Problem alle nichtredundanten Ablaufpläne berechnet werden können. Anschließend wird gezeigt, wie der Algorithmus in der funktionalen Programmiersprache Haskell ([HAB+96], [PHA+97]) implementiert werden kann. Zum Schluß wird ein Korrektheitsbeweis für den Algorithmus geführt.

7.1 Problembeschreibung

Das $\mathcal{RCPS}$-Problem wurde bereits in Abschnitt 2.1.2 formal spezifiziert. In diesem Abschnitt spezifizieren wir dieses Problem in einer leicht veränderten Form noch einmal.

Ein $\mathcal{RCPS}$-Problem kann wie folgt definiert werden:

A ($\subset \mathbb{N}$) sei eine endliche Menge von Aktivitäten (genauer gesagt Grundaktivitäten) und R sei eine endliche Menge von Ressourcen. Ferner seien die folgenden

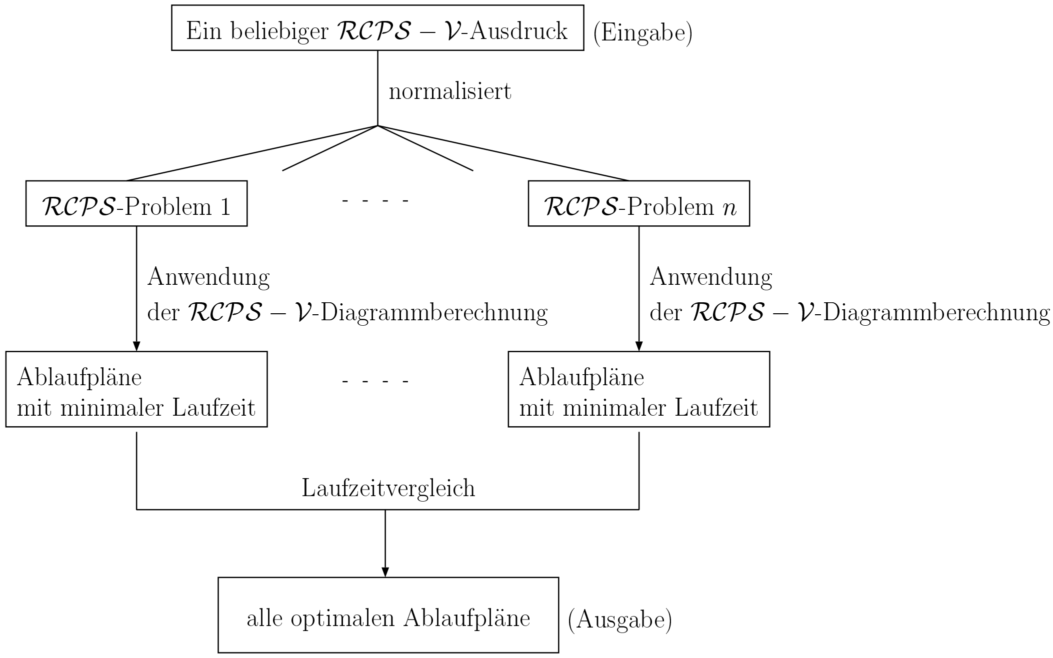


Abbildung 7.1: Übersicht zur Lösung des $\mathcal{RCPSV}$-Problems

Mengen gegeben:

- $X = \{(i, r(i), d(i))|$ für alle $i \in A$, $r(i) \in R$, $d(i) \in \mathbb{R}\}$
- $P = \{(i,j)|i,j \in A\}$

Von X kann abgelesen werden, welche Aktivität welche Ressource wie lange beansprucht. Die Aktivitäten sind hierbei nichts anders als die in Abschnitt 3.1.1 bzw. 6.1 definierten Grundaktionen. Im weiteren werden wir daher Aktivitäten auch als Aktionen bezeichnen. P beschreibt die vorgegebenen Präzedenzbeziehungen zwischen Aktionen. Ein Tupel (i, j) aus P beschreibt, daß die Aktionen i ein direkter Vorgänger der Aktion j ist.
Gesucht wird hierbei *ein optimaler (aktiver) Ablaufplan*

$$\{(i, s(i))| \text{ für alle } i \in A,\ s(i) \in \mathbb{R}\}.$$

$s(i)$ beschreibt hierbei den Zeitpunkt, zu dem die Durchführung der Aktion i beginnt. Wie bereits in Abschnitt 2.1.2 vereinbart, muß ein Ablaufplan die folgenden Nebenbedingungen erfüllen:

1. Präzedenzbeschränkung: $s(j) \geq s(i) + d(i)$ für alle $(i, j) \in P$.

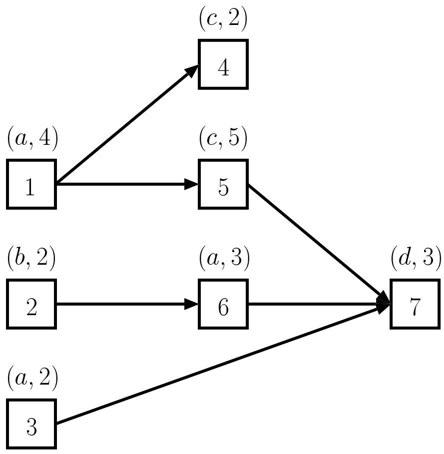


Abbildung 7.2: Netzplan

2. Ressourcenbeschränkung: $s(m) \geq s(l) + d(l)$ oder $s(l) \geq s(m) + d(m)$ für alle $l, m \in A$ mit $r(l) = r(m)$

Im folgenden Abschnitt werden wir unter Zuhilfenahme des in Abschnitt 4.2 benutzten Scan-line-Prinzips einen Lösungsalgorithmus entwickeln.

7.2 Algorithmus zur Lösung des $\mathcal{RCPS}$-Problems

Zunächst soll mittels eines Beispiels die Arbeitsweise des Algorithmus erläutert werden.

Gegeben sei eine Menge von Ressourcen $R = \{a, b, c, d\}$. 7 Aktivitäten

$$X = \{(1, a, 4), (2, b, 2), (3, a, 2), (4, c, 2), (5, c, 5), (6, a, 3), (7, d, 3)\}$$

mit $A = \{1, 2, \cdots, 7\}$ seien durchzuführen. Dabei seien die folgenden Präzedenzbeziehungen gegeben:

$$P = \{(1, 4), (1, 5), (2, 6), (3, 7), (5, 7), (6, 7)\}$$

Die Ablaufstruktur dieses Problems[1] zeigt der Netzplan aus Abbildung 7.2.

[1] Wie bereits in Kapitel 6 beschrieben, kann jedes $\mathcal{RCPS}$-Problem als ein $\mathcal{RCPS}$-Ausdruck dargestellt werden. Der entsprechende $\mathcal{RCPS}$-Ausdruck des betrachteten Beispiels hat folgendes Aussehen:

$$(\mathbf{hnet}[((1, a, 4), (4, c, 2)), ((1, a, 4), (5, c, 5)), ((2, b, 2), (6, a, 3)), \\ ((3, a, 2), (7, d, 3)), ((5, c, 5), (7, d, 3)), ((6, a, 3), (7, d, 3))])$$

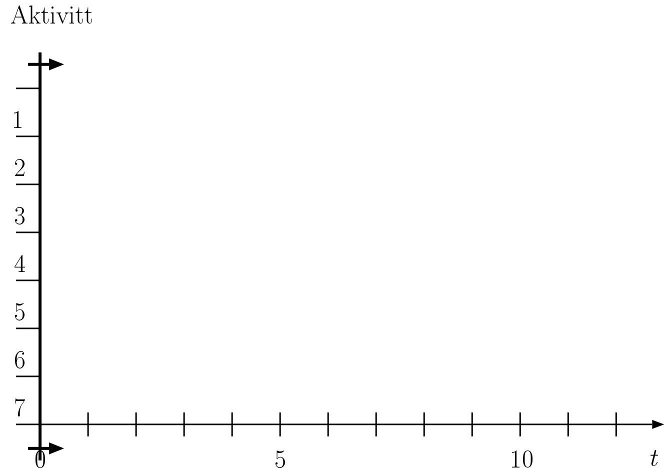


Abbildung 7.3: Diagramm mit einer Scan-line

Ein optimaler Ablaufplan

$$\{(i, s(i))|i \in \{1, 2, \cdots, 7\}\}$$

wird unter Berücksichtigung der zwei Nebenbedingungen "Präzedenzbeschränkung" sowie "Ressourcenbeschränkung" gesucht.

Im Rahmen des im folgenden beschriebenen Algorithmus werden alle nichtredundanten Ablaufpläne berechnet und alle Ablaufpläne mit minimaler Gesamtbearbeitungszeit als optimale Ablaufpläne zurückgegeben. Der Algorithmus arbeitet wie folgt:

Es wird wiederum das bereits im Rahmen der $\mathcal{RSV}$-Diagrammberechnung angewendete Scan-line-Prinzip verwendet. Hierbei wird jedoch ein anderes Diagramm als das $\mathcal{RSV}$-Diagramm als Darstellungshilfsmittel verwendet, da das $\mathcal{RSV}$-Diagramm zur Lösung von $\mathcal{RCPS}$-Problemen ungeeignet ist. Daher nennen wir die zur Lösung des $\mathcal{RCPSV}$-Problems verwendete Diagrammberechnung $\mathcal{RCPSV}$-Diagrammberechnung. Die Begriffe, die wir in Kapitel 4 eingeführt haben, werden in diesem Abschnitt übernommen, sofern sie widerspruchsfrei angewendet werden können.

Am Anfang befindet sich die Scan-line auf dem Zeitpunkt 0 des Diagramms, das aus einer Abszisse und einer Ordinate besteht. Die Abszisse bildet eine Zeitachse, auf der Ordinate werden die Aktionen aufgetragen. Das entstehende Diagramm zeigt Abbildung 7.3.

Ermittlung aller Startaktionen
Zunächst werden alle Startaktionen ermittelt, dies sind alle Aktionen aus X, für die in P kein direkter Vorgänger spezifiziert wird. In unserem Beispiel sind dies die Aktionen $1, 2$ und 3. Anschließend werden bei dem Diagramm die drei Startaktionen an die Scan-line angefügt. "An die Scan-line anfügen" bedeutet dabei

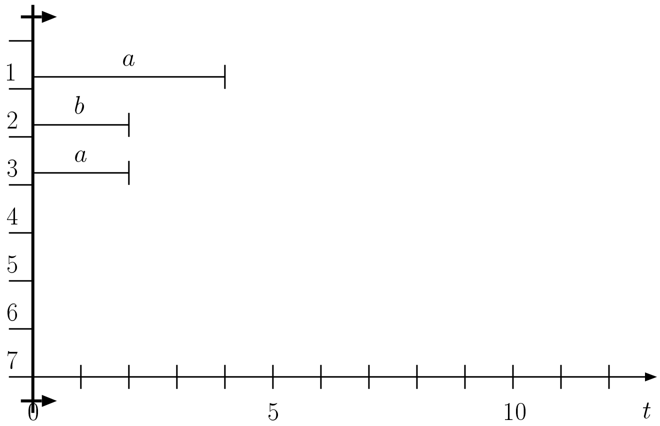


Abbildung 7.4: Startdiagramm

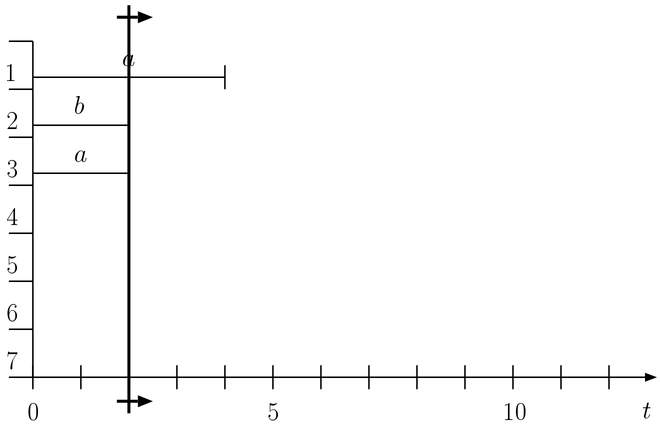


Abbildung 7.5: Scan-line bewegen

wie bei der $\mathcal{RSV}$-Diagrammberechnung, daß bei allen anzufügenden Aktionen die Startzeit der Durchführung auf dem Zeitpunkt 0 festgelegt wird. Das dabei erhaltene Diagramm wird als *Startdiagramm* bezeichnet und zeigt Abbildung 7.4.

Scan-line bewegen
Als nächsten Schritt springt die Scan-line zum nächsten Haltepunkt $t(H) = 2$. Das danach erhaltene Diagramm zeigt Abbildung 7.5.

Ressourcenkonflikte bestimmen und auflösen
Alle fest plazierten Aktionen einfrieren
Im Anschluß wird geprüft, ob Ressourcenkonflikte entstehen. Wenn Ressourcenkonflikte entstehen, werden diese mittels derselben Methodik wie bei der $\mathcal{RSV}$-Diagrammberechnung aufgelöst. Bezogen auf unser Beispiel bedeutet dies:
a ist eine 2-Zeit-direkte Scan-line-Ressource. Da mehr als eine Scan-line-Aktion (1 und 3) die 2-Zeit-direkte Scan-line-Ressource a beanspruchen, entsteht ein Ressourcenkonflikt. Es werden zwei 2-Zeit-Konfliktkombinationen $\{1\}$ und $\{3\}$

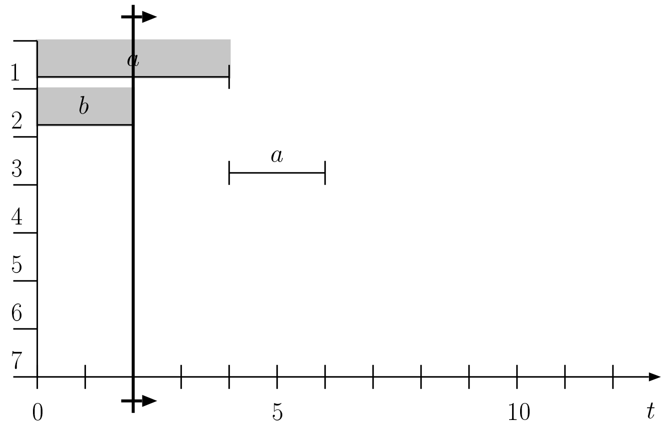


Abbildung 7.6: Ressourcenkonflikte auflösen, Einfrieren fest plazierter Aktionen

gebildet, um den 2-Zeit-Ressourcenkonflikt aufzulösen. Die Auflösung des Ressourcenkonflikts erfolgt durch die Duplizierung des Diagramms und anschließendes Einfrieren jeweils einer am Konflikt beteiligten Aktion. Stellvertretend verfolgen wir das Diagramm, bei dem die $(2, a)$-Konfliktgrundaktion 1 eingefroren wird. Bei diesem Fall ist es erforderlich, die $(2, a)$-Konfliktgrundaktion 3 hinter die $(2, a)$-Konfliktgrundaktion 1 zu verschieben.
Anschließend werden alle 2-Zeit-konfliktfreien Aktionen eingefroren, wenn solche existieren. Beim von uns verfolgten Fall entspricht die Aktion 2 einer 2-Zeit-konfliktfreien Aktion, und wird somit eingefroren. Das entstehende Diagramm zeigt Abbildung 7.6.

Anfügen weiterer Aktionen an die Scan-line
Zunächst wollen wir einen weiteren Begriff *'die t_{SL}-Zeit-anfügbaren Aktionen'* bzw. *'die an die Scan-line anfügbaren Aktionen'* wie folgt definieren:

Definition 7.2.1. *Für ein gegebenes aktuelles Diagramm (D, t_{SL}) und einen Netzplan (X, P) heißt eine Aktion $(i, r(i), d(i)) \in X$ eine t_{SL}-Zeit anfügbare Aktion bzw. die an die Scan-line anfügbaren Aktionen gdw.*

1. *i ist nicht aus dem Diagramm (D, t_{SL}),*
2. *$r(i)$ ist keine t_{SL}-Zeit blockierte Ressource*[2] *und*
3. *alle direkten Vorgänger e von i, d.h. für e und i gilt $(e, i) \in P$, sind bereits Elemente aus (D, t_{SL}) und für alle e gilt $RE(e) \leq t_{SL}$*[3] *bei (D, t_{SL}).*

[2] Wie in Kapitel 4 definiert, ist eine t_{SL}-Zeit blockierte Ressource eine Ressource, der eine Aktion aus dem Diagramm (D, t_{SL}) aufweist, die über der Scanline liegt, eingefroren ist und die Resource benötigt. Eine solche eingefrorene Aktion, die eine Ressource über die Scan-line hinaus belegt, beruht auf eine vorherige Ressourcenkonfliktauflösung (siehe Definition 4.3.4).
[3] siehe Notation 4.2.1

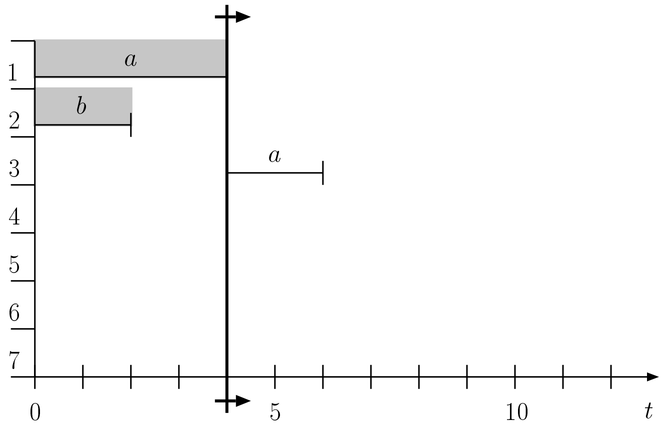


Abbildung 7.7: Scan-line bewegen

Als nächsten Schritt werden alle *an* 2-*Zeit anfügbaren* Aktionen aus X ermittelt und an die Scan-line angefügt. Die Aktionen $4, 5$ und 6 sind alle Aktionen aus X, die noch nicht im Diagramm enthalten sind, bei denen sich jedoch alle direkten Vorgänger bereits im Diagramm befinden. 4 und 5 können nicht an die Scan-line angefügt werden, da sie die dritte Bedingung nicht erfüllen (Für Aktion 1, die ein direkter Vorgänger von 4 ist, gilt z. B. $RE(1) = 4 > 2$.). Die Aktion 6 kann nicht an die Scan-line angefügt werden, da 6 die zweite Bedingung verletzt (Die Ressource a ist bis zum Zeitpunkt $4 = RE(1)$ blockiert.). Insgesamt können somit keine neuen Aktionen angefügt werden.

Im weiteren werden die Schritte

- **Scan-line bewegen,**
- **Ressourcenkonflikte bestimmen und auflösen, alle fest plazierten Aktionen einfrieren**
- **Anfügen weiterer Aktionen an die Scan-line**

rekursiv angewendet, bis alle Aktionen aus X im Diagramm enthalten und eingefroren sind.

Wir verfolgen diesen rekursiven Prozeß nun bei unserem Beispiel weiter.

Nach dem Springen der Scan-line zum nächsten Haltepunkt $t = 4$ sieht das Diagramm wie in Abbildung 7.7 gezeigt aus. Da hierbei kein 4-Zeit Ressourcenkonflikt entsteht und keine 4-Zeit konfliktfreie Grundaktionen existieren, wird zum nächsten Schritt übergegangen, bei dem die Aktionen $4, 5$ und 6 an die Scan-line angefügt werden. Das zu diesem Zeitpunkt erhaltene Diagramm enthält Abbildung 7.8. Das Aussehen des Diagramms nach der anschließenden Bewegung der Scan-line zeigt Abbildung 7.9. Nun existieren zwei 6-Zeit-Konfliktressourcen a und c und in der Folge vier 6-Zeit Konfliktkombinationen

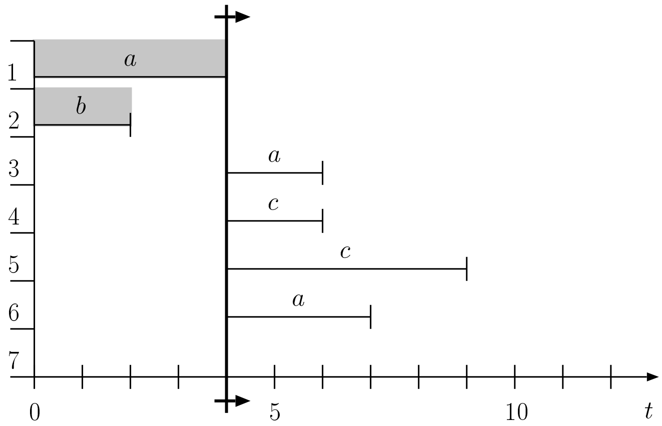


Abbildung 7.8: Anfügen der Aktionen an die Scan-line

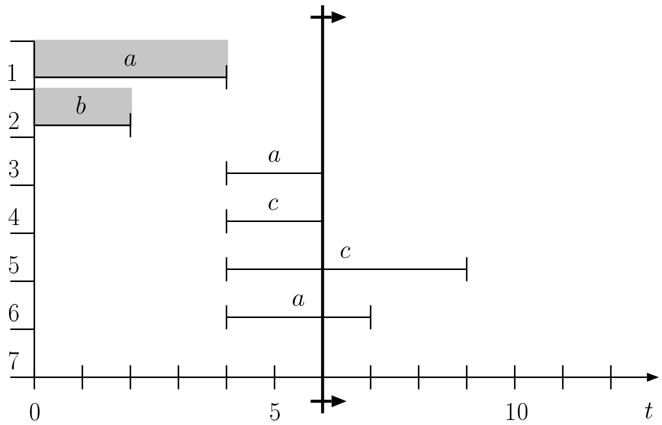


Abbildung 7.9: Scan-line bewegen

$\{[3,4],[3,5],[6,4],[6,5]\}$. Das Diagramm wird entsprechend zu vier Diagrammen vervielfacht. Im Rahmen dieses Beispiels verfolgen wir den Fall weiter, bei dem die beiden zur Kombination $[3,5]$ gehörigen Aktionen 3 und 5 eingefroren werden. Das entstehende Diagramm zeigt Abbildung 7.10.

Als nächsten Schritt wird geprüft, ob die einzig verbliebene Aktion 7 an die Scan-line angefügt werden kann. Die Aktion kann noch nicht angefügt werden, da sie die dritte der oben genannten Bedingungen nicht erfüllt (Die Scan-line liegt immer noch über dem direkten Vorgänger 5.). Anschließend springt die Scan-line zum nächsten Haltepunkt $t(H) = 9$, was der Übergang vom Diagramm aus Abbildung 7.10 zum Diagramm aus Abbildung 7.11 zeigt. Es entsteht kein 9-Zeit-Ressourcenkonflikt und die 9-Zeit-konfliktfreie Aktion 6 wird eingefroren. Das entstehende Diagramm gibt Abbildung 7.12 wieder.

Im nächsten Schritt wird die Aktion 7 an die Scan-line angefügt, da 7 nun alle oben aufgeführten Bedingungen erfüllt. Nach weiteren rekursiven Anwendungen

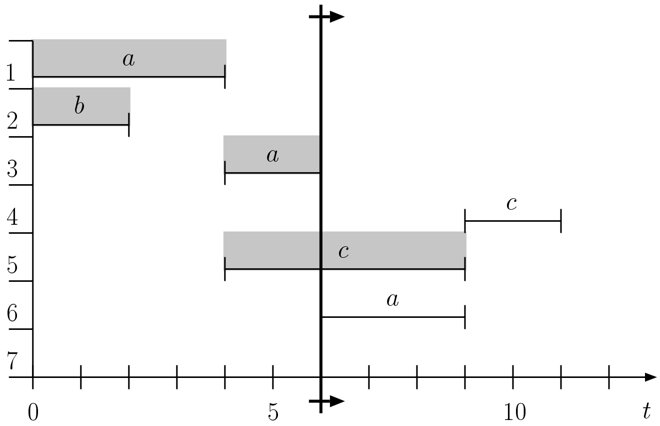


Abbildung 7.10: Ressourcenkonflikte auflösen, Einfrieren fest plazierter Aktionen

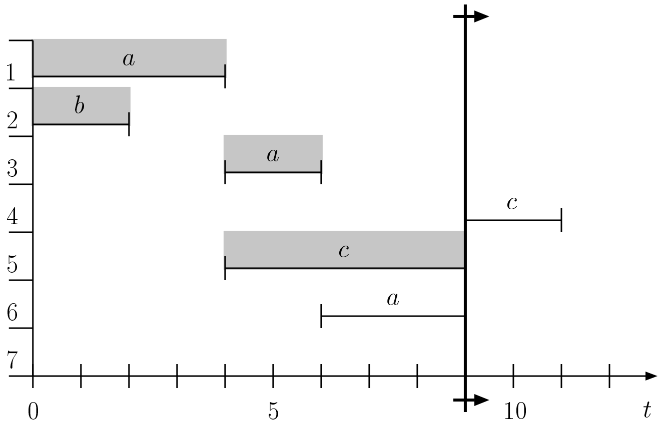


Abbildung 7.11: Scan-line bewegen

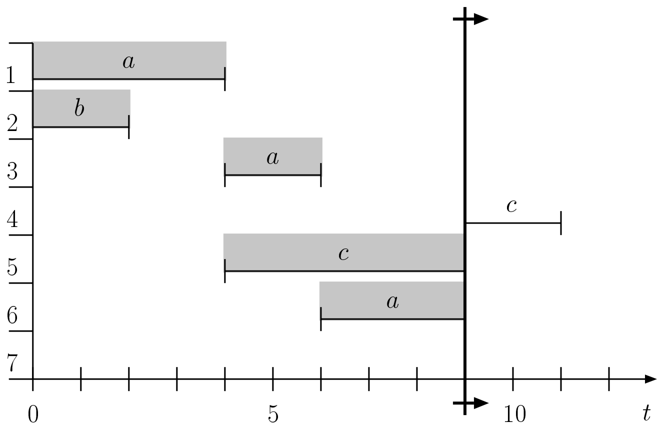


Abbildung 7.12: Einfrieren der fest plazierten Aktion 6

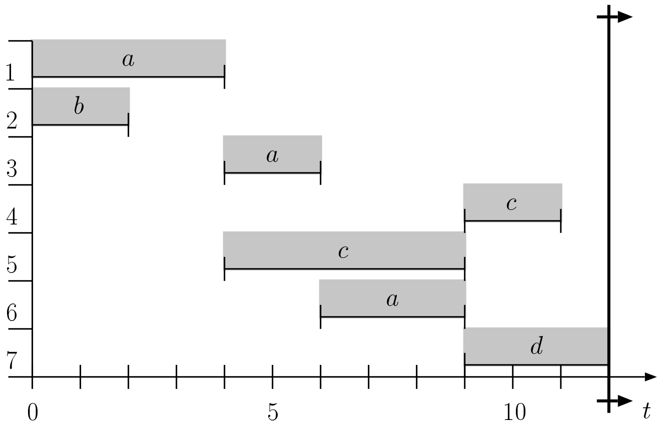


Abbildung 7.13: Ein optimaler Ablaufplan

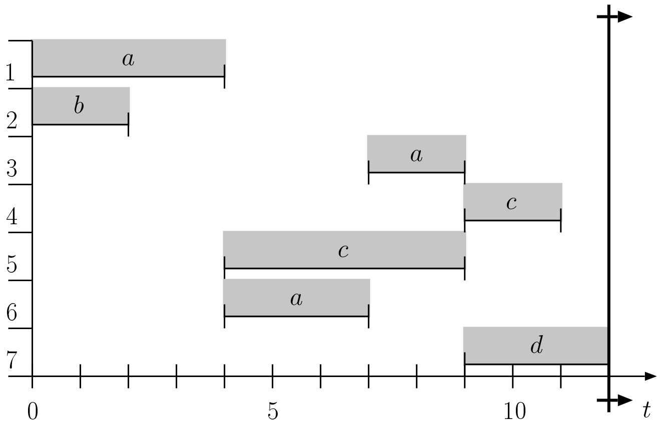


Abbildung 7.14: Ein optimaler Ablaufplan

des Verfahrens erreicht die Scan-line den Zeitpunkt 12 und alle 7 Aktionen sind eingefroren. Als Resultat erhalten wir den in Abbildung 7.13 gezeigten Ablaufplan mit einer Gesamtbearbeitungszeit von 12 Zeiteinheiten.
Insgesamt werden auf diese Weise 8 nichtredundante Ablaufpläne berechnet. Zwei davon benötigen die minimale Gesamtbearbeitungszeit von 12 Zeiteinheiten. Den zweiten optimalen Ablaufplan zeigt das Diagramm aus Abbildung 7.14.

7.2.1 Formale Beschreibung

Nachdem im letzten Abschnitt der Algorithmus zur Lösung des $\mathcal{RCPS}$-Problems informal vorgestellt wurde, wird in diesem Abschnitt eine formale Beschreibung des Algorithmus gegeben. Der Algorithmus hat das folgende Aussehen:

<u>Algorithmus</u>

START:

Gegeben: ein $\mathcal{RCPS}$-Problem $\mathtt{p} = (X, P)$
Gesucht: alle zugehörigen optimalen Ablaufpläne.

1. Erzeuge für $\mathtt{p} = (X, P)$ das Startdiagramm (D, t_{SL}) mit $t_{SL} = 0$

2. *Scan-line bewegen:*
   $t_{SL} :=$ `Nächster_Haltepunkt` $((D, t_{SL}))$

3. *Ressourcenkonflikte bestimmen und auflösen,*
   Alle fest plazierten Aktionen einfrieren:
   IF `Berechne_Konfliktkombination` $((D, t_{SL})) = \phi$ THEN
   $(D, t_{SL}) :=$`Mark` $((D, t_{SL}))$
   ELSE
   `Vervielfache`$((D, t_{SL}))$.

   /* (D, t_{SL}) sei n mal vervielfacht. Es seien die Diagramme $(D, t_{SL})_1, \cdots, (D, t_{SL})_n$ entstanden, wobei n der Anzahl der Elemente der Menge `Berechne_Konfliktkombination`$((D, t_{SL}))$ entspricht. Jedes Diagramm $(D, t_{SL})_i (i = 1, \cdots, n)$ verfolgt eine t_{SL}-Zeit Konfliktkombination.
   Bei jedem $(D, t_{SL})_i (i = 1, \cdots, n)$ wird die folgende Anweisung durchgeführt. */

   $(D, t_{SL}) :=$`Mark`$((D, t_{SL})_i)$

4. *Anfügen der Aktionen an die Scan-line:*

   /* Alle t_{SL}-Zeit anfügbaren Aktionen aus X werden beim Diagramm (D, t_{SL}) an die Scan-line angefügt. Anschließend wird überprüft, ob alle Aktionen aus (D, t_{SL}) eingefroren wurden. Ist es der Fall, repräsentiert das Diagramm einen Ablaufplan (siehe Lemma 7.4.1 aus Abschnitt 7.4). So wird das Diagramm als ein Ablaufplan ausgegeben. Falls nicht, erfolgt ein rekursiver Aufruf ab dem Schritt 2. */

   $(D, t_{SL}) :=$`Anfüge_Aktionen` $((D, t_{SL}))$
   IF `Alle_eingefroren` $((D, t_{SL}))$ THEN
   Gebe (D, t_{SL}) als ein Ablaufplan aus.
   ELSE
   Gehe zu dem Schritt 2.

5. *Ausgeben aller optimalen Ablaufpläne*:
   Vergleiche die t_{SL}'s aller berechneten Ablaufpläne (D, t_{SL});
   gebe die Ablaufpläne (D, t_{SL}) aus, die die minimale Gesamtbearbeitungszeit benötigen.

HALT

Die Teilprozeduren beim obigen Algorithmus erfüllen die folgenden Aufgaben:

- Die Teilprozedur `Anfüge_Aktionen`$((D, t_{SL}))$ fügt alle t_{SL}-Zeit-anfügbaren Aktionen aus X an die Scan-line bei (D, t_{SL}) (siehe Definition 7.2.1) an.
- Die Teilprozedur `Alle_eingefroren` $((D, t_{SL}))$ überprüft, ob alle Aktionen aus (D, t_{SL}) eingefroren wurden.
- Alle übrigen Teilprozeduren `Nächster_Haltepunkt`, `Berechne_Konfliktkombination`, `Mark`, und `Vervielfache` sind aus Abschnitt 4.3 übernommen worden, wo der Algorithmus zur $\mathcal{RSV}$-Diagrammberechnung vorgestellt wird.

7.3 Implementation des Algorithmus

Nachdem im letzten Abschnitt der Algorithmus zur Lösung des $\mathcal{RCPS}$-Problems vorgestellt wurde, wird in diesem Abschnitt eine konkrete Implementation in der funktionalen Programmiersprache Haskell gegeben.

Ein Datentyp, der die Elemente aus der in Abschnitt 7.1 vorgestellten Menge A darstellt, wird wie folgt definiert:

```
>type ActIdent = Int
```

Desweiteren werden die folgenden beiden Datentypen, die Ressourcen sowie zeitliche Dauer darstellen, definiert:

```
>type Resource = Char
>type Duration = Int
```

Somit besitzen Aktionen den folgenden Datentyp

```
>type Action = (ActIdent, Resource, Duration)
```

Nun wird eine Datenstruktur, mittels deren Hilfe Netzpläne dargestellt werden können, wie folgt definiert:

```
>type Netplan = ([Action], [(ActIdent, ActIdent)])
```

Die erste Komponente `[Action] = [(ActIdent, Resource, Duration)]` stellt die in Abschnitt 7.1 beschriebene Menge X dar. Die zweite Komponente `[(ActIdent, ActIdent)]` repräsentiert die Menge aller Präzedenzbeziehungen P.
Eine weitere Datenstruktur `ExtSchedule`, mittels deren Hilfe Diagramme sowie Ablaufpläne dargestellt werden können, wird wie folgt definiert:

```
>type Time        = Int
>type Frozen      = Bool
>type Schedule    = [(Action, Time, Frozen)]
>type ExtSchedule = (Time, Schedule)
```

Die erste Komponente vom Typ `ExtSchedule` bezeichnet den Haltepunkt der Scan-line. Die zweite Komponente vom Typ `Schedule` beschreibt den Zeitpunkt, zu dem die Durchführung der zugehörigen Aktion (die erste Komponente) beginnt. Mittels der dritten Komponente `Frozen` wird gekennzeichnet, ob die Aktion bei dem durch `ExtSchedule` repräsentierten Diagramm eingefroren ist oder nicht. Ist die Aktion eingefroren, nimmt die Komponente `Frozen` den booleschen Wert `True` an.

Ermittlung aller Startaktionen
Die Funktion

```
>startAct :: Netplan -> [Action]
```

ermittelt alle Startaktionen. Startaktionen sind diejenigen Aktionen, die in der Menge P keinen Vorgänger aufweisen. Wird die Funktion `startAct` auf einen Netzplan angewendet, wird eine aus allen Startaktionen bestehende Menge (`[Action]`) als Resultat zurückgegeben.

Die folgende Funktion

```
>startSchedule :: Netplan -> ExtSchedule
>startSchedule p = (0, [(a, 0, False) | a <- startAct p])
```

erzeugt für einen gegebenen Netzplan ein Startdiagramm, auf dem sich die Scan-line auf dem Zeitpunkt 0 befindet und alle Startaktionen an die Scan-line angefügt sind (Allen Startaktionen wird die Startzeit 0 gegeben.). Als Wert der Variable `Frozen` wird der boolesche Wert `False` angenommen.

Scan-line bewegen
Die Funktion, die die Scan-line auf einem Diagramm zum nächsten Haltepunkt bewegt, nennen wir `moveScanline` und wird wie folgt definiert:

```
>moveScanline :: ExtSchedule -> ExtSchedule
>moveScanline e@(t, s) = (nextStop e, s)
```

Hierbei wird die den nächsten Haltepunkt ermittelnde Hilfsfunktion `nextStop` zur Hilfe genommen. Diese hat den Typ

```
>nextStop :: ExtSchedule -> Time
```

und ihre Definition kann dem Anhang entnommen werden.

Ressourcenkonflikte bestimmen und auflösen,
Alle fest plazierten Aktionen einfrieren

Um Ressourcenkonflikte zu bestimmen und aufzulösen, wird die folgende Funktion definiert:

```
>solveConflicts :: ExtSchedule -> [ExtSchedule]
>solveConflicts e@(t, s)
>     = case scanLineGrp e of
>         [] -> [e]
>         gs -> [(t, moveAct xs s) | xs <- combine gs]
```

Die Hilfsfunktion

```
>scanLineGrp :: ExtSchedule -> [[Action]]
```

berechnet eine Liste von Listen (Mengen). Eine Liste `[Action]` besteht dabei aus Aktionen, die an einem $t_{SL}(= \mathtt{t})$-Zeit-Ressourcenkonflikt beteiligt sind. Die zweite case-Alternative von `solveConflicts`

```
>gs -> [(t, moveAct xs s) | xs <- combine gs]
```

leistet die bei der Beschreibung des Algorithmus aufgeführte Vervielfachung der Diagramme.

Die Hilfsfunktion `combine` nimmt das Resultat von `scanLineGrp e` und liefert als Ausgabe eine Liste von Listen `[[(Action, [Action])]]`.

```
>combine :: [[Action]] -> [[(Action, [Action])]]
```

Ein Element vom Typ `[(Action, [Action])]` des Resultats beschreibt hierbei die folgende Menge:

Die die erste Komponente `Action` darstellende Aktion und die zu der zweiten Komponente `[Action]` gehörigen Aktionen beschreiben gemeinsam diejenigen Aktionen, die an einem `t`-Zeit-Ressourcenkonflikt beteiligt sind. Die erste Komponente des Tupels `(Action, [Action])` beschreibt eine einzufrierende (ausgewählte) Aktion und die zu der zweiten Komponente des Tupels `(Action, [Action])` gehörigen Aktionen beschreiben alle Aktionen, die hinter diese eingefrorene Aktion zu verschieben sind. Die aus allen ersten Komponenten der Tupel `(Action, [Action])` bei `[(Action, [Action])]`bestehende Menge entspricht einer `t`-Zeit-Konfliktkombination.

Die Hilfsfunktion `moveAct` verfügt über die folgende Definition:

```
>moveAct :: [(Action, [Action])] -> Schedule -> Schedule
>moveAct []              s = s
>moveAct (g@(a, as):xs) s = moveAct xs [moveSlot x g s| x <- s]
```

`moveAct` nimmt als Argumente das oben beschriebene Ergebnis von `combine` sowie ein Diagramm (ohne Scan-line) `s` und verschiebt sowie friert in `s` Aktionen gemäß des ersten Arguments vom Typ `[(Action, [Action])]`. Dies geschieht unter Zuhilfenahme der Funktion `moveSlot`

```
>moveSlot :: (Action, Time, Frozen) -> (Action, [Action])
                        -> Schedule -> (Action, Time, Frozen)
```

deren Definition im Anhang gefunden werden kann.

Die Funktion `frzActScl`, die alle t_{SL}-Zeit konfliktfreien Aktionen einfriert d.h. die für alle t_{SL}-Zeit-konfliktfreien Aktionen den booleschen Wert der Variable `Frozen` von `False` zu `True` wechselt, besitzt folgende Definition:

```
>frzActScl :: ExtSchedule -> ExtSchedule
>frzActScl (t, s)
>   = (t, [check x | x <- s])
>     where
>       check x@(a, t_s, b)
>         | t_s + (durAct a) == t  = (a, t_s, True)
>         | otherwise              = x
```

Anfügen der Aktionen an die Scan-line
Wie bereits beschrieben, wird den t_{SL}-Zeit-anfügbaren Aktionen beim Anfügen an die Scan-line als Startzeit der Zeitpunkt t_{SL} zugewiesen, auf dem sich die Scan-line befindet. Die Funktion `getAct`, die unter Zuhilfenahme weiterer Hilfsfunktionen alle an die Scan-line anfügbaren Aktionen berechnet und anschließend anfügt, wird wie folgt definiert:

```
>getAct :: ExtSchedule -> Netplan -> ExtSchedule
>getAct e@(t, s) p@(as, ps)
>      = (t, s ++ [(a, t, False)| a <- as4])
>        where
>         bs  = blocRes e
>         as0 = [a | (a, _, _) <- s]
>         as1 = [a | (a@(_, _, t_d), t_s, _) <- s, (t_s + t_d) <= t]
>         as2 = as \\ as0
>         as3 = [a | a@(i', _, _) <- as2,
>                    ((predAct a p) \\ as1) == []]
>         as4 = [a | a@(_, r, _) <- as3, notElem r bs]
```

Unter Zuhilfenahme der Hilfsfunktion `blocRes` wird eine Menge, die aus den $t_{SL}(=$ `t`)-Zeit blockierten Ressourcen besteht, ermittelt. Diese Funktion hat den folgenden Typ:

```
>blocRes :: ExtSchedule -> [Resource]
```

Eine Liste `as0` wird berechnet, die aus allen Aktionen des Diagramms `e` besteht. Die Liste `as1` besteht aus denjenigen Aktionen des Diagramms `e`, deren rechter Endpunkt links von der Scan-line oder auf der Scan-line liegt. Die Liste `as2` besteht aus allen Aktionen, die im Netzplan vorkommen, jedoch nicht in das Diagramm `e` aufgenommen wurden. Die Liste `as3` enthält alle Aktionen aus `as2`, bei denen alle Vorgänger einen rechten Endpunkt auf oder links von der Scanline haben. Die Liste `as4` entspricht allen Aktionen aus `as3`, die eine nicht-blockierte Ressource aufweisen. Die Aktionen aus der Liste `as4` entsprechen den an die Scan-line anfügbaren Aktionen. Diese werden an die Scan-line bei dem aktuellen Diagramm `(t, s)` angefügt, wodurch ein erneuertes aktuelles Diagramm

```
(t, s ++ [(a, t, False)| a <- as4])
```

erhalten wird.

Kernteil des Programms
Die Funktion `solve`, die alle nichtredundanten Ablaufpläne ermittelt, wird wie folgt definiert:

```
>solve :: Netplan -> [ExtSchedule]
>solve p = core p (startSchedule p)

>core :: Netplan -> ExtSchedule -> [ExtSchedule]
>core p e@(_, s)
>   | elem False [b | (_, _, b) <- s]
>       = concat [core p (getAct (frzActScl e') p) |
>                 e' <- solveConflicts (moveScanline e)]
>   | otherwise
>       = [e]
```

Die Hilfsfunktion `core` wird ab dem Schritt **Scan-line bewegen** (`moveScanline`) rekursive angewendet, solange eine nicht-eingefrorene Aktion aus dem Netzplan existiert (siehe Lemma Terminierungskriterium). Wie bereits erläutert, beschreibt der Teil dieser Definition

```
solveConflicts (moveScanline e)
```

die Entstehung der vervielfachten Diagramme. Schließlich werden Diagramme

```
[ExtSchedule]
```

als Resultat geliefert, die alle nichtredundanten Ablaufpläne repräsentieren. Die Funktion, die alle optimalen Ablaufpläne ermittelt, nennen wir `extSolve` und wird wie folgt definiert:

```
>extSolve :: Netplan -> [ExtSchedule]
>extSolve p = [e | e@(t, _) <- es, t == tMin]
>             where
>               tMin = minimum [t | (t, _) <- es]
>               es   = solve p
```

Wie bereits erwähnt, entsprechen diejenige Ablaufpläne, die die minimale Gesamtlaufzeit benötigen, den gesuchten optimalen Ablaufplänen.

Die Funktion `schedStr` hat den Typ

```
>schedStr :: ExtSchedule -> [Char]
```

und dient zur formatierten Ausgabe von Diagrammen am Bildschirm. Ihre genaue Definition kann dem Anhang entnommen werden. Das folgende Beispiel zeigt das Aussehen der Eingabe sowie der Ausgabe am Bildschirm, bei der soeben vorgestellten Implementation.

Beispiel 7.3.1. Wir betrachten das am Anfang dieses Abschnitts eingeführte $\mathcal{RCPS}$-Problem. Im Rahmen des Programms kann dieses Problem wie folgt formuliert werden:

```
>plan :: Netplan
>plan = ([(1, 'a', 4),
>         (2, 'b', 2),
>         (3, 'a', 2),
>         (4, 'c', 2),
>         (5, 'c', 5),
>         (6, 'a', 3),
>         (7, 'd', 3)],
>        [(1,4), (1, 5), (2, 6), (3,7), (5, 7), (6, 7)])
```

Die Eingabe für dieses Problem hat dann folgendes Aussehen:

```
prtschedStrLst (extSolve plan)
```

Als Ausgabe werden die bereits gezeigten Diagramme formatiert am Bildschirm ausgegeben:

```
F  1|aaaa
F  2|bb
F  3|    aa
F  4|         cc
F  5|    ccccc
F  6|      aaa
F  7|         ddd
Scan-Line: 12

F  1|aaaa
F  2|bb
F  3|       aa
F  4|         cc
F  5|    ccccc
F  6|    aaa
F  7|         ddd
Scan-Line: 12
```

Der Buchstabe `F` zu Beginn jeder Zeile symbolisiert, daß die zugehörige Aktion eingefroren wurde.

7.4 Korrektheitsbeweis

In diesem Abschnitt beweisen wir formal, daß der im letzten Abschnitt vorgestellte Algorithmus für ein beliebiges $\mathcal{RCPS}$-Problem alle nichtredundanten optimalen Ablaufpläne ermittelt. Zunächst wird gezeigt, daß die Berechnung eines Diagramms nicht abgeschlossen werden kann, bevor alle Aktionen aus X eines Netzplans (X, P) in das Diagramm aufgenommen wurden.

Lemma 7.4.1. *Gegeben sei ein Netzplan* (X, P). *Die Berechnung eines Diagramms* D *wird nur dann abgeschlossen, wenn alle Aktionen aus* X *bereits in* D *aufgenommen sowie eingefroren wurden.*

Beweis. (D, t_{SL}) sei ein beliebiges nach der Durchführung des Schritts “Anfügen der Aktionen an die Scan-line” entstehendes Diagramm. Falls bei (D, t_{SL}) mindestens eine nicht eingefrorene Aktion aus X existiert, erfolgt ein rekursiver Aufruf. Sind alle Aktionen bei (D, t_{SL}) eingefroren, wird die Berechnung des Diagramms (D, t_{SL}) abgeschlossen.
Wenn bei X mindestens eine Aktion existiert, die noch nicht in (D, t_{SL}) aufgenommen wurde, dann existiert bei (D, t_{SL}) mindestens eine Aktion, die noch einzufrieren ist. Dies gilt aus dem folgenden Grund:

Sind durch die Durchführung des Schritts “Anfügen der Aktionen an die Scan-line” beim Diagramm (D, t_{SL}) eine bzw. mehrere Aktionen aus X aufgenommen worden, dann existiert trivialerweise mindestens eine einzufrierende Aktion bei (D, t_{SL}). Wurde bei letzterem Schritt keine Aktion an die Scan-line angefügt, obwohl noch mindestens eine Aktion aus X existiert, die noch nicht in (D, t_{SL}) aufgenommen wurde, dann wurde das Anfügen weiterer Aktionen an die Scan-line weiterer Aktionen aus einem der folgenden Gründe verhindert (siehe 7.2.1):

1. Ressourcenbeschränkung, d.h. die benötigten Ressourcen sind zu dem Zeitpunkt blockiert oder

2. Präzedenzbedingung, d.h. die Scan-line hat bei (D, t_{SL}) noch nicht alle direkten Vorgänger überschritten.

Bei den beiden obigen Fällen existiert noch mindestens eine Aktion aus (D, t_{SL}), die die Scan-line noch nicht überschritten hat. Damit existiert noch mindestens eine einzufrierende Aktion. Solange bei einem Diagramm noch mindestens eine einzufrierende Aktion existiert, erfolgt wie bereits erwähnt ein rekursiver Aufruf. Somit wird die Berechnung eines Diagramms nur dann abgeschlossen, wenn alle Aktionen aus X bereits in D aufgenommen sowie eingefroren wurden. $\square$

Im folgenden Satz wird bewiesen, daß der Algorithmus terminiert.

Satz 7.4.1. *Der Algorithmus terminiert.*

Beweis. Der Algorithmus terminiert, da bei jedem oder jedem zweiten rekursiven Aufruf ab dem Schritt 2 des Algorithmus die Anzahl der nicht-eingefrorenen Aktionen immer kleiner wird. Wenn alle Aktionen eingefroren worden sind, ist die Berechnung eines Diagramms abgeschlossen. □

Lemma 7.4.2. *Gegeben sei ein Netzplan (X, P). Der Algorithmus angewendet auf (X, P) liefert alleinig Ablaufpläne, bei denen die Präzedenzbedingungen aus P eingehalten werden.*

Beweis. Die vorgegebenen Präzedenzbedingungen werden eingehalten, da Aktionen nur dann an die Scan-line angefügt werden können, wenn die Scan-line bereits alle ihre Vorgänger überschritten hat. □

Lemma 7.4.3. *Mit einer eingefrorenen Aktion entsteht im Verlauf einer verbleibenden Berechnung kein Ressourcenkonflikt.*

Beweis. Wenn eine Aktion A eingefroren wird, ist sie entweder eine konfliktfreie Aktion oder eine Aktion, die bei der Auflösung eines Konflikts ausgewählt wurde. Ist A eine konfliktfreie Aktion, so befindet sich die Scan-line bereits auf dem rechten Endpunkt von A. Daraus folgt direkt, daß A im Verlauf der weiteren Berechnung an keinem Ressourcenkonflikt beteiligt sein kann. Im anderen Fall wurde A im Rahmen der Auflösung eines Ressourcenkonflikts ausgewählt und die Scan-line befindet sich entweder auf dem rechten Endpunkt von A oder über A. Falls die Scan-line den rechten Endpunkt von A angetroffen hat, kann A im Verlauf der weiteren Berechnung an keinem Ressourcenkonflikt beteiligt sein. Liegt die Scan-line über A, kann A ebenfalls im Verlauf der weiteren Berechnung an keinem Ressourcenkonflikt beteiligt sein, da keine weitere dieselbe Ressource benötigenden Aktionen an die Scan-line angefügt werden, solange die Scan-line A nicht überschritten hat (siehe Definition der Funktion `blocRes`). □

Aus obigem Lemma ergibt sich folgender Satz:

Korollar 7.4.1. *Jeder berechnete Ablaufplan ist ressourcenkonfliktfrei.*

Im Verlauf des algorithmischen Verfahrens werden alle Aktionen nacheinander eingefroren. Das Einfrieren einer Aktion bedeutet, daß sie auf dem Diagramm endgültig fest plaziert wird. Das folgende Lemma zeigt, daß ein durch das Verfahren ermittelter Ablaufplan auch mittels der zugehörigen Folge, in der alle Aktionen eingefroren werden, dargestellt werden kann.

Lemma 7.4.4. *Für einen beliebigen Netzplan $\mathtt{p} = (X, P)$ sei ein Ablaufplan $\mathcal{P}$ durch das algorithmische Verfahren ermittelt. F sei die dabei entstehende Folge des Einfrierens aller Aktionen aus $\mathtt{p}$. Wenn die Aktionen in der Folge F eingeplant werden, wird der Ablaufplan $\mathcal{P}$ erhalten.*

Beweis. Dies kann ähnlich wie Lemma 4.5.6 bewiesen werden. □

Im folgenden Lemma beweisen wir, daß das Verfahren alle zu einem Netzplan gehörigen nichtredundanten Folgen des Einfrierens aufzählt.

Lemma 7.4.5. *Für einen beliebigen Netzplan $\mathtt{p} = (X, P)$ werden bei dem Algorithmus zur Lösung des $\mathcal{RCPS}$-Problems alle nichtredundanten Folgen des Einfrierens aufgezählt.*

Beweis. Wir führen eine Induktion über die Anzahl der einzufrierenden Aktionen bei $\mathtt{p}= (X, P)$.

Induktionsanfang: Falls X aus einer einzelnen Aktion besteht, so existiert nur eine einzige Folge des Einfrierens, die aus der einzelnen Aktion besteht. Bei dem Algorithmus wird diese aufgezählt.

Induktionsschritt: Am Anfang befindet sich die Scan-line auf dem Zeitpunkt 0. Zunächst werden alle Startaktionen ermittelt und an die Scan-line angefügt. Danach wird die Scan-line zum nächsten Haltepunkt bewegt; dieser Haltepunkt sei $\mathtt{l}$. Es sind nun die folgenden zwei Fälle zu unterscheiden.
Fall 1: Alle $\mathtt{l}$-Zeit direkten Scan-line-Aktionen sind $\mathtt{l}$-Zeit konfliktfreie Aktionen. Da in diesem Fall kein l-Zeit-Ressourcenkonflikt entsteht, wird zum nächsten Berechnungsschritt übergegangen, bei dem alle $\mathtt{l}$-Zeit konfliktfreien Aktionen eingefroren werden. Alle $\mathtt{l}$-Zeit direkten Scan-line-Aktionen werden nacheinander eingefroren, wodurch eine Einfrierungsreihenfolge für die $\mathtt{l}$-Zeit direkten Scan-line-Aktionen bestimmt wird. Diese Folge sei f. Die Anzahl der noch einzufrierenden Aktionen aus X ist nun kleiner. Die aus diesen noch einzufrierenden Aktionen aus X bestehende Menge sei X'. Nach Induktionsannahme werden für X' alle nicht redundanten Folgen des Einfrierens aufgezählt. Am Anfang dieser nicht redundanten Folgen des Einfrierens fügen wir die Folge f hinzu. So erhalten wir alle nicht redundanten Folgen des Einfrierens für $\mathtt{p}= (X, P)$, die durch den Algorithmus aufgezählt werden.
Fall 2: Es entstehen einer oder mehrere Ressourcenkonflikte, d.h. es existieren eine oder mehrere $\mathtt{l}$-Zeit-Konfliktressourcen. Zunächst werden die Ressourcenkonflikte aufgelöst. Hierbei wird das Diagramm zu mehreren Diagrammen vervielfacht. $D_1, \cdots, D_k$ seien alle Diagramme, die vom Startdiagramm aus vervielfacht wurden und bei denen zum ersten Mal Aktionen aus X eingefroren werden. Gemäß Lemma 7.4.1 und Satz 7.4.1 werden früher oder später vom Startdiagramm aus derartige Diagramme erhalten. Wir betrachten zunächst D_1. Die Folge, in der

Aktionen bei D_1 eingefroren wurden, bezeichnen wir als F_1. Im Anschluß erfolgt rekursiver Aufruf. X_1' sei die Menge der noch bei D_1 einzufrierenden Aktionen aus X. Nach Induktionsannahme zählt das Verfahren für X_1' alle nicht redundanten Folgen des Einfrierens auf, die mit F_1 beginnen. Für alle sonstige $D_i (i = 2, \cdots, k)$ verläuft die Aufzählung aller nicht redundanten Folgen des Einfrierens, die mit F_i beginnen, genau so wie bei D_1. Somit haben wir wegen der Gültigkeit von $F_1 \neq F_2 \cdots \neq F_k$ sowie $X_1 \neq X_2' \neq \cdots \neq X_k'$ gezeigt, daß alle nicht redundanten Folgen des Einfrierens für $\mathtt{p}= (X, P)$ aufgezählt werden. □

Aus den Lemmata 7.4.2, 7.4.1 und 7.4.5 folgt der Satz:

Satz 7.4.2. *Für einen beliebigen Netzplan $\mathtt{p}= (X, P)$ berechnet der Algorithmus zur Lösung des $\mathcal{RCPS}$-Problems alle nichtredundanten Ablaufpläne von $\mathtt{p}$.*

Beweis. Gemäß Lemma 7.4.4 repräsentiert eine Folge des Einfrierens einen Ablaufplan. Der Algorithmus berechnet alle nichtredundanten Ablaufpläne von $\mathtt{p}$, da gemäß Lemma 7.4.5 bei dem Algorithmus alle nichtredundanten Folgen des Einfrierens aufgezählt werden. □

Satz 7.4.3. *Für einen beliebigen Netzplan $\mathtt{p}= (X, P)$ ermittelt der Algorithmus zur Lösung des $\mathcal{RCPS}$-Problems alle optimalen Ablaufpläne.*

Beweis. Lemma 7.4.5 besagt, daß der Algorithmus alle nichtredundanten Ablaufpläne ermittelt. Nach Ermittlung aller nichtredundanten Ablaufpläne vergleicht der Algorithmus die zugehörigen Laufzeiten und gibt als Ergebnis des Vergleichs alle optimalen Ablaufpläne aus. □

7.5 Abschließende Bemerkungen

Wie bereits erwähnt, kann ähnlich wie beim $\mathcal{RSV}$- Problem ein $\mathcal{RCPSV}$-Ausdruck N durch Anwendung des in Abschnitt 6.2 definierten Kalküls in einen semantisch äquivalenten Ausdruck M transformiert werden, bei dem alle in dem Ausdruck N enthaltenen *nichtredundanten* Netzpläne durch einen einzelnen Operator '**xor**' verknüpft sind. Ferner konnten für jeden solchen Netzplan mittels des in diesem Kapitel vorgestellten Algorithmus alle herleitbaren *nichtredundanten* Ablaufpläne berechnet werden. Somit wird ein Berechnungsmodell erhalten, mit dem für ein beliebiges $\mathcal{RCPSV}$-Problem alle *nichtredundanten optimalen* Ablaufpläne berechnet werden können. Dies kann mittels eines ähnlichen Bild wie Abbildung 4.24 illustriert werden. Wie auch bei der abschließenden Bemerkung von Kapitel 4 liegt wiederum die Betonung auf dem Wort "nichtredundant". Die Nichtredundanz ist ein wesentlicher Vorteil gegenüber naiven Verfahren zur Lösung des $\mathcal{RCPSV}$-Problems.

Kapitel 8

$\mathcal{RSV}$ und $\mathcal{RCPSV}$ sind gleich mächtig

Abbildung 8.1, die die Sub- bzw. Oberklassenbeziehungen der Schedulingprobleme darstellt, wurde bereits in Abschnitt 2.1.4 vorgestellt. Im Bezug auf obige Abbildung ist jedoch zu beachten, daß die Sprache $\mathcal{RCPSV}$ nur *von der syntaktischen Repräsentationsfähigkeit eines Problems her* eine echte Oberklasse der Sprache $\mathcal{RSV}$ bildet, d.h. es existieren Probleme, die mittels der Sprache $\mathcal{RCPSV}$ viel kompakter dargestellt werden können als mittels $\mathcal{RSV}$. Von der Ausdruckskraft her sind die beiden Sprachen $\mathcal{RSV}$ und $\mathcal{RCPSV}$ (wie bei $\mathcal{RSVX}$ und $\mathcal{RSV}$) jedoch gleich mächtig, d.h. jeder $\mathcal{RCPSV}$-Ausdruck kann als ein $\mathcal{RSV}$-Ausdruck dargestellt werden und umgekehrt. In Abschnitt 8.1 wird die Gleichmächtigkeit der beiden Sprachen $\mathcal{RSV}$ und $\mathcal{RCPSV}$ formal bewiesen. In Abschnitt 8.2 wird erläutert, was genau unter der Formulierung "ein Problem kann mittels der Sprache $\mathcal{RCPSV}$ viel kompakter dargestellt werden als mittels $\mathcal{RSV}$" zu verstehen ist.

8.1 $\mathcal{RSV}$ und $\mathcal{RCPSV}$ sind gleich mächtig

Es ist offensichtlich, daß ein beliebiger $\mathcal{RSV}$-Ausdruck als ein $\mathcal{RCPSV}$-Ausdruck dargestellt werden kann. Um die umgekehrte Richtung zu beweisen, wird zunächst gezeigt, daß ein beliebiger aktiver Ablaufplan σ, der mittels der $\mathcal{RCPSV}$-Diagrammberechnung aus einem Netzplan $\mathsf{p}= (X, P)$ eines gegebenen $\mathcal{RCPSV}$-Problems hergeleitet (berechnet) wurde, als eine reduzierte Aktion $A(\sigma)$ aus $\mathcal{RSV}$ dargestellt werden kann. Zur Erleichterung des Verständnis des Beweises soll zunächst ein Beispiel betrachtet werden. Wir greifen das Beispiel aus Abschnitt 7.2 auf. Im Rahmen dieses Beispiels wurde das $\mathcal{RCPS}$-Problem, das der Netzplan aus Abbildung 8.2 darstellt, mittels der $\mathcal{RCPSV}$-Diagrammberechnung gelöst.

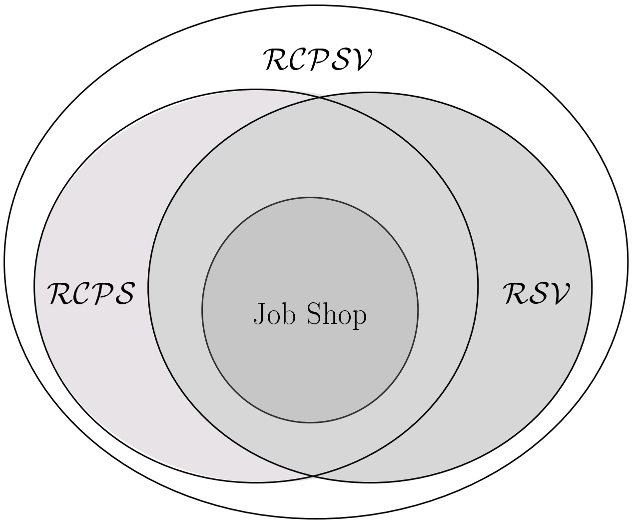


Abbildung 8.1: Sub- bzw. Oberklassenbeziehungen der Schedulingprobleme bzgl. der syntaktischen Repräsentationsfähigkeit

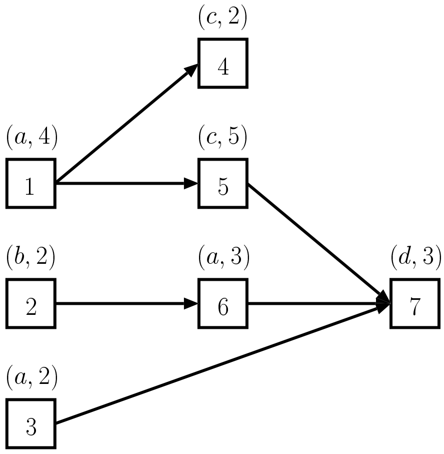


Abbildung 8.2: Ein Netzplan

Im Rahmen der Problemlösung wurden 8 nichtredundante aktive Ablaufpläne erhalten. Einer dieser aktiven Ablaufpläne, nämlich der in Abbildung 8.3 gezeigte, wird nun im weiteren als reduzierte Aktion aus $\mathcal{RSV}$ dargestellt. Wir bezeichnen den aktiven Ablaufplan aus Abbildung 8.3 mittels σ und die zugehörige reduzierte Aktion aus $\mathcal{RSV}$ mittels $A(\sigma)$.

Die reduzierte Aktion $A(\sigma)$ kann unter Zuhilfenahme der $\mathcal{RCPSV}$-Diagrammberechnung durch folgendes Vorgehen aus σ konstruiert werden:
Im weiteren sei n die Anzahl der Aufrufe des Schritts "Einfrieren" bei der Berechnung des Ablaufplans σ mittels der $\mathcal{RCPSV}$-Diagrammberechnung. Bei jedem Aufruf des Schritts "Einfrieren" wird ein Teil $A(\sigma)_i$ von $A(\sigma)$, der al-

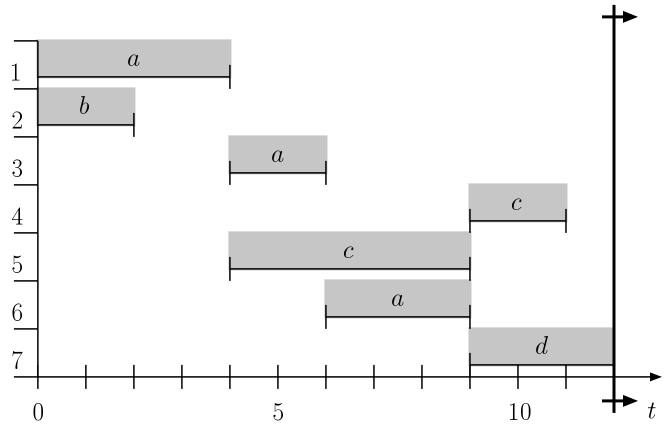


Abbildung 8.3: Ein aktiver Ablaufplan

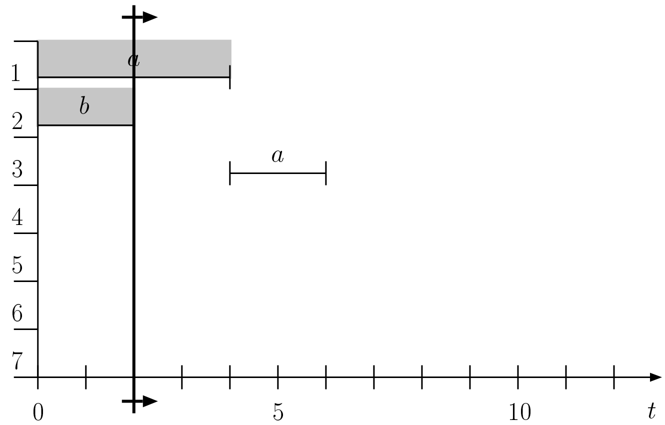


Abbildung 8.4: Ressourcenkonflikte auflösen, Einfrieren fest plazierter Aktionen

le bis zu dem Zeitpunkt eingefrorenen Aktionen enthält, wie im Anschluß beschrieben konstruiert. Auf diese Weise kann eine Folge von RSV-Ausdrücken $A(\sigma)_0, A(\sigma)_1, \cdots A(\sigma)_n$ erhalten werden, bei der $A(\sigma)_0$ dem leeren Ausdruck und $A(\sigma)_n$ dem Ausdruck $A(\sigma)$ entspricht. Die Bildung der $A(\sigma)_i$ für $i = 0, \cdots, n$ geschieht wie folgt:

Nach dem *ersten* Aufruf des Schritts "Einfrieren" wurde bei dem vorherigen Beispiel das in Abbildung 8.4 gezeigte Diagramm erhalten.

$A(\sigma)_1$ wird aus den beiden eingefrorenen Aktionen 1 und 2 gebildet, deren Beginnzeit der Durchführung 0 entspricht, d.h. für die beiden eingefrorenen Aktionen 1 und 2 gilt $LE(1) = LE(2) = 0$. Da zu diesem Zeitpunkt nicht klar ist, ob die beiden Aktionen Nachfolger besitzen werden, werden beide Aktionen mittels eines '**seq**'-Operators gekapselt und somit die Subaktionen **seq** 1 und **seq** 2 erhalten. Offensichtlich werden beide Aktionen parallel durchgeführt, entsprechend erfolgt eine Verknüpfung der beiden soeben erhaltenen Subaktionen mittels eines

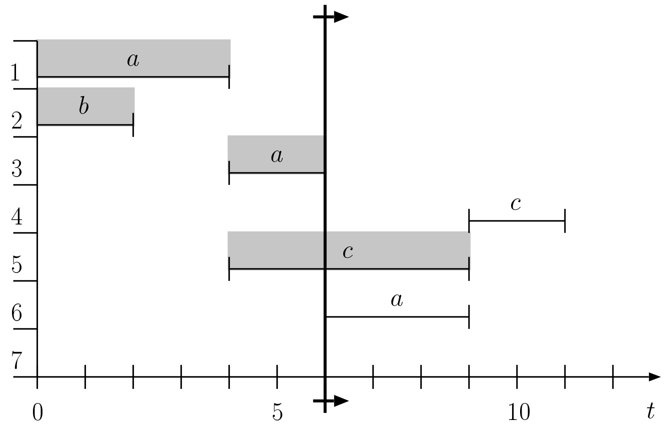


Abbildung 8.5: Ressourcenkonflikte auflösen, Einfrieren fest plazierter Aktionen

'**pll**'-Operators und es wird final als $A(\sigma)_1$ die folgende Subaktion erhalten:

(**pll** (**seq** 1),
(**seq** 2))

Das entstehende Diagramm nach dem *zweiten* Aufruf des Schritts "Einfrieren", bei dem die Aktionen 3 und 5 eingefroren werden, zeigt Abbildung 8.5. Für die neu eingefrorenen Aktionen gilt $LE(3) = LE(5) = 4$. Der Ausdruck $A(\sigma)_1$wird zu dem Ausdruck $A(\sigma)_2$ erweitert, indem die beiden eingefrorenen Aktionen 3 und 5 zu $A(\sigma)_1$hinzugefügt werden. Folgender Sachverhalt ist für das Verständnis der weiteren Ausführungen von großer Bedeutung:

> Jede Aktion i, die keine Startaktion ist, verfügt zu dem Zeitpunkt ihres Einfrierens über *mindestens eine* Aktion h, die die Eigenschaft $RE(h) = LE(i)$ erfüllt (und selbst eingefroren ist).

Die Anordnungsbeziehung zwischen den Aktionen h und i ist dabei in *genau einem* der beiden folgenden Sachverhalte begründet[1]:

- Entweder besteht zwischen h und i eine Präzedenzbeziehung oder
- h und i waren an einem Ressourcenkonflikt beteiligt und i wurde hinter h geschoben.

Wir benötigen nun die folgende Definition, um die weiteren Ausführungen zu vereinfachen.

[1] Es ist zu beachten, daß im Rahmen dieser Arbeit nur aktive Ablaufpläne betrachtet werden.

Definition 8.1.1. *Gegeben seien bei einer $\mathcal{RCPSV}$-Diagrammberechnung zwei eingefrorene Aktionen h und i mit den Eigenschaften:*

- *die Aktion h ist ein direkter Vorgänger von i*
- *es gilt $RE(h) = LE(i)$*

Wir bezeichnen h als eine fv-Aktion der Aktion i und umgekehrt i als eine fn-Aktion der Aktion h.

Die Aktion 1 ist eine *fv*-Aktion der beiden eingefrorenen Aktionen 3 und 5, da $RE(1) = LE(3) = LE(5) = 4$ gilt. Zwischen den Aktionen 1 und 5 besteht eine Präzedenzbeziehung, wohingegen die Aktion 3 im Rahmen der Auflösung eines Ressourcenkonflikts hinter die Aktion 1 geschoben wurde. Die obigen Ausführungen erlauben es uns nun die Subaktion $A(\sigma)_2$ schrittweise zusammensetzen:
Zunächst einmal betrachten wir die beiden Aktionen 3 und 5 für sich genommen. Offensichtlich gelten bezogen auf diese beiden Aktionen dieselben Bildungsgesetze wie wir sie bei $A(\sigma)_1$ kennengelernt haben und wir erhalten

$$\begin{array}{l}(\mathbf{pll}\ (\mathbf{seq}\ 3),\\ \qquad\ (\mathbf{seq}\ 5))\end{array}$$

Nun ist zu beachten, daß die Aktionen 3 und 5 *fn*-Aktion der Aktion 1 sind, was gleichbedeutend damit ist, daß beide Aktionen nach 1 durchzuführen sind. Bei der Bildung von $A(\sigma)_2$ berücksichtigen wir dies, indem wir den Subausdruck **seq** 1 aus $A(\sigma)_1$ unter Zuhilfenahme des zuletzt oben konstruierten Ausdrucks erweitern und wir erhalten in der Folge für $A(\sigma)_2$

$$\begin{array}{ll}(\mathbf{pll}\ (\mathbf{seq}\ 1, & (\mathbf{pll}\ (\mathbf{seq}\ 3),\\ & \qquad\ (\mathbf{seq}\ 5))),\\ \qquad\ (\mathbf{seq}\ 2)) & \end{array}$$

Das entstehende Diagramm nach dem *dritten* Aufruf des Schritts "Einfrieren" zeigt Abbildung 8.6. Neu eingefroren wurde jetzt die Aktion 6. Die Aktion 3 entspricht einer *fv*-Aktion der Aktion 6 (es gilt $RE(3) = LE(6)$.). Da *die Aktion 3 in Form der Aktion 6 genau eine fn-Aktion besitzt*, wird die Aktion 6 *ohne Hinzufügen eines zusätzlichen Operators* direkt hinter 3 angefügt. Die Subaktion $A(\sigma)_3$ erhält damit das folgende Aussehen:

$$\begin{array}{ll}(\mathbf{pll}\ (\mathbf{seq}\ 1, & (\mathbf{pll}\ (\mathbf{seq}\ 3, 6),\\ & \qquad\ (\mathbf{seq}\ 5))),\\ \qquad\ (\mathbf{seq}\ 2)) & \end{array}$$

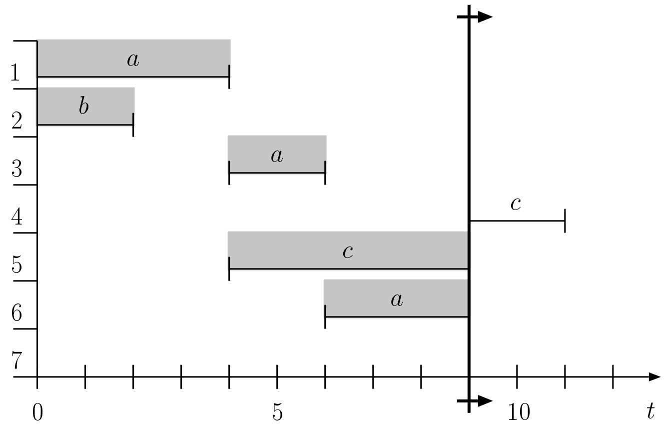


Abbildung 8.6: Einfrieren der fest plazierten Aktion 6

Nach dem *vierten* (und letzten) Aufruf des Schritts "Einfrieren" wird der Ablaufplan aus Abbildung 8.3 erhalten. Die neu eingefrorenen Aktionen sind die Aktionen 4 und 7. Die Aktion 5 entspricht einer *fv*-Aktion der Aktion 4. Eine der beiden Aktionen 5 und 6 kann als eine *fv*-Aktion der Aktion 7 ausgewählt werden. Wir wählen hier die Aktion 5 aus. Da die Aktionen 4 und 7 über keine Nachfolger verfügen, ist eine Kapselung mittels des Operators '**seq**' nicht erforderlich. Da sie ferner parallel durchgeführt werden und beide über dieselbe *fv*-Aktion verfügen, werden sie direkt mittels des Operators '**pll**' verknüpft. Als Vereinfachung kann der zu der Aktion 2 gehörige '**seq**'-Operator gelöscht werden, da die Aktion 2 keine *fn*-Aktion besitzt. Somit erhalten wir durch Erweiterung der Aktion $A(\sigma)_3$ als $A(\sigma)_4 (= A(\sigma))$ die Aktion

$$
\begin{array}{ll}
(\mathbf{pll}\ (\mathbf{seq}\ 1, & (\mathbf{pll}\ (\mathbf{seq}\ 3, 6), \\
 & \quad (\mathbf{seq}\ 5,\ (\mathbf{pll}\, 4, 7)))), \\
\quad 2) &
\end{array}
$$

Obige Aktion repräsentiert genau den Ablaufplan aus Abbildung 8.3.

Es ist offensichtlich, daß für einen beliebigen mittels der $\mathcal{RCPSV}$-Diagrammberechnung ermittelten Ablaufplan σ eine entsprechende Folge $A(\sigma)_0, A(\sigma)_2, \cdots A(\sigma)_n$ wie oben beschrieben konstruiert werden kann, wobei n der Anzahl der Aufrufe des Schritts "Einfrieren" bei der Berechnung von σ mittels der $\mathcal{RCPSV}$-Diagrammberechnung entspricht. Die Folge $A(\sigma)_0, A(\sigma)_2, \cdots A(\sigma)_n$ nennen wir im weiteren *Aktionsfolge von* σ.

Wie in Kapitel 7 erläutert, können für ein beliebiges $\mathcal{RCPSV}$-Problem alle nichtredundanten aktiven Ablaufpläne mittels der $\mathcal{RCPSV}$-Diagrammberechnung ermittelt werden. Somit kann für einen beliebigen aktiven Ablaufplan σ eines $\mathcal{RCPSV}$-Problems eine *Aktionsfolge* $A(\sigma)_0, A(\sigma)_2, \cdots A(\sigma)_n$ *von* σ konstruiert werden.

Nun wird formal bewiesen, daß $A(\sigma)_n$ genau den Ablaufplan σ repräsentiert, d.h. *ein beliebiger aktiver Ablaufplan* eines $\mathcal{RCPSV}$-Problems als *eine reduzierte Aktion* aus $\mathcal{RSV}$ dargestellt werden kann.

Lemma 8.1.1. *Gegeben sei ein beliebiger Netzplan* $\mathfrak{p} = (X, P)$ *eines* $\mathcal{RCPSV}$*-Problems.* σ *sei ein beliebiger aus* $\mathfrak{p} = (X, P)$ *herleitbarer aktiver Ablaufplan. Ferner sei* $A(\sigma)_0, A(\sigma)_2, \cdots A(\sigma)_n$ *eine Aktionsfolge von* σ*. Dann repräsentiert* $A(\sigma)_n$ *genau den Ablaufplan* σ*.*

Beweis. Wir zeigen die Hypothese durch Induktion über die $A(\sigma)_i$ der Aktionsfolge von σ.

Induktionsanfang: $A(\sigma)_1$ entspricht offensichtlich einer reduzierten Aktion aus $\mathcal{RSV}$, die den beim ersten Aufruf des Schritts "Einfrieren" endgültig berechneten Teil von σ repräsentiert.

Induktionsschritt: Gemäß Induktionsvoraussetzung entspricht $A(\sigma)_i$ einer reduzierten Aktion, die den bis zu dem i-ten Aufruf des Schritts "Einfrieren" endgültig berechneten Teil von σ darstellt. $K = \{k_1, k_2, \cdots, k_s\}$ $(s \geq 1)$ seien alle Aktionen aus X, die beim $i+1$-ten Aufruf des Schritts "Einfrieren" eingefroren werden. Für jede Aktion $k_i \in K$ existiert bei σ mindestens eine eingefrorene Aktion v (eine *fv*-Aktion von k_i), die die Eigenschaft

$$RE(v) = LE(k_i)$$

erfüllt.

$D = \{d_1, \cdots, d_j\} \subseteq K$ sei die Menge aller *fn*-Aktionen der Aktion v, d.h. für alle $d_k \in D$ gilt bei σ die Eigenschaft

$$RE(v) = LE(d_1) = LE(d_2) = \cdots = LE(d_j)$$

Nach dem im obigen Beispiel beschriebenen Bildungsgesetz wird $A(\sigma)_{i+1}$ wie folgt konstruiert:

Zunächst wird die Subaktion

$$\begin{aligned}(\textbf{pll}\ (&\textbf{seq}\ d_1),\\ &(\textbf{seq}\ d_2),\\ &\quad\vdots\\ &(\textbf{seq}\ d_j))\end{aligned}$$

gebildet. Anschließend wird diese Subaktion hinter v zu $A(\sigma)_i$ hinzugefügt. Die Semantik der Operatoren '**pll**' sowie '**seq**' ermöglicht es, bei der Erweiterung der Aktion $A(\sigma)_i$ zur Aktion $A(\sigma)_{i+1}$ für jede Aktion $d_1, d_2, \cdots, d_j$ genau denselben Durchführungszeitraum bei σ zu erzwingen. Bei allen übrigen beim $i+1$-ten

Aufruf des Schritts "Einfrieren" eingefroren Aktionen kann das oben beschriebene Bildungsgesetz genauso angewendet werden. Insgesamt wird auf diese Weise die reduzierte Aktion $A(\sigma)_{i+1}$ erhalten. Offensichtlich entspricht $A(\sigma)_{i+1}$ einer reduzierten Aktion, die den bis zu dem $i+1$-ten Aufruf des Schritts "Einfrieren" endgültig berechnete Teil von σ repräsentiert. □

Lemma 8.1.2. *Gegeben sei ein Netzplan $\mathfrak{p}= (X, P)$ eines beliebigen $\mathcal{RCPSV}$-Problems. Jeder aus $\mathfrak{p}$ herleitbare aktive Ablaufplan σ kann als eine reduzierte Aktion $A(\sigma)$ aus $\mathcal{RSV}$ dargestellt werden.*

Beweis. Gemäß Satz 7.4.2 ist für einen beliebigen Ablaufplan σ eine zugehörige Aktionsfolge $A(\sigma)_0, A(\sigma)_2, \cdots A(\sigma)_n$ konstruierbar. Gemäß Lemma 8.1.1 repräsentiert $A(\sigma)_n$ den Ablaufplan σ. □

Satz 8.1.1. *Für eine beliebige h-Aktion A aus $\mathcal{RCPSV}$ existiert eine $\mathcal{RSV}$-Aktion A' mit $A^{\mathcal{I}} = (A')^{\mathcal{I}}$, d.h. eine beliebige $\mathcal{RCPSV}$-Aktion kann als eine $\mathcal{RSV}$-Aktion repräsentiert werden.*

Beweis. Für eine beliebige $\mathcal{RCPSV}$-Aktion A können mittels der in Kapitel 7 vorgestellten Methodik alle nichtredundanten aktiven Ablaufpläne hergeleitet (berechnet) werden. Gemäß Lemma 8.1.2 kann ein beliebiger (aus einer $\mathcal{RCPSV}$-Aktion herleitbarer) Ablaufplan als eine reduzierte Aktion A_r aus $\mathcal{RSV}$ dargestellt werden. $\sigma_1, \cdots \sigma_n$ seien alle aus A herleitbaren aktiven Ablaufpläne. $A_{r1}, \cdots, A_{rn}$ seien alle reduzierten Aktionen aus $\mathcal{RSV}$, die die Ablaufpläne $\sigma_1, \cdots \sigma_n$ repräsentieren. Für die aus $A_{r1}, \cdots, A_{rn}$ konstruierbare $\mathcal{RSV}$-Aktion

$$\mathbf{xor}\, A_{r1}, \cdots, A_{rn}$$

gilt offensichtlich $A^{\mathcal{I}} = (\mathbf{xor} A_{r1}, \cdots, A_{rn})^{\mathcal{I}}$. □

Bemerkung: Zu beachten ist im Zusammenhang mit dem obigen Satz, daß der Ausdruck **xor** $A_{r1}, \cdots, A_{rn}$ vermutlich einen exponentiell größeren Platzbedarf aufweisen kann als die h-Aktion A.

8.2 Vergleich der Repräsentationsfähigkeit von $\mathcal{RCPSV}$ und $\mathcal{RSV}$

In diesem Abschnitt wird anhand eines Beispiels die Repräsentationsfähigkeit von den beiden Sprachen $\mathcal{RCPSV}$ und $\mathcal{RSV}$ verglichen. Wir betrachten das $\mathcal{RCPS}$-Problem, das in Abbildung 8.7 wiedergegeben wird. Dieses Problem kann mittels

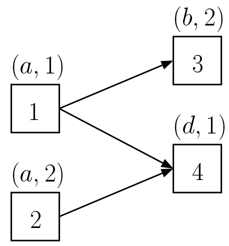


Abbildung 8.7: Ein $\mathcal{RCPS}$-Problem

der h-Aktion

$$(\mathbf{hnet}[let\, 1 = (1, a, 1), 2 = (2, a, 2), \cdots ; (1, 3), (1, 4), (2, 4)]) \tag{1}$$

dargestellt werden. Wird obige h-Aktion alternativ als Aktion aus $\mathcal{RSV}$ dargestellt, so erhalten wir den Ausdruck

$$\begin{array}{ll}(\mathbf{xor} & (\mathbf{seq}\,(1, a, 1), (\mathbf{pll}\,(3, b, 2), (\mathbf{seq}\,(2, a, 2)), (4, d, 1))), \\ & (\mathbf{seq}\,(2, a, 2), (1, a, 1), (\mathbf{pll}\,(3, b, 2), (4, d, 1))))\end{array} \tag{2}$$

Es ist offensichtlich, daß der untere Ausdruck strukturell sehr viel aufwendiger ist als der obere. Es stellt sich nun die Frage, ob dieser strukturelle Aufwand tatsächlich notwendig ist, oder ob es sich nicht nur um eine ungünstig gewählte Repräsentation des Problems handelt. Es wird nun gezeigt, daß kein kompakterer Ausdruck aus $\mathcal{RSV}$ existiert, mittels dessen sich die obige h-Aktion (1) als $\mathcal{RSV}$-Ausdruck darstellen läßt.

Wegen der induktiven Konstruktionsregel der Aktionen der Sprache $\mathcal{RSV}$ (siehe Definition 3.1.1) besitzt eine Aktion A aus $\mathcal{RSV}$ stets die besondere Eigenschaft, daß bei A strukturell kleinere Subaktionen aus $\mathcal{RSV}$ durch einen **xor**-, **seq**- bzw. **pll**-Operator verknüpft sind. Es ist nicht möglich, das $\mathcal{RCPS}$-Problem aus Abbildung 8.7 derart zu zerlegen, daß irgendwie geartete Aktionen aus $\mathcal{RSV}$ mittels eines **seq**- bzw. **pll**-Operator verknüpft werden und diese Verknüpfung in ihrer Gesamtheit das obige $\mathcal{RCPS}$-Problem repräsentiert. Ein **seq**- bzw. **pll**-Operator auf oberster Ebene scheidet damit aus. Dies soll kurz verdeutlicht werden:
Die Anzahl möglicher Zerlegungen des obigen $\mathcal{RCPS}$-Problems ist trivialerweise endlich. Unter Zerlegung verstehen wir dabei, die Aufteilung der Grundaktionen in Gruppen, die mittels eines $\mathcal{RSV}$-Ausdrucks dargestellt werden können. Bei der Zerlegung in zwei Gruppen $\{1, 2, 4\}$ und $\{3\}$ kann beispielsweise $\{1, 2, 4\}$ als der $\mathcal{RSV}$-Ausdruck

$$(\mathbf{seq}(\mathbf{pll}1, 2), 4)$$

und $\{3\}$ als eine einzelne Grundaktion 3 dargestellt werden. Es ist jedoch nicht möglich, die beiden $\mathcal{RSV}$-Ausdrücke $(\mathbf{seq}(\mathbf{pll}1, 2), 4)$ und 3 mittels eines Operators aus $\mathcal{RSV}$ derart zu verknüpfen, daß der daraus resultierende Ausdruck das

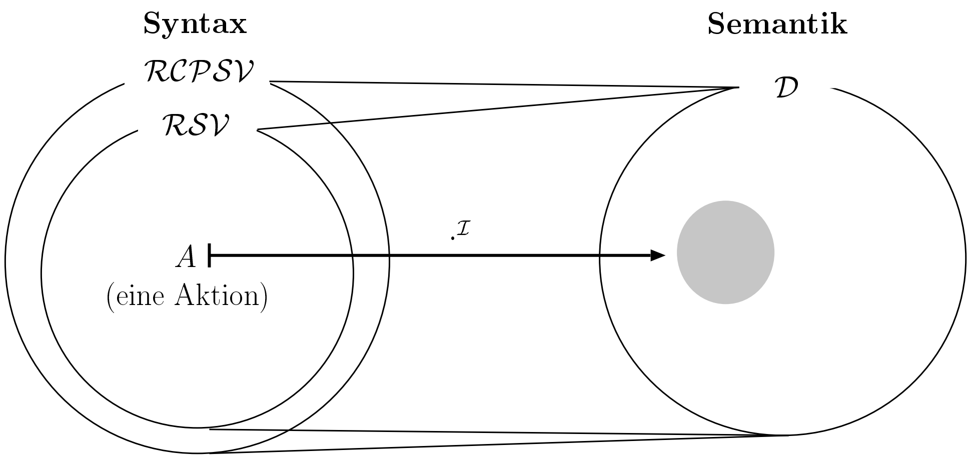


Abbildung 8.8: Die Subklassenbeziehung zwischen $\mathcal{RCPSV}$ und $\mathcal{RSV}$ bzgl. der syntaktischen Repräsentationsfähigkeit sowie die Interpretationsabbildung der beiden Sprachen

obige $\mathcal{RCPS}$-Problem darstellt. Wie bereits erwähnt, ist die Anzahl der möglichen Zerlegungen endlich und es kann geprüft werden, daß für jede Zerlegung die Darstellung des obigen Problems als ein $\mathcal{RSV}$-Ausdruck *in einer kompakten Form* nicht möglich ist.

8.3 Abschließende Bemerkungen

Die *(von der Repräsentationsfähigkeit eines Problems her* bestehende) Subklassenbeziehung zwischen $\mathcal{RSV}$ und $\mathcal{RCPSV}$ sowie die Interpretationsabbildung der beiden Sprachen zeigt Abbildung 8.9. Bei der Sprache $\mathcal{RCPSV}$ werden die beiden Operatoren '**hnet**' und '**xor**' verwendet, während bei der Sprache $\mathcal{RSV}$ die drei Operatoren '**seq**', '**pll**' und '**xor**' verwendet werden. Die in Abbildung 8.9 gezeigte Subklassenbeziehung der Sprachen $\mathcal{RSV}$ von $\mathcal{RCPSV}$ basiert darauf, daß unabhängig von der Verwendung teilweise unterschiedlicher Operatoren jeder Ausdruck aus $\mathcal{RSV}$ in einen Ausdruck aus $\mathcal{RCPSV}$ transformiert (repräsentiert) werden kann. Dieser Transformationsprozeß verfügt über eine lineare Zeitkomplexität.

Es sei an dieser Stelle nochmalig auf Abbildung 3.2 hingewiesen, wo derselbe interpretatorische Zusammenhang für die beiden Sprachen $\mathcal{RSV}$ und $\mathcal{RSVX}$ dargestellt wird. Aus beiden Abbildungen 3.2 und 8.9 kann somit die Abbildung 8.9 hergeleitet werden, aus der abgelesen werden kann, daß alle drei Sprachen $\mathcal{RSVX}$, $\mathcal{RSV}$ und $\mathcal{RCPSV}$ von der Ausdruckskraft her gleich mächtig sind. Von der syntaktischen Repräsentationsfähigkeit her bilden sie jedoch Subklassenbeziehungen.

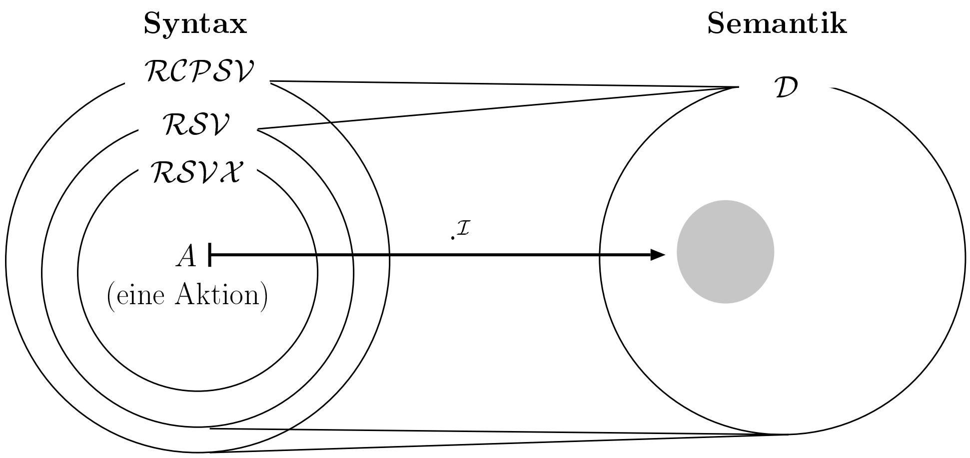


Abbildung 8.9: Die Subklassenbeziehung zwischen $\mathcal{RCPSV}$, $\mathcal{RSV}$ und $\mathcal{RSVX}$ bzgl. der syntaktischen Repräsentationsfähigkeit sowie die Interpretationsabbildung der Sprachen

Kapitel 9

Komplexität des $\mathcal{RSV}$-Problems

In Abschnitt 2.5 wurde die $\mathcal{NP}$-Vollständigkeitstheorie vorgestellt, die im Bereich der theoretischen Informatik als ein Bewertungskriterium für die zeitliche Komplexität von Problemen benutzt wird. Wie bereits erwähnt, entspricht ein $\mathcal{NP}$-vollständiges Problem einem der schwierigsten Probleme aus der Klasse $\mathcal{NP}$. Im Rahmen dieses Kapitels wird gezeigt, daß die Sprache $\mathcal{RSV}$ $\mathcal{NP}$-vollständig ist[1]. Die in diesem Kapitel benutzte Form der Beweisführung ist durch [SS76] motiviert.

9.1 Drei Varianten des $\mathcal{RSV}$-Problems

Je nachdem, was für eine Ausgabe erwartet wird, können drei verschiedene Varianten des $\mathcal{RSV}$-Problems formuliert werden. Die zwei Varianten *"Optimierungsvariante und Zahlvariante"* haben wir bereits in Kapitel 4 kennengelernt. Im Rahmen der $\mathcal{NP}$-Vollständigkeitstheorie wird eine dritte Variante benutzt, die als *Entscheidungsvariante* von Optimierungsproblemen bezeichnet wird. Im folgenden werden diese drei Varianten vertiefend betrachtet.

Ein Ausdruck $A \in \mathcal{RSV}$ spezifiziert, wie bereits in Abschnitt 2.1.4 erwähnt, ein Ablaufplanungsproblem. Durch Anwendung des $\mathcal{RSV}$-Kalküls wird ein Ausdruck $A \in \mathcal{RSV}$ in einen Ausdruck $A' \in \mathcal{RSVX}$ transformiert, bei dem alle zu A gehörigen reduzierten Aktionen (Netzpläne) durch einen einzelnen Operator '**xor**' verknüpft sind. Für eine reduzierte Aktion können mehrere Prozeßaktionen existieren, die unterschiedliche Laufzeiten benötigen. Eine zu A gehörige reduzierte Aktion, aus der eine optimale Prozeßaktion für A hergeleitet werden kann, wird *eine optimale reduzierte Aktion für A* genannt. Dies wird wie folgt formal definiert:

[1] Wie bereits in Abschnitt 2.5 erwähnt, ist die $\mathcal{NP}$-Vollständigkeit des $\mathcal{RCPS}$-Problems bereits bekannt [LRK78].

Definition 9.1.1. *In einem gegebenen Ausdruck $A \in \mathcal{RSV}$ seien n reduzierte Aktionen $A_{p_1}, A_{p_2}, \cdots, A_{p_n}$ enthalten. Wird aus A_{p_i} eine optimale Prozeßaktion für A hergeleitet, dann heißt A_{p_i} eine optimale reduzierte Aktion.*

Gegeben sei ein Ausdruck $A \in \mathcal{RSV}$ als Eingabe des $\mathcal{RSV}$-Problems. Bei der ersten Variante, die als *Optimierungsvariante* des $\mathcal{RSV}$-Problems bezeichnet wird, besteht die Aufgabe in der Berechnung der folgenden Ausgabe:

- Berechne eine optimale reduzierte Aktion sowie eine zugehörige optimale Prozeßaktion.

Bei der zweiten Variante des $\mathcal{RSV}$-Problems, die als *Zahlvariante* bezeichnet wird, soll der Wert einer optimalen Lösung berechnet werden:

- Berechne die optimale Gesamtbearbeitungszeit.

Entscheidungsvarianten, die auch Entscheidungsprobleme genannt werden, sind Funktionen mit dem Bildbereich $\{0, 1\}$; jedes Entscheidungsproblem kann somit als eine Funktion $f : \sum^* \to \{0, 1\}$ beschrieben werden. Wird ein gegebenes Eingabewort $x \in \Sigma^*$ auf den Wert 1 abgebildet, so spricht man davon, daß das *Wort x akzeptiert* wird. Wird ein Wort x auf den Wert 0 abgebildet, so spricht man davon, daß das *Wort x verworfen* wird. Als Sprache eines Entscheidungsproblems bezeichnet man die Menge aller Wörter, die auf den Wert 1 abgebildet werden. Aussagen über $\mathcal{NP}$-Vollständigkeit werden somit immer über Sprachen getroffen. Die Sprache des $\mathcal{RSV}$-Entscheidungsproblems kann wie folgt charakterisiert werden:

Notation 9.1.1. *A sei ein Ausdruck aus $\mathcal{RSV}$. Die Gesamtbearbeitungszeit einer zu A gehörigen Prozeßaktion $P_{a_{ij}}$ wird mit $\mathit{lt}(P_{a_{ij}})$ bezeichnet.*

Das $\mathcal{RSV}$-Entscheidungsproblem

Gegeben sei ein Ausdruck A aus $\mathcal{RSV}$ und eine (minimale) Zeitgrenze $t \in \mathbb{N}$. Das $\mathcal{RSV}$-Entscheidungsproblem lautet:

- Ist die optimale Gesamtbearbeitungszeit von A kleiner gleich der gegebenen Zeitgrenze t ?

Die Sprache $\mathcal{RSV}^*$

Das $\mathcal{RSV}$-Entscheidungsproblem besteht darin, zu entscheiden, ob für einen vorgegebenen Ausdruck aus $\mathcal{RSV}$ und für eine vorgegebene Zeitgrenze $t \in \mathbb{N}$ eine Prozeßaktion (ein Ablaufplan) existiert, deren Gesamtlaufzeit kleiner als die Zeitgrenze t ist.
Mit $\mathcal{RSV}^*$ wird die Sprache bezeichnet, die aus sämtlichen Paaren (X, t) mit $X \in \mathcal{RSV}$ und $t \in \mathbb{N}$ besteht, bei denen für den Ausdruck X eine Prozeßaktion $P_{a_{ij}}$ mit $\mathit{lt}(P_{a_{ij}}) \leq t$ existiert.

9.2 Das $\mathcal{RSV}$-Entscheidungsproblem ist $\mathcal{NP}$-vollständig

In diesem Abschnitt wird gezeigt, daß $\mathcal{RSV}^*$ durch eine polynomiell beschränkte nichtdeterministische Turingmaschine akzeptiert werden kann, d.h. $\mathcal{RSV}^* \in \mathcal{NP}$. Weiterhin wird gezeigt, daß die Sprache *SE (SPEZIELLE ERFÜLLBARKEIT)* auf $\mathcal{RSV}^*$ polynomiell reduzierbar ($SE \leq \mathcal{RSV}^*$) ist. Aus beidem folgt, daß das $\mathcal{RSV}$-Entscheidungsproblem $\mathcal{NP}$-vollständig ist.

Satz 9.2.1. *$\mathcal{RSV}^*$ liegt in $\mathcal{NP}$.*

Beweis. Eine polynomiell beschränkte nichtdeterministische Turingmaschine NT, die genau die Wörter aus $\mathcal{RSV}^*$ akzeptiert, kann wie folgt konstruiert werden:

NT prüft, ob die Eingabe die korrekte Kodierung eines Tupels (A, t) mit $A \in \mathcal{RSV}$ und $t \in \mathbb{N}$ darstellt. Falls die Eingabe unzulässig ist, stoppt NT und verwirft die Eingabe. Ansonsten errät NT nichtdeterministisch eine *optimale* Abarbeitungsfolge[2] (Reihenfolge) einer optimalen reduzierten Aktion der Eingabeaktion A (genauer: kodierten Eingabeaktion). Mittels der erratenen optimalen Abarbeitungsfolge wird eine optimale Prozeßaktion erzeugt. Anschließend prüft NT, ob die (in polynomieller Laufzeit) erratene Prozeßaktion die zu A gehörigen Bedingungen erfüllt. Erfüllt die Prozeßaktion die Bedingungen, ermittelt NT die zugehörige minimale Gesamtbearbeitungszeit. Falls die Gesamtbearbeitungszeit kleiner gleich t ist, dann geht NT in den akzeptierenden Zustand und stoppt; damit ist das Eingabewort (A, t) akzeptiert worden. Falls nicht, wird das Eingabewort (A, t) verworfen.
Falls (A, t) aus $\mathcal{RSV}^*$ ist, gibt es sicher eine Berechnung, die im akzeptierenden Zustand endet, da $\mathcal{RSV}^*$ aus sämtlichen Paaren $(X,\ t)$ mit $X \in \mathcal{RSV}$ und $t \in N$ besteht, für die in dem Ausdruck X eine Prozeßaktion $P_{a_{ij}}$ mit $\mathit{lt}(P_{a_{ij}}) \leq t$ existiert.

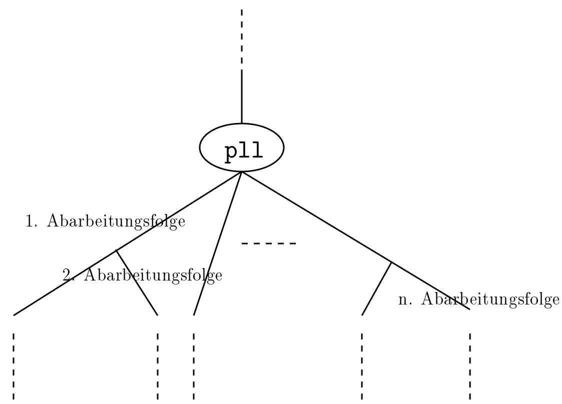
Falls umgekehrt NT nach einer Berechnung im Zustand q_f stoppt, dann existiert eine Prozeßaktion, deren Gesamtbearbeitungszeit $\leq t$ ist. □

Bemerkung: Die beim Beweis des Satzes 9.2.1 konstruierte nichtdeterministische Turingmaschine NT kann als eine Variation sowie eine Erweiterung der in Kapitel 4 vorgestellten $\mathcal{RSV}$-Diagrammberechnung angesehen werden. NT ist eine nichtdeterministische Turingmaschine, währenddessen der Algorithmus ($\mathcal{RSV}$-Diagrammberechnung) mit einer deterministischen Turingmaschine korrespondiert. Für einen als Eingabe gegebenen Ausdruck $A \in \mathcal{RSV}$ errät NT eine optimale Abarbeitungsfolge einer optimalen reduzierten Aktion. NT ermittelt dann

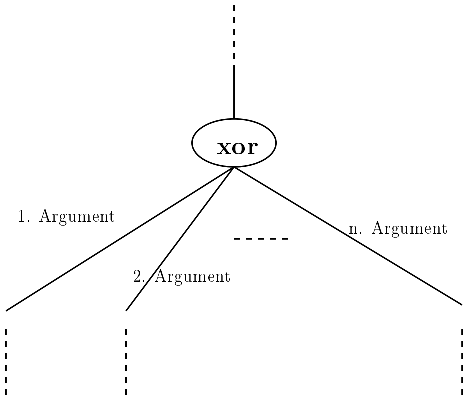
[2] Eine ausführliche Erläuterung darüber, was unter einer Abarbeitungsfolge zu verstehen ist, befindet sich in Abschnitt 4.1.

die zugehörige optimale Gesamtbearbeitungszeit, um den Wert mit der Zeitgrenze $t \in N$ zu vergleichen, währenddessen bei der $\mathcal{RSV}$-Diagrammberechnung für eine als Eingabe gegebene reduzierte Aktion alle nichtredundanten Reihenfolgen (alle nichtredundanten Prozeßaktionen) berechnet werden (siehe Abschnitte 4.5.6 und 4.5.7). Die Variation sowie die Erweiterung der $\mathcal{RSV}$-Diagrammberechnung wird im folgenden näher erläutert:

- Variation der $\mathcal{RSV}$-Diagrammberechnung:
  Das $\mathcal{RSV}$-Diagramm wird vervielfacht, immer wenn Ressourcenkonflikte entstehen. Diese Vervielfachung kann in NT durch Nichtdeterminismus erfaßt werden. Immer wenn NT auf eine mit '**pll**' verbundene Subaktion trifft, errät NT eine optimale Abarbeitungsfolge (Reihenfolge). Angenommen, für eine mit '**pll**' verbundene Subaktion existieren n verschiedene Abarbeitungsfolgen. Mindestens eine der n Abarbeitungsfolgen ist eine optimale Abarbeitungsfolge, die NT erraten kann. Die folgende Abbildung verdeutlicht den Nichtdeterminismus bzgl. des Operators **pll**:



- Erweiterung der $\mathcal{RSV}$-Diagrammberechnung:
  Die Eingaben bei der $\mathcal{RSV}$-Diagrammberechnung sind stets reduzierte Aktionen. Bei einer Aktion aus $\mathcal{RSV}$ kann zusätzlich der Operator '**xor**' vorkommen. Bei der Maschine NT kann die Berechnung für den Operator '**xor**' mittels Nichtdeterminismus beschrieben werden; NT errät das optimale Argument unter allen Argumenten (Subaktionen) des Operators '**xor**'. Angenommen, '**xor**' habe n Argumente. NT hat in diesem Moment mindestens eine der n nächsten Berechnungen zur Auswahl (Nichtdeterminismus). Die folgende Abbildung verdeutlicht den Nichtdeterminismus bzgl. des Operators '**xor**':



Satz 9.2.2. *$SE \leq \mathcal{RSV}^*$*

Vorbereitungen für den Beweis

$\mathcal{C} := C_1 \wedge C_2 \wedge \cdots \wedge C_m$ sei eine aussagenlogische Formel aus der Menge SE. C_i hat die Form $x_{i1} \vee x_{i2} \vee \cdots \vee x_{in_i}$ mit $x_{ik} \in \{p_1, \cdots, p_n, \neg p_1, \cdots, \neg p_n\}$ und $h \neq k \Rightarrow (x_{ih} \neq x_{ik}$ *und* $x_{ih} \neq \neg x_{ik})$. Desweiteren gilt $i \neq j \Rightarrow$ C_i und C_j sind verschieden $(Occ\, C_i \neq Occ\, C_j)$.

Zuordnung von Ressourcen zu jedem Literal x_{ij} aus $\mathcal{C}$

Jedem Literal x_{ij} aus $\mathcal{C}$ wird eine Menge, die aus Ressourcen besteht, zugeordnet.

Notation 9.2.1. *Die dem Literal x_{ij} zugeordnete Menge von Ressourcen bezeichnen wir als $M(x_{ij})$.*

Die Bildung der Menge $M(x_{ij})$ geschieht wie folgt:

Für jedes Literal x_{uv}, das die folgenden beiden Eigenschaften

1. $u < i$ und
2. $(\neg x_{uv}) = x_{ij}$

besitzt, wird eine neue Ressource eingeführt. Die neu eingeführte Ressource wird zu den beiden Mengen $M(x_{ij})$ und $M(x_{uv})$ hinzugefügt.

Zwischen den zugeordneten Mengen sind dann die folgenden Eigenschaften erfüllt:

$$\begin{cases} M(x_{ij}) \cap M(x_{kl}) = \phi & \text{für } x_{ij} \neq \neg x_{kl} \\ M(x_{ij}) \cap M(x_{kl}) \neq \phi,\ |M(x_{ij}) \cap M(x_{kl})| = 1 & \text{für } x_{ij} = \neg x_{kl} \end{cases}$$

Die zur Bildung der obigen Mengen benötigte Komplexität beträgt $O(n^2)$.

Zuordnung eines Ausdrucks aus $\mathcal{RSV}$ zu einem Literal x_{ij}

Definition 9.2.1. *Ein Literalausdruck für x_{ij} ($Litexp\,x_{ij}$) ist ein Ausdruck aus $\mathcal{RSV}$, der wie folgt gebildet wird:*

1. *Ist $M(x_{ij})$ leer, dann wird eine neue Ressource r, die zu keiner Menge gehört, eingeführt. Die Grundaktion $(x_{ij}, C_i, r)(r, 1)$ wird dem Literal x_{ij} als Literalausdruck zugeordnet. Hierbei entspricht (x_{ij}, C_i, r) dem Prädikat der Grundaktion. Zur Erfüllung der Grundaktion wird die Ressource r eine Zeiteinheit lang benötigt.*
   ($Litexp\,x_{ij} := (x_{ij}, C_i, r)(r, 1)$).
2. *Besteht $M(x_{ij})$ aus den u Ressourcen $s_1, \cdots, s_u$, wird für jede Ressource s_h aus $\{s_1, \cdots, s_u\}$ eine Grundaktion $(x_{ij}, C_i, s_h)(s_h, 1)$ gebildet. Dabei entspricht (x_{ij}, C_i, s_h) dem Prädikat, s_h der beanspruchten Ressource und 1 der benötigten zeitlichen Dauer der Grundaktion $(x_{ij}, C_i, s_h)(s_h, 1)$. Die derart erzeugten Grundaktionen $(x_{ij}, C_i, s_1)(s_1, 1)$, $(x_{ij}, C_i, s_2)(s_2, 1)$, $\cdots$ und $(x_{ij}, C_i, s_u)(s_u, 1)$ werden durch den Operator '**pll**' verbunden. Der durch '**pll**' verbundene Ausdruck wird dem Literal x_{ij} als Literalausdruck zugeordnet.*
   ($Litexp\,x_{ij} := pll\,(x_{ij}, C_i, s_1)(s_1, 1), (x_{ij}, C_i, s_2)(s_2, 1), \cdots, (x_{ij}, C_i, s_u)(s_u, 1)$)

Die zur Bildung der Literalausdrücke erforderliche Komplexität beträgt $O(n)$.

*Zuordnung eines mit '**xor**' verbundenen Ausdrucks aus $\mathcal{RSV}$ zu C_i ($i = 1, \cdots m$)*

Zu jedem C_i wird ein Ausdruck aus $\mathcal{RSV}$, den wir Okkurenzausdruck von C_i ($Occexp\,C_i$) nennen, wie folgt zugeordnet:

Definition 9.2.2. *Es sei $Occ\,C_i = \{x_{i1}, x_{i2}, \cdots, x_{in_i}\}$; ferner seien $Litexp\,x_{i1}$, $Litexp\,x_{i2}$, $\cdots$, $Litexp\,x_{in_i}$ gegeben. Occexp C_i ist ein Ausdruck aus $\mathcal{RSV}$, in dem alle zu C_i gehörigen Literalausdrücke $Litexp\,x_{i1}$, $Litexp\,x_{i2}$, $\cdots$, $Litexp\,x_{in_i}$ durch den Operator '**xor**' verbunden sind.*
($Occexp\,C_i := \mathbf{xor}\ Litexp\,x_{i1}, \cdots, Litexp\,x_{in_i}$)

Die zur Bildung der Okkurenzausdrücke erforderliche Komplexität beträgt $O(n)$.

Beweis. Wir zeigen die polynomielle Reduzierbarkeit $SE \leq \mathcal{RSV}^*$ mit der folgenden Zuordnung:

$C_1 \wedge C_2 \wedge \cdots \wedge C_m \longmapsto_f ((\mathbf{pll}\,Occexp\,C_1,\ Occexp\,C_2, \cdots, Occexp\,C_m),\ 1)$

Es wird gezeigt, daß $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ genau dann erfüllbar ist, wenn der zugeordnete Ausdruck $((\mathbf{pll}\,Occexp\,C_1,\ Occexp\,C_2, \cdots, Occexp\,C_m),\ 1)$ in $\mathcal{RSV}^*$ liegt:
Es sei $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ erfüllbar. Dann existiert eine Belegung $v : \{p_1, \cdots p_n\} \rightarrow$

$\{0,1\}$ mit $v(C_1 \wedge \cdots \wedge C_m) = 1$, d.h. es existiert für jedes $i \leq m$ ein $x_{ij_i} \in Occ\, C_i$ mit $v(x_{ij_i}) = 1$. Dies seien $x_{1j_1}, \cdots, x_{mj_m}$. Wegen $v(x_{hj_h}) = 1 = v(x_{kj_k}) \neq v(\neg x_{kj_k})$ für alle $1 \leq h, k \leq m$ gilt: $h \neq k \Rightarrow x_{hj_h} \neq \neg x_{kj_k}$. Daraus folgt: $M(x_{1j_1}) \cap M(x_{2j_2}) \cap \cdots \cap M(x_{mj_m}) = \phi$. Somit existiert eine reduzierte Aktion

$$(\mathbf{pll}\,(Litexp\, x_{1j_1}), \cdots, (Litexp\, x_{mj_m})),$$

die ressourcenkonfliktfrei ist, d.h. die reduzierte Aktion ist eine Prozeßaktion mit der Gesamtbearbeitungszeit 1.

Sei umgekehrt $(\mathbf{pll}\, Occexp\, C_1,\ Occexp\, C_2, \cdots, Occexp\, C_m)$ ein Ausdruck in $\mathcal{RSV}$, in dem eine Prozeßaktion mit der Gesamtbearbeitungszeit ≤ 1 existiert. Dann existiert ein $Litexp\, x_{ij_i}$ in jedem $Occexp\, C_i$, so daß die Aktion $(\mathbf{pll}\, Litexp\, x_{1j_1}, Litexp\, x_{2j_2}, \cdots, Litexp\, x_{mj_m})$ ressourcenkonfliktfrei ist. Daraus folgt $M(x_{1j_1}) \cap \cdots \cap M(x_{mj_m}) = \phi$. Dann gilt: $h \neq k \Rightarrow x_{hj_h} \neq \neg x_{kj_k}$ für alle $1 \leq h, k \leq m$. Somit haben wir eine Belegung

$$v(p_l) := \begin{cases} 1 & \text{wenn } p_l \in \{x_{1j_1}, \cdots, x_{mj_m}\} \\ 0 & \text{sonst} \end{cases}$$

für alle $l \leq n$. Diese Belegung nimmt den Wert 1 auf $x_{1j_1}, \cdots, x_{mj_m}$ an, da für alle x_{ij_i} entweder $x_{ij_i} = p_h$ oder $x_{ij_i} = \neg p_h$ für ein gewisses $h \leq n$ ist. Ist $x_{ij_i} = p_h$, dann gilt $v(x_{ij_i}) = v(p_h) = 1$. Ist $x_{ij_i} = \neg p_h$, dann gilt $p_h \notin \{x_{1j_1}, \cdots, x_{mj_m}\}$ und $v(p_h) = 0$. Somit gilt ebenfalls $v(x_{ij_i}) = v(\neg p_h) = 1$. Für alle $i = 1, \cdots, m$ existiert also ein x_{ij_i} mit $v(x_{ij_i}) = 1$ und $x_{ij_i} \in Occ\, C_i$, d.h. $v(C_1 \wedge \cdots \wedge C_m) = 1$.

Daß $f \in \pi$ ist, zeigen die ermittelten Komplexitätsberechnungen. □

Das folgende Beispiel verdeutlicht die polynomiellen Zuordnungen bzgl. Satz 9.2.2.

Beispiel 9.2.1. Als Eingabe sei die folgende Formel in konjunktiver Normalform gegeben:

$$(a \vee \neg b \vee c) \wedge (\neg a \vee b \vee \neg c) \wedge (\neg a \vee \neg b \vee c)$$

Zuordnung einer Menge von Ressourcen zu jedem Literal; Komplexität: $O(n^2)$

Okkurenzmenge (Ok)	Literal	die zugeordnete Menge (d. z. M.)
C_1	a	$\{r_1, r_4\}$
	$\neg b$	$\{r_2\}$
	c	$\{r_3\}$
C_2	$\neg a$	$\{r_1\}$
	b	$\{r_2, r_5\}$
	$\neg c$	$\{r_3, r_6\}$
C_3	$\neg a$	$\{r_4\}$
	$\neg b$	$\{r_5\}$
	c	$\{r_6\}$

Zuordnung eines Literalausdrucks zu jedem Literal

In der folgenden Tabelle wird jedem Literal ein Literalausdruck zugeordnet; Komplexität: $O(n)$:

Ok	Literal	d. z. M.	Literalausdruck
	a	$\{r_1, r_4\}$	$(\mathbf{pll}(a, C_1, r_1)(r_1, 1), (a, C_1, r_4)(r_4, 1))$
C_1	$\neg b$	$\{r_2\}$	$(\neg b, C_1, r_2)(r_2, 1)$
	c	$\{r_3\}$	$(c, C_1, r_3)(r_3, 1)$
	$\neg a$	$\{r_1\}$	$(\neg a, C_2, r_1)(r_1, 1)$
C_2	b	$\{r_2, r_5\}$	$(\mathbf{pll}(b, C_2, r_2)(r_2, 1),\ (b, C_2, r_5)(r_5, 1))$
	$\neg c$	$\{r_3, r_6\}$	$(\mathbf{pll}(\neg c, C_2, r_3)(r_3, 1),\ (\neg c, C_2, r_6)(r_6, 1))$
	$\neg a$	$\{r_4\}$	$(\neg a, C_3, r_4)(r_4, 1)$
C_3	$\neg b$	$\{r_5\}$	$(\neg b, C_3, r_5)(r_5, 1)$
	c	$\{r_6\}$	$(c, C_3, r_6)(r_6, 1)$

Zuordnung des entsprechenden Okkurenzausdrucks zu jeder Disjunktion

Alle zu der Okkurenzmenge C_i gehörigen Literalausdrücke werden durch den Operator '**xor**' verknüpft. Dies zeigt die folgende Tabelle; Komplexität: $O(n)$:

Ok	Literalausdrücke	Okkurenzausdruck
C_1	$\{(\mathbf{pll}(a, C_1, r_1)(r_1, 1), (a, C_1, r_4)(r_4, 1)),$ $(\neg b, C_1, r_2)(r_2, 1),$ $(c, C_1, r_3)(r_3, 1)\}$	$\mathbf{xor}\ (\ \mathbf{pll}(a, C_1, r_1)(r_1, 1), (a, C_1, r_4)(r_4, 1)),$ $(\neg b, C_1, r_2)(r_2, 1),$ $(c, C_1, r_3)(r_3, 1)$
C_2	$\{(\neg a, C_2, r_1)(r_1, 1),$ $(\mathbf{pll}(b, C_2, r_2)(r_2, 1),\ (b, C_2, r_5)(r_5, 1)),$ $(\mathbf{pll}(\neg c, C_2, r_3)(r_3, 1),\ (\neg c, C_2, r_6)(r_6, 1))\}$	$\mathbf{xor}\ (\neg a, C_2, r_1)(r_1, 1),$ $(\mathbf{pll}(b, C_2, r_2)(r_2, 1),\ (b, C_2, r_5)(r_5, 1)),$ $(\mathbf{pll}(\neg c, C_2, r_3)(r_3, 1),\ (\neg c, C_2, r_6)(r_6, 1))$
C_3	$\{(\neg a, C_3, r_4)(r_4, 1),$ $(\neg b, C_3, r_5)(r_5, 1),$ $(c, C_3, r_6)(r_6, 1)\}$	$\mathbf{xor}\ (\neg a, C_3, r_4)(r_4, 1),$ $(\neg b, C_3, r_5)(r_5, 1),$ $(c, C_3, r_6)(r_6, 1)$

Der zu der Okkurenzmenge C_i gehörige Okkurenzausdruck wird als $Occexp\,C_i$ bezeichnet. Alle Okkurenzausdrücke $Occexp\,C_i (i = 1, 2, 3)$ werden durch den Operator '**pll**' verknüpft. Für die als Eingabe gegebene Formel sieht die polynomielle Zuordnung f wie folgt aus:

$(a \vee \neg b \vee c) \wedge (\neg a \vee b \vee \neg c) \wedge (\neg a \vee \neg b \vee c) \longmapsto_f$ $((\mathbf{pll}\,OccexpC_1, OccexpC_2, OccexpC_3), 1)$

9.3 Die Übertragbarkeit der $\mathcal{NP}$-Vollständigkeit auf die Optimierungsvariante des $\mathcal{RSV}$-Problems

Im letzten Abschnitt wurde gezeigt, daß das $\mathcal{RSV}$-Entscheidungsproblem $\mathcal{NP}$-vollständig ist. Wie schon erwähnt, ist die $\mathcal{NP}$-Vollständigkeitstheorie ein Bewertungsmodell für die zeitliche Komplexität von Entscheidungsproblemen. Für viele praxisbezogenen Optimierungsprobleme [Weg93], [Weg96] reicht es aus, sich mit der Entscheidungsvariante zu beschäftigen, da die Optimierungsvariante (bzw. Zahlvariante) *genau dann* polynomiell (effizient) berechenbar ist, *wenn* dies die Entscheidungsvariante ist. Im folgenden wird gezeigt, daß dies auch für das $\mathcal{RSV}$-Problem gilt:

Satz 9.3.1. *Die Optimierungsvariante des $\mathcal{RSV}$-Problems ist genau dann effizient lösbar, d.h. in $\mathcal{P}$, wenn die Zahlvariante des $\mathcal{RSV}$-Problems effizient lösbar ist.*

Beweis. "$\Rightarrow$"
Angenommen die Optimierungsvariante sei effizient lösbar. Dann ist auch die Zahlvariante effizient lösbar, da die Gesamtbearbeitungszeit für eine optimale Prozeßaktion effizient berechenbar ist.

"$\Leftarrow$"
Angenommen die Zahlvariante sei effizient mittels eines Algorithmus G lösbar. Zuerst kann dann eine optimale reduzierte Aktion einer beliebigen Eingabeaktion wie folgt effizient bestimmt werden:
Um die optimale Gesamtbearbeitungszeit für die Eingabeaktion zu berechnen, wird der Algorithmus G einmal aufgerufen. Die berechnete optimale Gesamtbearbeitungszeit sei α. Für einen Operator '**xor**' kann ein zeitlich optimales Argument (Subaktion) unter allen Argumenten des Operators '**xor**' wie folgt bestimmt werden:
Das erste Argument des Operators '**xor**' wird entfernt; Anschließend wird durch Aufruf des Algorithmus G entschieden, ob die optimale Gesamtbearbeitungszeit immer noch der Zeit α entspricht. Wenn ja, wird das zweite Argument gelöscht und wiederum der Algorithmus G aufgerufen. Wenn nein, dann ist das Argument das gesuchte Argument, das zu einer optimalen reduzierten Aktion gehört. Es muß mindestens ein Argument existieren, bei dessen Entfernung eine verlängerte optimale Gesamtbearbeitungszeit als α berechnet wird. Ist ein solches Argument gefunden, werden außer dem Argument alle sonstigen Argumente zusammen mit dem Operator '**xor**' gelöscht.
Für jeden weiter vorhandenen Operator '**xor**' kann ein zu einer optimalen reduzierten Aktion gehöriges Argument in derselben Art und Weise bestimmt werden.

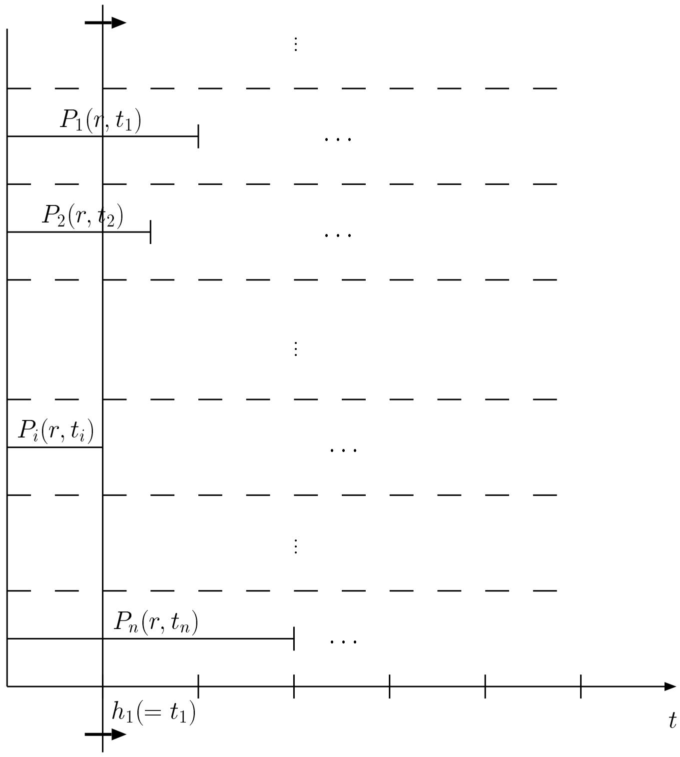


Abbildung 9.1: Ein $\mathcal{RSV}$-Diagramm mit $t(H) = h_1$

Schließlich wird eine optimale reduzierte Aktion erhalten. Die Anzahl der Aufrufe des Algorithmus G ist mit Sicherheit kleiner als die Eingabelänge, somit kann eine optimale reduzierte Aktion effizient berechnet werden.
Nun muß noch gezeigt werden, daß für eine beliebige **pll**-Aktion eine zugehörige optimale Prozeßaktion (ein zugehöriger optimaler Ablaufplan) effizient berechenbar ist.
Gegeben sei eine beliebige mit '**pll**' verbundene reduzierte Aktion A. Um die optimale Gesamtbearbeitungszeit für die Aktion A zu berechnen, wird der Algorithmus G einmal aufgerufen. Die berechnete optimale Gesamtbearbeitungszeit sei β. Zunächst wird A mittels eines $\mathcal{RSV}$-Diagramms repräsentiert (siehe Abschnitt 4.2). Als erstes springt die Scan-line zum nächsten Haltepunkt H mit $t(H) = h_1$. Für jede h_1-Zeit Konfliktressource wird die bei einer optimalen Prozeßaktion zuerst durchzuführende Grundaktion bestimmt; r sei hierbei eine h_1-Zeit Konfliktressource (siehe Definition 4.3.3 aus Kapitel 4). Desweiteren seien $P_1(r, t_1), P_2(r, t_2), \cdots P_n(r, t_n)$ alle (h_1, r)-Konfliktgrundaktionen (siehe Abbildung 9.1). Die Grundaktion, die bei einer optimalen Prozeßaktion zuerst durchzuführen ist, wird wie folgt bestimmt:
$P_1(r, t_1)$ bleibt an der ursprünglichen Stelle stehen. Alle sonstigen P_i's werden nach rechts geschoben, so daß die linken Endpunkte von den sonstigen Grundaktionen und der rechte Endpunkt von $P_1(r_1, t_1)$ übereinstimmen. Die geleer-

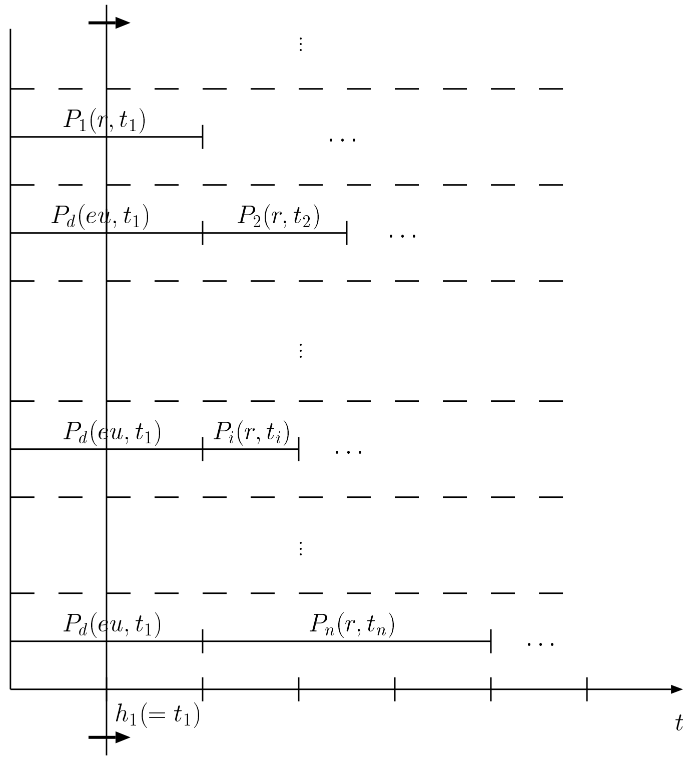


Abbildung 9.2: $P_1(r, t_1)$ als die zuerst durchzuführende Grundaktion

ten Stellen werden durch eine Pseudo-Grundaktion ersetzt. Die Gesamtbearbeitungszeit der Pseudo-Grundaktion beträgt t_1 Zeiteinheiten (siehe Abbildung 9.2). (Statt Verwendung von Pseudo-Grundaktionen können hierbei auch Grundaktionen eingesetzt werden, die eine neue verschiedene Ressource beanspruchen, die bei der Eingabeaktion nie beansprucht werden.). Der Algorithmus G wird für die veränderte Aktion aufgerufen. Ist die optimale Gesamtbearbeitungszeit gleich β, dann ist $P_1(r_1, t_1)$ die zuerst durchzuführende Grundaktion. Ist dies nicht der Fall, werden alle eingesetzten Pseudogrundaktionen gelöscht und alle Grundaktionen in den ursprünglichen Zustand zurückgebracht. $P_2(r, t_2)$ bleibt an der ursprünglichen Stelle stehen. Alle sonstigen P_i's werden von ihrer ursprünglichen Stelle t_2 Zeiteinheiten nach rechts verschoben. Die geleerten Stellen werden durch eine Pseudo-Grundaktion gefüllt. Der Algorithmus G wird für diese veränderte Aktion aufgerufen. Ist die optimale Gesamtbearbeitungszeit gleich β, dann ist $P_2(r, t_2)$ die zuerst zu bearbeitende Grundaktion. Ist dies nicht der Fall, wird auf die übrigen Grundaktionen P_is derselbe Prozeß wiederholt angewendet, bis die zuerst zu bearbeitende Grundaktion gefunden wird. Mindestens eine solche Grundaktion muß existieren (Satz 4.5.1 aus Kapitel 4.5).
Wenn für jede h_1-Zeit Konfliktressource bei einer optimalen Prozeßaktion die zuerst zu bearbeitende Grundaktion bestimmt worden ist, springt man mit der Scan-line zu dem nächsten Haltepunkt H mit $t(H) = h_2$. Falls h_2-Zeit Kon-

fliktressourcen existieren, werden die zuerst zu bearbeitenden Grundaktionen in derselben Art und Weise bestimmt. Alle sonstigen Ressourcenkonflikte können durch rekursive Anwendung des zuletzt beschriebenen Prozesses aufgelöst werden. Ferner ist die so erzeugte ressourcenkonfliktfreie Prozeßvariante eine optimale Prozeßaktion, da dessen optimale Gesamtbearbeitungszeit β entspricht. Die Anzahl der Aufrufe von Zahlvariante ist gleich $O(n^2)$. Mit der Zahlvariante ist auch die Optimierungsvariante effizient lösbar. □

Satz 9.3.2. *Die Zahlvariante des $\mathcal{RSV}$-Problems ist genau dann effizient lösbar, wenn das $\mathcal{RSV}$-Entscheidungsproblem effizient lösbar ist.*

Beweis. “$\Rightarrow$”
Angenommen die Zahlvariante sei effizient lösbar. Dann ist auch die Entscheidungsvariante effizient lösbar, da die optimale Gesamtbearbeitungszeit nur zusätzlich mit der zu der Zahlvariante gehörigen Zeitgrenze verglichen werden muß.

“$\Leftarrow$”
Angenommen die Entscheidungsvariante sei effizient mittels eines Algorithmus G lösbar. Für eine beliebige Eingabeaktion aus $\mathcal{RSV}$ werden die Laufzeiten aller Grundaktionen addiert, die in der Eingabeaktion vorkommen. Offensichtlich ist diese Summe s eine obere Schranke für die optimale Gesamtbearbeitungszeit. Nun wird der Algorithmus G für die Zeitgrenze $[s/2]$ aufgerufen, wobei $[s/2]$ die größte natürliche Zahl ist, die kleiner gleich wie $[s/2]$ sein muß. Bei Antwort ja wird die Zeitgrenze auf $[s/4]$ gesetzt, sonst auf $[3s/4]$, usw. Insgesamt reichen ca. $\log s$ Aufrufe des Algorithmus G für das $\mathcal{RSV}$-Problem, um die Zahlvariante zu lösen. $\log s$ ist kleiner als die Eingabelänge, da alle Laufzeiten von Grundaktionen der Eingabeaktion zur Eingabe gehören, also ist mit der Entscheidungsvariante auch die Zahlvariante effizient lösbar. □

Kapitel 10

Zusammenfassung und Ausblick

Die terminologischen Schedulingsprachen $\mathcal{RSV}$ sowie $\mathcal{RCPSV}$ wurden vorgestellt, unter deren Zuhilfenahme das $\mathcal{RSV}$- bzw. $\mathcal{RCPSV}$-Problem mittels bisher nicht in der Literatur vorgestellter Methodiken dargestellt und gelöst werden kann. Diese Sprachen sind logische Sprachen, d.h. syntaktische Elemente (Ausdrücke) erhalten durch die Definition einer Semantik eine Bedeutung (eine Interpretation). Die Bedeutung eines Ausdrucks, der ein $\mathcal{RSV}$- bzw. ein $\mathcal{RCPSV}$-Problem repräsentiert, entspricht der Menge aller aus dem Ausdruck herleitbaren aktiven Ablaufpläne. Mittels dieser Semantik kann auf den Sprachen $\mathcal{RSV}$ sowie $\mathcal{RCPSV}$ jeweils ein Kalkül definiert werden, mit dem jeder Ausdruck aus $\mathcal{RSV}$ bzw. $\mathcal{RCPSV}$ stets in einen *semantisch äquivalenten normalisierten* Ausdruck transformiert werden kann. Semantische Äquivalenz zweier Ausdrücke bedeutet dabei, daß die Menge aller aus den beiden Ausdrücken herleitbaren aktiven Ablaufpläne identisch ist. Aus der semantischen Äquivalenz ergibt sich, daß ein Ablaufplan für einen Ausdruck genau dann optimal ist, wenn er für einen *herleitbaren semantisch äquivalenten normalisierten* Ausdruck optimal ist.
In einem *normalisierten* Ausdruck sind alle zugehörigen *nichtredundanten* Netzpläne eines $\mathcal{RSV}$- bzw. eines $\mathcal{RCPSV}$-Problems einzeln getrennt beschrieben. Diese strukturelle Einfachheit erlaubt die vereinfachte Entwicklung eines Lösungsalgorithmus. Für das $\mathcal{RSV}$- sowie $\mathcal{RCPSV}$-Problem wurde jeweils ein Lösungsalgorithmus entwickelt, mit dem für einen beliebigen Netzplan eines $\mathcal{RSV}$- bzw. eines $\mathcal{RCPSV}$-Problems alle *nichtredundanten (optimalen)* Ablaufpläne berechnet werden können. Die Berechnungsvorgänge bei den Algorithmen konnten mittels Diagrammen auf Basis eines Scan-line-Prinzips dargestellt werden. Unter Zuhilfenahme dieses Scan-line-Prinzips konnten zudem entstehende Ressourcenkonflikte aufgelöst werden.

Für jeden eingeführten Algorithmus wurde gezeigt, wie der Algorithmus in der funktionalen Programmiersprache Haskell implementiert werden kann. Desweiteren wurde für jeden vorgestellten Algorithmus ein Korrektheitsbeweis geführt, bei dem jeweils formal bewiesen wurde, daß für einen beliebigen Netzplan eines $\mathcal{RSV}$- bzw. $\mathcal{RCPSV}$-Problems alle *nichtredundanten* Ablaufpläne berechnet werden.

Die Schedulingsprache $\mathcal{RSV}$ bildet von der syntaktischen Repräsentationsfähigkeit her eine echte Subklasse der Schedulingsprache $\mathcal{RCPSV}$. Die beiden Sprachen $\mathcal{RSV}$ und $\mathcal{RCPSV}$ sind von der Ausdruckskraft her jedoch gleich mächtig. Die Gleichmächtigkeit der beiden Sprachen $\mathcal{RSV}$ und $\mathcal{RCPSV}$ wurde formal bewiesen.

Abschließend wurde die Komplexität der Ablaufplanungsprobleme untersucht und gezeigt, daß das $\mathcal{RSV}$-Problem ein $\mathcal{NP}$-vollständiges Problem ist.

Um ein $\mathcal{RSV}$- bzw. ein $\mathcal{RCPSV}$-Problem zu lösen, haben wir eine Methodik verwendet, bei der ein $\mathcal{RSV}$- bzw. ein $\mathcal{RCPSV}$-Ausdruck zunächst normalisiert wird, d.h. alle in dem Ausdruck enthaltenen nichtredundanten Netzpläne hergeleitet werden. Auf jeden einzelnen so hergeleiteten Netzplan eines $\mathcal{RSV}$- bzw. eines $\mathcal{RCPSV}$-Problems wurde anschließend der Lösungsalgorithmus zur Berechnung aller nichtredundanten optimalen Ablaufpläne angewendet. Eine alternative Methodik wäre für ein $\mathcal{RSV}$- bzw. ein $\mathcal{RCPSV}$-Problem alle optimalen Ablaufpläne ohne Beschreiten des Normalisierungsprozesses direkt zu berechnen. Die folgenden Fragestellungen wären dabei zu untersuchen:

- Wie ist ein Algorithmus aufgebaut, der das $\mathcal{RSV}$- bzw. $\mathcal{RCPSV}$-Problem ohne Beschreiten des Normalisierungsprozesses direkt löst ?
- Welches Maß an zeitlichem Rechenaufwand wird bei solchen Modellen bei praxisrelevanten Problemen benötigt und welche Vor- oder Nachteile ergeben sich gegenüber dem in dieser Arbeit vorgestellten Scan-line-Prinzip ?

Eine weitere untersuchungswerte Fragestellung wäre, wie die in dieser Arbeit vorgestellten Algorithmen optimiert werden könnten. Beispielsweise können bei der Ermittlung aller optimalen Ablaufpläne solche Ablaufpläne, die offensichtlich nicht mehr optimal sein können, vor der Berechnung aller nichtredundanten Ablaufpläne ausgeschlossen werden.

Bei der vorliegenden Arbeit wurde alleinig das Zielkriterium der Minimierung der Gesamtbearbeitungszeit betrachtet. Weitere Zielsetzungen, die betrachtet werden könnten, sind z. B.

- Kostenminimierung,
- Ermöglichung flexibler Umplanung (Eine Umplanung sollte schnell und einfach erstellbar sein, um auf sich ständig ändernde Umfeldsituationen sowie neue Anforderungen flexibel reagieren zu können.),

- Minimierung der Gesamtbearbeitungszeit bzw. der Kosten unter Einhaltung zusätzlicher Bedingungen (beispielsweise der Nichtverfügbarkeit von Ressourcen in bestimmten Zeitabschnitten).

Fragestellungen, die sich daraus ergeben, wären:

- Können die in der vorliegenden Arbeit vorgestellten Schedulingsprachen derart erweitert werden, so daß die Verfolgung der oben aufgeführten Zielsetzungen ermöglicht wird ?
- Wenn ja, welches Aussehen weisen die zugehörigen Algorithmen auf ?

Die in dieser Arbeit vorgestellten Modelle könnten zu einem terminologischen Schedulingsystem erweitert werden, das über eine benutzerfreundliche Schnittstelle zur Repräsentation der Ablaufplanungsprobleme verfügt und die Ermittlung optimaler Ablaufpläne im Bezug auf alle praxisrelevanten Zielsetzungen erlaubt.

Kapitel 11

Anhang-Programme

In diesem Kapitel werden die in den Abschnitten 4.4 und 7.3 vorgestellten Implementierungen[1] vervollständigt. Zu jedem Programm (Implementation) werden jeweils experimentelle Ergebnisse anhand einiger Beispiele vorgestellt.

Nähere Erläuterungen über die Arbeitweisen sowie die Algorithmen hinter den Implementationen befinden sich in den Abschnitten 4.4 und 7.3.

11.1 Die $\mathcal{RSV}$-Diagrammberechnung

Es ist zu beachten, daß alle Namen von Grundaktionen unterschiedlich zu wählen sind.

11.1.1 Programm

Einige zusätzliche Standardfunktionen zur Arbeit mit Listen werden eingeladen.

```
>import List
```

Definition der Datenstruktur zur Darstellung von Aktionen

```
>type Name     = [Char]
>type Resource = Char
>data
>    Action =   GroundAction Name Resource Int
>             | SEQ          [Action]
>             | PLL          [Action]
```

[1] In den Abschnitten 4.4 und 7.3 wurde jeweils eine Implementierung der Algorithmen aus den Abschnitten 4.3 und 7.2 in der funktionalen Programmiersprachen "HASKELL" vorgestellt.

Ermöglichung des Druckens einer Ausprägung des Datentyps Aktion

```
>instance Show Action where
>  showsPrec _ = showString.toString
>    where
>     toString (GroundAction n r l)
>       = n ++ "(" ++ r:"," ++ (show l) ++ ")"
>     toString (SEQ xs)
>       = "(SEQ" ++ concat (map (\s -> ' ':s) (map toString xs)) ++ ")"
>     toString (PLL xs)
>       = "(pll" ++ concat (map (\s -> ' ':s) (map toString xs)) ++ ")"


>data
>    Exist a =   NOTHING
>              | SOMETHING a


>instance Show a => Show (Exist a) where
>    showsPrec _ = showString.toString
>        where toString NOTHING       = "NOTHING"
>              toString (SOMETHING a) = show a


>isSomething NOTHING = False
>isSomething _       = True
```

Hilfsfunktionen

```
>minimum0 [] = 0
>minimum0 ts = minimum ts


>maximum0 [] = 0
>maximum0 ts = maximum ts


>fst3 (n, _, _) = n
>snd3 (_, r, _) = r
```

```
>thd3 (_, _, e) = e


>getParallel :: Action -> [(Name, Resource, Int)]
>getParallel (GroundAction n r l) = [(n, r, l)]
>getParallel (SEQ (x:xs))         = (getParallel x)
>getParallel (PLL xs    )         = concat (map getParallel xs)


>fRemove ps c =  ps \\ [x| x <- ps, (snd3 x) == (fst c)]


>delGAExt :: Name -> (Exist Action) -> (Exist Action)
>delGAExt _ (NOTHING)     = NOTHING
>delGAExt n (SOMETHING a) = delGA n a

>delGA :: Name -> Action -> (Exist Action)
>delGA n (GroundAction n' r' l')
>        | (n == n') = NOTHING
>        | otherwise = SOMETHING (GroundAction n' r' l')
>delGA n a = case a of
>                SEQ xs -> construct xs (\ys -> SEQ ys)
>                PLL xs -> construct xs (\ys -> PLL ys)
>            where
>              construct xs g = case a' of
>                                 []  -> NOTHING
>                                 [x] -> SOMETHING x
>                                 xs  -> SOMETHING (g xs)
>                        where
>                          a' = foldl f [] (map (delGA n) xs)
>                               where f a (SOMETHING x) = a ++ [x]
>                                     f a _             = a


>rcombine [] = []
>rcombine (xs:xxs)
>    = case xxs of
>        [] -> ts
>        _  -> concat (map (\z -> (map (\x -> x ++ z) ts))
>                      (rcombine xxs)
>                      )
>             where
>                ts = (transpose [xs])
```

```
>type WorkingTime = (Name, Resource, Int, Int)
>type WorkingInfo = [WorkingTime]

>getName     (n, _, _, _) = n

>getResource (_, r, _, _) = r

>getStartTime  (_, _, t, _) = t

>getEndTime  (_, _, _, t) = t


>addAction :: Int -> WorkingInfo -> (Name, Resource, Int)
>                                -> WorkingInfo
>addAction t as (n, r, l)
>    | (elem n (map getName as)) = as
>    | otherwise                 = (n, r, t, t + l):as


>nextScanline l as
>    = minimum0 (filter (\t -> l < t) (map getEndTime as))


>confResr :: [Resource] -> WorkingInfo -> [Resource]
>confResr [] _    = []
>confResr (r:rs) bs
>   | 1 < (length (filter (\g -> r == getResource g) bs))
>          = r:(confResr rs bs)
>   | otherwise
>          = confResr rs bs


>scanLineRes :: Int -> WorkingInfo -> [Resource]
>scanLineRes l bs
>  = nub (map getResource (filter (\g -> l == getEndTime g) bs))


>scanLineAct :: Int -> WorkingInfo -> WorkingInfo
>scanLineAct l as
>    = filter (\g ->
>             (l == getEndTime g) ||
```

```
>             (((getStartTime g) < l) && (l < (getEndTime g)))
>             ) as


>transAction :: WorkingInfo -> WorkingInfo -> [Name] -> WorkingInfo
>transAction as bs ns = foldl getTrans as (generateTLt bs ns)


>getTrans :: WorkingInfo -> (Int, WorkingInfo) -> WorkingInfo
>getTrans as (t, ds)
>   = (filter (\g -> notElem g ds) as) ++ (map (transTime t) ds)
>          where
>            transTime t (a, b, m, n) = (a, b, t, t+(n-m))


>generateTLt :: WorkingInfo -> [Name] -> [(Int, WorkingInfo)]
>generateTLt bs []   = []
>generateTLt bs (n:ns) = (getTup bs n):(generateTLt bs ns)


>getTup :: WorkingInfo -> Name -> (Int, WorkingInfo)
>getTup bs n
>   = (getEndTime searchAct,
>      filter (\g -> (getResource g == getResource searchAct) &&
>                    (n /= getName g)) bs)
>      where
>        searchAct = losBrac (filter (\g -> n == getName g) bs)


>searchAct bs n = losBrac (filter (\g -> n == getName g) bs)


>losBrac [a] = a


>contRK :: Int -> WorkingInfo -> [Name] -> [(Resource, Int)]
>contRK l bs ns
>    = case ns of
>        []   -> []
>        n:ns -> (contInt l bs n)++(contRK l bs ns)
>          where
>            contInt l bs n
>              = case (l == (getEndTime tupGA)) of
```

```
>                True  -> []
>                False -> [((getResource tupGA), (getEndTime tupGA))]
>            tupGA = confAct bs n
>
>
>confAct bs n = losBrac [b| b <- bs, getName b == n]


>minTime ess = minT (map (\x -> maxT (map getEndTime x)) ess)


>maxT ts = foldl max 0 ts

>minT []  = 0
>minT [x] = x
>minT (x:xs) = min x (minT xs)
```

Anfügen aller Startaktionen, weitere Aktionen an die Scan-line anfügen,
Scan-line bewegen

```
>evalStep1 l as cs a = evalStep2 l' as' cs' a
>                      where
>                         l'  = nextScanline l as'
>                         ps  = getParallel a
>                         qs  = foldl fRemove ps cs
>                         as' = foldl (addAction l) as qs
>                         cs' = cs \\ [x| x <- cs, (snd x) <= l']
```

Ressourcenkonflikte bestimmen

```
>evalStep2 l as cs a
>   = evalStep3 l as bs cs a ccs
>       where
>         bs  = scanLineAct l as
>         ccs = rcombine (map
>            (\r -> map getName (filter (\g -> r == getResource g) bs))
>                               (confResr (scanLineRes l bs) bs)
```

```
>                         )
```

Ressourcenkonflikte auflösen

```
>evalStep3 l as bs cs a ccs
>    = case ccs of
>        [] -> evalStep4 l as cs a
>        _  -> concat (map execBranch ccs)
>         where
>           execBranch ns = evalStep4 l as' cs' a
>             where
>               as' = transAction as bs ns
>               cs' = cs ++ (contRK l bs ns)
```

Löschen von l-Zeit-direkten Scan-line-Grundaktionen aus der Eingabeaktion ‘a’

```
>evalStep4 l as cs a
>   = case a' of
>       NOTHING       -> [as]
>       (SOMETHING a) -> evalStep1 l as cs a
>         where
>           a' = foldl (\a n -> delGAExt n a) (SOMETHING a) ds
>                where
>                  ds = (map getName
>                       (filter (\a -> (getEndTime a) <= l) as)
>                        )
```

```
>eval = evalStep1 0 [] []
```

```
>evalOpt a = [e| e <- es,  (tMax e == minTime es)]
>             where
>               tMax e = maxT (map getEndTime e)
>               es     = eval a
```

Graphische Ergebnisdarstellung

```
  rs            : Liste aller bei ‘as’ vorkommenden Ressourcen
  t (Argument) : minimale Laufzeit

>matrix1 as
>       | as == []  = []
>       | otherwise = (map (\c -> " " ++ [c] ++ " ") rs):
>                       map matrixCol [1 .. t]
>              where
>                matrixCol t
>                  = map (rowCol as’) rs
>                    where
>                      as’ = (filter activeInT as)
>                            where
>                              activeInT (_, _, b, e)
>                                 = (b < t) && (e >= t)
>
>                rowCol as’ r
>                  = case (filter (\a -> (getResource a) == r) as’) of
>                     []            -> []
>                     [(n, r, _, _)] -> n
>
>                rs           = sort (nub (map getResource as))
>                t            = maximum0 (map getEndTime as)
```

```
  cs (Argument) : Liste von Spalteninformationen
  c             : Einzelne Spalte

>matrix2 css
>    | css == [] = [[]]
>    | otherwise = map visCol css
>        where
>          visCol cs
>            = foldl (\ds c -> ds ++ [c] ++ sep) sep cs’
>              where
```

```
>               cs'  = map (\s -> s ++ (repeatTake ' '
>                          (colB - (length s))) ++ "|") cs
>               colB = maximum (map length cs)
>               sep  = [(repeatTake '-' colB) ++ "+"]

>repeatTake c l = take l (repeat c)

>evalP' a = map (\as -> makeS (transpose (matrix2 (matrix1 as)))
>                 ++ "\n") aas
>            where
>              aas
>               = eval a
>              makeS ccs
>               = foldl (\s' s -> s' ++ s ++  "\n") [] (map concat ccs)
>              clean (a:as)
>               = a:clean (filter
>                         (\a' -> (a \\ a' /= []) || (a' \\ a /= []))
>                          as
>                          )
>              clean aas    = aas


>evalP'' a = map (\as -> makeS (transpose (matrix2 (matrix1 as)))
>                 ++ "\n") aas
>            where
>              aas
>               = evalOpt a
>              makeS ccs
>               = foldl (\s' s -> s' ++ s ++  "\n") [] (map concat ccs)
>              clean (a:as)
>               = a:clean (filter
>                         (\a' -> (a \\ a' /= []) || (a' \\ a /= []))
>                          as
>                          )
>              clean aas    = aas
```

Die Funktion '`evalP`' ermittelt für eine Eingabe-Aktion '`a`' alle nichtredundanten Ablaufpläne. Die Funktion '`evalOptP`' ermittelt alle nichtredundanten *optimalen* Ablaufpläne.

Funktionen '`evalP`' und '`evalOptP`'

```
>evalP a = sequence (map putStr (evalP' a)

>evalOptP a = sequence (map putStr (evalP'' a)
```

11.1.2 Auswertung

Für die Aktion

$$\begin{aligned}&\mathbf{pll}\ (\mathbf{seq}\ P_1(a,1), P_2(b,2)),\\ &\quad P_3(a,2)\end{aligned}$$

wurden in Abschnitt 4.4 die folgenden Darstellungen eingeführt, wobei die Funktion '`action`' die obige Aktion identifiziert:

```
>gA1 = GroundAction "P1" 'a' 1
>gA2 = GroundAction "P2" 'b' 2
>gA3 = GroundAction "P3" 'a' 2
>action = PLL [SEQ [gA1, gA2], gA3]
```

Mit Hilfe des Programms können für die Aktion '`action`' alle Ablaufpläne berechnet werden. Eingabe und Ausgabe am Bildschirm haben folgendes Aussehen:

```
Main> evalP action
```

a	P_3	P_3	P_1		
b				P_2	P_2

a	P_1	P_3	P_3
b		P_2	P_2

Mittels der Anfragefunktion '`evalOptP`' können alle optimalen Ablaufpläne ermittelt werden. Eingabe und Ausgabe haben folgendes Aussehen:

```
Main> evalOptP action
```

a	P_1	P_3	P_3
b		P_2	P_2

Bei der experimentellen Arbeit mit dem obigen Programm zeigte sich, daß auch bei Beispielen, bei denen ein ziemlich hoher ressourcenkonfliktbehafteter Parallelitätsgrad vorhanden ist, realistische Rechenzeiten benötigt werden. Beispielsweise konnten für *die folgende Aktion mit einer 6-stufigen Parallelität* mittels eines üblichen PCs (Pentium 350 Mhz) alle 322 nichtredundanten Ablaufpläne *innerhalb einer halben Minute* und alle 128 nichtredundanten optimalen Ablaufpläne *innerhalb einer drei viertel Minute* ausgegeben werden:

$$\begin{aligned}&\textbf{pll}\ (\textbf{pll}\ P_{10}(a,1),\ (\textbf{seq}\ P_1(b,3), P_2(a,3)),\ (\textbf{seq}\ P_3(a,2), P_4(b,3)),\ P_5(a,2)),\\ &\qquad (\textbf{pll}\ P_8(a,3),\ (\textbf{seq}\ P_6(b,6), P_7(a,1)))\end{aligned}$$

Für das in Abschnitt 2.1.1 vorgestellte Flow Shop-Problem, das der $\mathcal{RSV}$-Ausdruck

$$\begin{aligned}\textbf{pll}\ &(\textbf{seq}\ P_{11}(a,4), P_{12}(b,2)),\\ &(\textbf{seq}\ P_{21}(a,4), P_{22}(b,6)),\\ &(\textbf{seq}\ P_{31}(a,2), P_{32}(b,3)),\\ &(\textbf{seq}\ P_{41}(a,3), P_{42}(b,1)),\\ &(\textbf{seq}\ P_{51}(a,6), P_{52}(b,4)),\\ &(\textbf{seq}\ P_{61}(a,2), P_{62}(b,4))\end{aligned}$$

wiedergibt, können mit Hilfe desselben Rechners alle 13217 nichtredundanten Ablaufpläne *innerhalb einer halben Stunde* ausgegeben werden.

Wie das Beispiel 4.3.1 zeigt, können zu einer Eingabeaktion mehrere optimale Ablaufpläne (Prozeßaktionen d.h. Produktion- bzw. Projektdurchführungsmöglichkeiten) existieren. Ein interessantes Ergebnis ist nun, daß für eine Aktion mehrere optimale Ablaufpläne existieren können, die unterschiedliche Freizeiten der Ressourcen (Aufgabenträger) besitzen. Somit erhalten die Aufgabenträger sogar unter Beibehalten des Optimalitätskriteriums *eine hohe terminliche Flexibilität*, da sie eine optimale Projektdurchführungsmöglichkeit mit günstigen Freizeiten auswählen können. Dies soll an dem folgenden Beispiel verdeutlicht werden:

$$\begin{aligned}&\textbf{seq}\ (\textbf{pll}\ P_{10}(a,1),\ (\textbf{seq}\ P_1(b,3), P_2(a,3)),\ (\textbf{seq}\ P_3(a,2), P_4(b,3)),\ P_5(a,2)),\\ &\qquad (\textbf{pll}\ P_8(a,3),\ (\textbf{seq}\ P_6(b,6), P_7(a,1)))\end{aligned}$$

Für diese Aktion erscheinen nach der Berechnung alle 14 Ablaufpläne am Bildschirm; davon sind 10 optimale Ablaufpläne. Das Aussehen der 10 optimalen Ablaufpläne zeigen die folgenden Abbildungen:

a	P_5	P_5	P_3	P_3	P_{10}	P_2	P_2	P_2	P_8	P_8	P_8				P_7
b	P_1	P_1	P_1		P_4	P_4	P_4		P_6	P_6	P_6	P_6	P_6	P_6	

a	P_5	P_5	P_3	P_3	P_2	P_2	P_2	P_{10}	P_8	P_8	P_8				P_7
b	P_1	P_1	P_1		P_4	P_4	P_4		P_6	P_6	P_6	P_6	P_6	P_6	

a	P_5	P_5	P_{10}	P_3	P_3	P_2	P_2	P_2	P_8	P_8	P_8				P_7
b	P_1	P_1	P_1			P_4	P_4	P_4	P_6	P_6	P_6	P_6	P_6	P_6	

a	P_3	P_3	P_5	P_5	P_{10}	P_2	P_2	P_2	P_8	P_8	P_8				P_7
b	P_1	P_1	P_1	P_4	P_4	P_4			P_6	P_6	P_6	P_6	P_6	P_6	

a	P_3	P_3	P_5	P_5	P_2	P_2	P_2	P_{10}	P_8	P_8	P_8				P_7
b	P_1	P_1	P_1	P_4	P_4	P_4			P_6	P_6	P_6	P_6	P_6	P_6	

a	P_3	P_3	P_{10}	P_2	P_2	P_2	P_5	P_5	P_8	P_8	P_8				P_7
b	P_1	P_1	P_1		P_4	P_4	P_4		P_6	P_6	P_6	P_6	P_6	P_6	

a	P_3	P_3	P_{10}	P_5	P_5	P_2	P_2	P_2	P_8	P_8	P_8				P_7
b	P_1	P_1	P_1		P_4	P_4	P_4		P_6	P_6	P_6	P_6	P_6	P_6	

a	P_{10}	P_5	P_5	P_3	P_3	P_2	P_2	P_2	P_8	P_8	P_8				P_7
b	P_1	P_1	P_1		P_4	P_4	P_4		P_6	P_6	P_6	P_6	P_6	P_6	

a	P_{10}	P_3	P_3	P_2	P_2	P_2	P_5	P_5	P_8	P_8	P_8				P_7
b	P_1	P_1	P_1	P_4	P_4	P_4			P_6	P_6	P_6	P_6	P_6	P_6	

a	P_{10}	P_3	P_3	P_5	P_5	P_2	P_2	P_2	P_8	P_8	P_8				P_7
b	P_1	P_1	P_1	P_4	P_4	P_4			P_6	P_6	P_6	P_6	P_6	P_6	

11.2 Die $\mathcal{RCPSV}$-Diagrammberechnung

11.2.1 Programm

```
>import List
```

Die folgenden Funktionen beschreiben drei einfache Projektionen:

```
>idtAct (i, _, _) = i

>resAct (_, r, _) = r

>durAct (_, _, t) = t
```

Definition der Datenstruktur zur Darstellung von Netzplänen

```
>type Netplan  = ([Action], [(ActIdent, ActIdent)])

>type ActIdent = Int
>type Resource = Char
>type Duration = Int
>type Action   = (ActIdent, Resource, Duration)
```

Ermittlung der Menge aller Startaktionen

```
>startAct :: Netplan -> [Action]
>startAct p@(as, ps) = [a | a <- as, noAncestor a ps]
>                      where
>                          noAncestor _ [] = True
>                          noAncestor a@(i, _, _) ((_, i'):ps)
>                              | i == i'   = False
>                              | otherwise = noAncestor a ps
```

Die folgende Funktion '`predAct`' ermittelt alle Vorgänger einer Aktion '`i`'.
'`xs`' sind die Identifier aller Vorgänger von '`i`'.

```
>predAct :: Action -> Netplan -> [Action]
>predAct (i, _, _) (as, ps) = [a | a@(i, _, _) <- as, elem i xs]
>                             where
>                                 xs = [i' | (i', j') <- ps, i == j']
```

Datenstruktur zur Darstellung von Diagrammen bzw. Ablaufplänen

```
>type Time        = Int
>type Frozen      = Bool
>type Schedule    = [(Action, Time, Frozen)]
>type ExtSchedule = (Time, Schedule)
```

Erzeugung eines Startdiagramms:

```
>startSchedule :: Netplan -> ExtSchedule
>startSchedule p = (0, [(a, 0, False) | a <- startAct p])
```

Anfügen von Aktionen an die Scan-line

```
>blocRes :: ExtSchedule -> [Resource]
>blocRes (t, s)
>  = nub [r | ((_, r, t_d), t_s, b) <- s, b == True, t < (t_s + t_d)]

>getAct :: ExtSchedule -> Netplan -> ExtSchedule
>getAct e@(t, s) p@(as, ps)
>   = (t, s ++ [(a, t, False)| a <- as4])
>     where
>       bs  = blocRes e
>       as0 = [a | (a, _, _) <- s]
>       as1 = [a | (a@(_, _, t_d), t_s, _) <- s, (t_s + t_d) <= t]
>       as2 = as \\ as0
```

```
>       as3 = [a | a@(i', _, _) <- as2,  ((predAct a p) \\ as1) == [] ]
>       as4 = [a | a@(_, r, _) <- as3, notElem r bs]
```

Scanline Bewegen

Mittels der folgenden Funktion '`nextStop`' werden die Endzeiten aller Aktionen bei dem aktuellen Diagramm `(t, s)` berechnet, die größer als '`t`' (aktuelle Position der Scanline) sind. Anschließend wird der minimale Wert davon als der nächste Haltepunkt der Scan-line berechnet.

```
>nextStop :: ExtSchedule -> Time
>nextStop (t, s)
>  = minimum [t' | t' <- [t_s + (durAct a) | (a, t_s, _) <- s], t' > t]

>moveScanline :: ExtSchedule -> ExtSchedule
>moveScanline e@(t, s) = (nextStop e, s)
```

Ressourcenkonflikte bestimmen und auflösen
Alle fest plazierten Aktionen einfrieren

Die Funktion '`scanLineAct`' liefert die Menge aller '`t`'-Zeit Scan-line Aktionen bei dem Diagramm '`(t, s)`'.

```
>scanLineAct :: ExtSchedule -> [Action]
>scanLineAct (t, s)
>  = [a | (a, t_s, _) <- s, t_s + (durAct a) >= t, t_s < t]
```

Die Funktion '`dirScanLineAct`' liefert alle '`t`'-Zeit-direkten Scan-line-Aktionen.

```
>dirScanLineAct :: ExtSchedule -> [Action]
>dirScanLineAct (t, s) = [a | (a, t_s, _) <- s,  t_s + (durAct a) == t]
```

Die folgende Funktion '`scanLineRes`' liefert alle '`t`'-Zeit-Konfliktressourcen, wobei '`t`' der aktuellen Position der Scan-line bei dem Diagramm '`e`' entspricht.

```
>scanLineRes :: ExtSchedule -> [Resource]
>scanLineRes e = nub [r | (_, r, _) <- dirScanLineAct e]


>scanLineGrp :: ExtSchedule -> [[Action]]
>scanLineGrp e
>   = [xs | xs <- [[a | a <- as, (resAct a) == r]| r <- rs],
>                                               length xs > 1]
>           where
>             rs = scanLineRes e
>             as = scanLineAct e

>solveConflicts :: ExtSchedule -> [ExtSchedule]
>solveConflicts e@(t, s)
>     = case scanLineGrp e of
>         [] -> [e]
>         gs -> [(t, moveAct xs s) | xs <- combine gs]

>moveAct :: [(Action, [Action])] -> Schedule -> Schedule
>moveAct [] s
>    = s
>moveAct (g@(a, as):xs) s
>    = moveAct xs [moveSlot x g s| x <- s]

>moveSlot :: (Action, Time, Frozen) -> (Action, [Action])
>                        -> Schedule -> (Action, Time, Frozen)
>moveSlot x@(a', t_s, b) (a, as) s
>    | elem a' as
>        = let t_e
>        = (head [t_s | (a'', t_s, _) <- s, a'' == a]) + (durAct a)
>          in (a', t_e, b)
>    | a' == a
>        = (a', t_s, True)
>    | otherwise
>        = x

>combine :: [[Action]] -> [[(Action, [Action])]]
>combine []
>   = [[]]
```

```
>combine (xs:xxs)
>   = [((x, xs \\ [x]):xs') | x <- xs, xs' <- combine xxs]
```

Einfrieren aller ‘t’-Zeit-konfliktfreien Aktionen bei ‘s’:

```
>frzActScl :: ExtSchedule -> ExtSchedule
>frzActScl (t, s) = (t, [check x | x <- s])
>                   where
>                     check x@(a, t_s, b)
>                         | t_s + (durAct a) == t  = (a, t_s, True)
>                         | otherwise              = x
```

Kern

```
>core :: Netplan -> ExtSchedule -> [ExtSchedule]
>core p e@(_, s)
>   | elem False [b | (_,  _, b) <- s]
>     = concat [core p (getAct (frzActScl e') p) |
>               e' <- solveConflicts (moveScanline e)]
>   | otherwise
>     = [e]

>solve :: Netplan -> [ExtSchedule]
>solve p = core p (startSchedule p)

>extSolve :: Netplan -> [ExtSchedule]
>extSolve p = [e | e@(t, _) <- es, t == tMin]
>            where
>                tMin = minimum [t | (t, _) <- es]
>                es   = solve p
```

Ausgabe aller optimalen Ablaufpläne am Bildschirm

Ausgabe eines einzelnen Ablaufplans in Form eines Strings:

```
>schedStr :: ExtSchedule -> [Char]
>schedStr e@(t, s)
>   = concat [froz b ++ (showActIdent (idtAct a))
>                 ++ "|" ++ (space t_s)
>                 ++ (take (durAct a) (repeat (resAct a)))
>                 ++ "\n"
>                 | (a, t_s, b) <- s'] ++ "Scan-Line: "
>                 ++ (show t) ++ "\n"
>     where
>       showActIdent i = (space (3 - length (str'))) ++ str'
>                        where
>                          str' = show i
>       space x        = take x str
>       str            = repeat ' '
>       froz True      = "F"
>       froz False     = " "
>       s'             = [head [a | a@((i', _, _), _, _) <- s,
>                                  i == i']|
>                           i <- (sort [i | ((i, _, _), _, _) <- s])]


>schedStrLst es = concat [(schedStr e) ++ "\n\n" | e <- es]

>prtschedStrLst es = putStr (schedStrLst es)
```

11.2.2 Auswertung

Jede reduzierte Aktion aus $\mathcal{RSV}$ stellt trivialerweise ein $\mathcal{RCPS}$-Problem dar. Somit können alle optimalen Ablaufpläne einer reduzierten Aktion statt mittels der $\mathcal{RSV}$-Diagrammberechnung auch mittels der $\mathcal{RCPSV}$-Diagrammberechnung ermittelt werden. Für die in Abschnitt 11.1.2 vorgestellte Aktion

$$\textbf{seq}\ (\textbf{pll}\ P_{10}(a,1),\ (\textbf{seq}\ P_1(b,3), P_2(a,3)),\ (\textbf{seq}\ P_3(a,2), P_4(b,3)),\ P_5(a,2)),$$
$$(\textbf{pll}\ P_8(a,3),\ (\textbf{seq}\ P_6(b,6), P_7(a,1)))$$

wird nun die Ermittlung aller optimalen Ablaufpläne mittels der $\mathcal{RCPSV}$-Diagrammberechnung vorgestellt. Die obige Aktion wird wie folgt als Eingabe für das Berechnungsprogramm dargestellt:

```
>plan1 :: Netplan
>plan1 = ([(1, 'b', 3),
>          (2, 'a', 3),
```

```
>          (3, 'a', 2),
>          (4, 'b', 3),
>          (5, 'a', 2),
>          (6, 'b', 6),
>          (7, 'a', 1),
>          (8, 'a', 3),
>          (10, 'a', 1)],
>          [(1,2), (3, 4), (6, 7), (10, 8), (10, 6),
>          (2, 8), (2, 6), (4, 8), (4, 6), (5, 8), (5, 6)])
```

Durch Eingabe von

```
Main> prtschedStrLst (extSolve plan)
```

wird die Berechnung aller nichtredundanten aktiven Ablaufpläne durch das im letzten Abschnitt beschriebene Haskell-Programm veranlaßt. Folgende Ausgabe wird in der Folge erhalten.

```
F  1|bbb
F  2|     aaa
F  3|aa
F  4|   bbb
F  5|  aa
F  6|        bbbbbb
F  7|              a
F  8|        aaa
F 10|    a
Scan-Line: 15


F  1|bbb
F  2|    aaa
F  3|aa
F  4|   bbb
F  5|  aa
F  6|        bbbbbb
F  7|              a
F  8|        aaa
F 10|       a
Scan-Line: 15
```

```
F  1|bbb
F  2|     aaa
F  3|aa
F  4|   bbb
F  5|   aa
F  6|        bbbbbb
F  7|              a
F  8|        aaa
F 10|  a
Scan-Line: 15


F  1|bbb
F  2|   aaa
F  3|aa
F  4|   bbb
F  5|      aa
F  6|        bbbbbb
F  7|              a
F  8|        aaa
F 10|  a
Scan-Line: 15


F  1|bbb
F  2|     aaa
F  3|  aa
F  4|    bbb
F  5|aa
F  6|        bbbbbb
F  7|              a
F  8|        aaa
F 10|    a
Scan-Line: 15


F  1|bbb
F  2|    aaa
F  3|  aa
F  4|    bbb
F  5|aa
F  6|        bbbbbb
F  7|              a
```

```
F  8|        aaa
F 10|       a
Scan-Line: 15


F  1|bbb
F  2|     aaa
F  3|   aa
F  4|     bbb
F  5|aa
F  6|        bbbbbb
F  7|              a
F  8|        aaa
F 10|  a
Scan-Line: 15


F  1|bbb
F  2|     aaa
F  3| aa
F  4|   bbb
F  5|   aa
F  6|        bbbbbb
F  7|              a
F  8|        aaa
F 10|a
Scan-Line: 15


F  1|bbb
F  2|   aaa
F  3| aa
F  4|   bbb
F  5|      aa
F  6|        bbbbbb
F  7|              a
F  8|        aaa
F 10|a
Scan-Line: 15


F  1|bbb
F  2|     aaa
```

```
F  3|   aa
F  4|     bbb
F  5| aa
F  6|        bbbbbb
F  7|              a
F  8|        aaa
F 10|a
Scan-Line: 15
```

Ein einfacher Vergleich zeigt, daß die 10 oben ermittelten optimalen Ablaufpläne genau den 10 zuvor in Abschnitt 11.1.2 mittels der $\mathcal{RSV}$-Diagrammberechnung ermittelten Ablaufplänen entsprechen. Während die Ausgabe bei der $\mathcal{RSV}$-Diagrammberechnung maschinenorientierte Gantt-Diagramme zeigte, zeigt die obige Ausgabe ($\mathcal{RCPSV}$-Diagrammberechnung) auftragsorientierte Gantt-Diagramme. Wie bereits erwähnt, symbolisiert der Buchstabe `F` zu Beginn jeder Zeile, daß die zugehörige Aktion eingefroren ist.

Literaturverzeichnis

[AH73] ASHOUR, S. und S. R. HIREMATH: *A Branch-and-Bound Approach to the Job-Shop Scheduling Problem.* International Journal of Production Research, 11:47–58, 1973.

[Alb65] ALBACH, H.: *Maschinenbelegungspläne bei Einzelfertigung.* Landesamt für Forschung des Landes Nordrhein-Westfalen (Hrsg.), Jahrbuch 1965, Köln Opladen:12–49, 1965.

[Alt96] ALTROGGE, G.: *Netzplantechnik.* Oldenbourg, München-Wien, 3 Auflage, 1996.

[ANS85] ANSI: *Programming Languages - COBOL*, 1985. ANSI X3.23-1985 (R1991).

[Bal65] BALINSKY, M. L.: *Integer Programming: Methods, Uses, Computation.* Management Science, Seiten 171–216, November 1965.

[Bal67] BALAS, E.: *Finding a Minimaximal Path in a Disjunctive PERT Network.* Théorie des Graphes, Journées internationales d'étude, Rome, Juillet 1966, Dunod, Paris-Gordon and Beach, pp. 21-30, New York, 1967.

[Bal69] BALAS, E.: *Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithm.* Operations Research, 17:941–957, 1969.

[Bal70] BALAS, E.: *Project Scheduling with Resource Constraints.* In: BEALE, E. M. L. (Herausgeber): *In Applications of Mathematical Programm Techniques.* American Elsevier, New York, 1970.

[BB74] BERGEN, R. und P. BUBOLZ: *Netzplantechnik. Studienbuch für Studierende der Wirtschafts- und Sozialwissenschaften sowie aller Ingenieur- und Naturwissenschaften ab 1. Semester.* Frankfurt am Main, 1974.

[BBMR89] BORGIDA, A., R. J. BRACHMANN, D. L. MCGUINNESS und L. A. RESNICK: *CLASSIC: A Structural Data Model for Objects.* In: *Proceedings of the International Conference on Management of Data*, Seiten 58–67, Portland, Oregon, 1989.

[BH90] BAADER, F. und B. HOLLUNDER: *KRIS: Knowledge Representation and Inference System -System Description-.* Technischer BerichtTM-90-03, Deutsche Forschungszentrum für Künstliche Intelligenz GmbH, 1990.

[BH91] BELL, C. E. und J. HAN: *A New Heuristic Solution Method in Resource-Constrained Project Scheduling.* Naval Research Logistics, 38:315–331, 1991.

[Bir98] BIRD, R.: *Introduction to Functional Programming using Haskell.* Prentice Hall Press, zweite Auflage, 1998.

[BJ93] BRUCKER, P. und B. JURISCH: *A New Lower Bound for the Job-Shop Scheduling Problem.* European Journal of Operational Research, 64:156–167, 1993.

[BJK92] BRUCKER, P., B. JURISCH und KRÄMER: *The Job-Shop Problem and Immediate Selection.* Technischer BerichtHeft 145, Osnabrücker Schriften zur Mathematik, Reihe P, Universität Osnabrück, 1992.

[BJS91] BRUCKER, P., B. JURISCH und B. SIEVERS: *A Fast Branch & Bound Algorithm for the Job-Shop Scheduling Problem.* Technischer Bericht-Heft 136, Osnabrücker Schriften zur Mathematik, Reihe P, Universität Osnabrück, 1991.

[BMS64] BRAND, J. D., W. L. MEYER und L. R. SCHAFFER: *The Resource Scheduling Problem in Construction.* Technischer BerichtCivil Engineering Studies, Report No. 5, Dept. of Civil Engineering, University of Illinois, Urbana, 1964.

[Bö92] BÖRGER, E.: *Berechenbarkeit Komplexität Logik.* Vieweg, 1992.

[Boc90] BOCTOR, FAYEZ F.: *Some efficient multi-heuristic procedures for resource-constrained project scheduling.* European Journal of Operational Research, 49:3–13, 1990.

[Bow59] BOWMAN, E. H.: *The Schedule-Sequencing Problem.* Operations Research, 7(5):621–624, 1959.

[BP90] BELL, C. E. und K. PARK: *Solving Resource-Constrained Project Scheduling Problems by A* Search.* Naval Research Logistics Quarterly, 37(1):61–84, 1990.

[BPH82] Blackstone, J. H., D. T. Phillips und G. L. Hogg: *A State-of-the-Art Survey of Dispatching Rules for Manufacturing Job Shop Operations.* International Journal of Production Research, 20:27–45, 1982.

[BS85] Brachmann, R. J. und J. G. Schmolze: *An Overview of the KL-ONE Knowledge Representation System.* Congnitive Science, 9(2):171–216, 1985.

[Buz71] Buzi, Karoly: *Die Simulation optimaler Prioritätsregeln für die Ablaufplanung in der Werkstattfertigung.* Opladen, Köln, 1971.

[BW65] Brooks, G. H. und C. R. White: *An Algorithm for Finding Optimal or Near Optimal Solutions to the Production Scheduling Problem.* The Journal of Industrial Engineering, 16:34–40, 1965.

[Cap93] Cap, C. H.: *Theoretische Grundlagen der Informatik.* Springer-Verlag, 1993.

[Car82] Carlier, J.: *The One-Machine Sequencing Problem.* European Journal of Operational Research, 11:42–47, 1982.

[CAVT87] Christofides, N. R., Alvarez-Valdes und J. M. Tamarit: *Project Scheduling with Resource Constraints: A Branch and Bound Approach.* European Journal of Operational Research, 29:262–273, November 1987.

[CD70] Charlton, J. und C. C. Death: *A Method of Solution for General Machine-Scheduling Problems.* Operations Research, 18:689–707, 1970.

[Chu32] Church, A.: *A Set of Postulates for the Foundation of Logic.* Annals of Mathmatics, 2(33):346–366, 1932.

[Chu41] Church, A.: *The Calculi of Lambda-Conversion.* Annals of Mathmatics, (6), Princeton University Press, 1941.

[CJ76] Coffman Jr., E. G.: *Computer and Job-Shop Scheduling Theory.* Wiley, New York, 1976.

[CJM60] Conway, R. W., B. M. Johnson und W. L. Maxwell: *An Experimental Investigation of Priority Dispatching.* The Journal of Industrial Engineering, 11:221–229, 1960.

[Cla35] Clark, W.: *The Gantt Chart: A Working Tool of Management.* Pitman, London, 1935.

[CM62] Conway, R. W. und W. L. Maxwell: *Network Dispatching by the Shortest-Operation Discipline.* Operations Research, 10:51–73, 1962.

[CMM67] Conway, R. W., W. L. Maxwell und L. W. Miller: *Theory of Scheduling.* Addison-Wesley, Reading, Mass., 1967.

[Con65a] Conway, R. W.: *Priority Dispatching and Job Lateness in a Job Shop.* The Journal of Industrial Engineering, 16:228–237, 1965.

[Con65b] Conway, R. W.: *Priority Dispatching and Work-in-Process Inventory in a Job Shop.* The Journal of Industrial Engineering, 16:123–130, 1965.

[Coo71] Cook, S. A.: *The complexity of theorem proving procedures.* In: *Conf. Rec. of 3rd ACM Symp. on Thoery of Computing*, 1971.

[Coo76] Cooper, D. F.: *Heuristics for Scheduling Resource Constrained Projects: An Experimental Investigation.* Management Science, 22(11):1186–1194, 1976.

[Cor90] Corsten, H.: *Produktionswirtschaft: ablauforganisatorische Aspekte.* Oldenbourg, München, 1990.

[CP89] Carlier, J. und E. Pinson: *An Algorithm for Solving the Job-Shop Problem.* Operations Research, 8:487–503, 1989.

[CP90] Carlier, J. und E. Pinson: *A Practical Use of Jackson's Preemptive Schedule for Solving the Job Shop Problem.* Annals of Operations Research, 26:269–287, 1990.

[CP94] Carlier, J. und E. Pinson: *Adjustment of Heads and Tails for the Job-Shop Problem.* European Journal of Operational Research, 78:146–161, 1994.

[Dav69] Davis, E. W.: *An Exact Algorithm for the Multiple Constrained-Resource Project Scheduling Problem.* Doktorarbeit, Yale University, 1969.

[Dav92] Davie, Antony: *An Introduction to Functional Programming Systems Using Haskell.* Cambridge University Press, 1992.

[DD98] Domschke, W. und A. Drexl: *Einführung in Operations Research.* Springer, Berlin u. a., 4 Auflage, 1998.

[DH70] Day, J. E. und M. P. Hottenstein: *Review of Sequencing Research.* Naval Research Logistics Quarterly, 17:11–39, 1970.

[DH71] DAVIS, E. W. und G. E. HEIDORN: *An Algorithm for optimal project scheduling under multiple resource constraints.* Management Science, 17(12):B–803–B–816, August 1971.

[DH92] DEMEULEMEESTER, E. und W. HERROELEN: *A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem.* Management Science, 38(12):1803–1818, 1992.

[DLNN91] DONINI, F. M., M. LENZERINI, D. NARDI und W. NUTT: *The Complexity of Concept Languages.* In: *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, Seiten 151–162, Morgan Kaufmann, San Mateo, CA, 1991.

[DP75] DAVIS, E. W. und J. H. PATTERSON: *A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling.* Management Science, 21(8):944–955, November 1975.

[DSV97] DOMSCHKE, W., A. SCHOLL und S. VOSS: *Produktionsplanung: ablauforganisatorische Aspekte.* 2 Auflage, 1997.

[Dul93] DULGER, A.: *Prioritätsregeln für die industrielle Werkstattfertigung.* Doktorarbeit, Universität Regensburg, 1993.

[EBL98] ELLINGER, T., G. BEUERMANN und R. LEISTEN: *Operations Research, Eine Einführung.* Springer, Berlin u. a., 4 Auflage, 1998.

[EH77] ELMAGHRABY, S. E. und W. S. HERROELEN: *A Cost Minimization Problem with Resource-Duration Interaction.* Technischer Bericht Department of Applied Economic Sciences, Katholieke Universiteit Leuven, Belgium, 1977.

[Elm77] ELMAGHRABY, S. E.: *Activity Networks: Project Planning and Control by Network Models.* Wiley, New York, 1977.

[EN86] ELSAYED, E. A. und N. Z. NASR: *Heuristics for Resource Constrained Scheduling.* International Journal of Production Research, 24(2):299–310, 1986.

[Fis70] FISHER, M. L.: *Optimal solution of resource constrained network scheduling problem.* Technischer Bericht56, Operations Research Center, Massachusetts Institute of Technology, 1970.

[Fis73] FISHER, M.: *Optimal Solution of Scheduling Problems Using Lagrange Multipliers. Part I.* Operations Research, 21(5):1114–1127, 1973.

[Für97] FÜRST, A.: *Auftragsfolge- und Personalplanung.* München und Mering, 1997.

[Gal92] GAL, T.(HRSG.): *Grundlagen des Operations Research 3.* Berlin, 3 Auflage, 1992.

[Gen79] GENRICH, H. J.: *Ein Kalkül des Planens und Handelns.* In: PETRI, C. A. (Herausgeber): *Ansätze zur Organisationstheorie rechnergestützter Informationssysteme.* Oldenbourg, München, 1979.

[Ger66] GERE, W. S. JR.: *Heuristics in Job Shop Scheduling.* Management Science, 13:167–190, 1966.

[GJ79] GAREY, M. R. und D. S. JOHNSON: *COMPUTERS AND INTRACTABILITY, A Guide to the Theory of NP-Completeness.* Bell Telephone Laboratories, 1979.

[Gor72] GORENSTEIN, S.: *An Algorithm for Project Scheduling with Resource Constraints.* Operations Research, 20(4):835–850, 1972.

[Gre68] GREENBERG, H. H.: *A Branch-Bound Solution to the General Scheduling Problem.* Operations Research, 16:353–361, 1968.

[GT60] GIFFLER, B. und G. L. THOMPSON: *Algorithms for Solving Production-Scheduling Problems.* Operations Research, 8:487–503, 1960.

[HAB+96] HAMMOND, K., L. AUGUSTSSON, B. BOUTEL, W. BURTON, J. FAIRBAIRN, J. FASEL, A. GORDON, M. GUZMÁN, J. HUGHES, P. HUDAK, T. JOHNSSON, M. JONES, D. KIEBURTZ, R. NIKHIL, W. PARTAIN, J. PETERSON, S. PEYTON JONES und P WADLER: *Report on the programming language Haskell 1.3.* Technischer Bericht Department of Computer Science, University of Glasgow, 1996.

[Hau89] HAUPT, R.: *A Survey of Priority Rule-Based Scheduling.* OR Spektrum, 11:316, 1989.

[HC90] HUTCHISON, J. und Y.-L. CHANG: *Optimal nondelay Job Shop Schedules.* International Journal of Production Research, 28:245–257, 1990.

[HL62] HELLER, J. und G. LOGEMANN: *An Algorithm for the Construction and Evaluation of Feasible Schedules.* Management Science, 8:168–183, 1961/1962.

[HL97] HILLIER, F. S. und G. J. LIEBERMAN: *Operations Research, Einführung.* München-Wien, 5 Auflage, 1997.

[HNS77] HOLLOWAY, C. A., R. T. NELSON und V. SURAPHONGSCHAI: *Project Scheduling Program (PSP): A Multi-Pass Heuristic Program for a Class of Resource-Constrained Project Scheduling Problems.* Technischer Bericht59, Graduate School of Business, Stanford University, 1977.

[HNS79] HOLLOWAY, C. A., R. T. NELSON und V. SURAPHONGSCHAI: *Comparison of a Multi-Pass Heuristic Decomposition Procedure with other Resource-Constrained Project Scheduling Procedures.* Management Science, 25(9):862–872, 1979.

[HNSS90] HOLLUNDER, B., W. NUTT und M. SCHMIDT-SCHAUSS: *Subsumption Algorithms for Concept Description Languages.* In: *Proceedings of the 9th European Conference on Artificial Intelligence*, Seiten 348–353, Stockholm, Sweden, 1990.

[Hol89] HOLLUNDER, B.: *Subsumption Algorithms for Some Attributive Concept Description Languages.* Technischer BerichtSEKI Report SR-89-16, FB Informatik, Universität Kaiserslautern, Germany, 1989.

[HU88] HOPCROFT, J. E. und J. D. ULLMAN: *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie.* ADDISON-WESLEY, 1988.

[Hub71] HUBER, W. D.: *Computational Experience with Zero-One Programs for Multi-Project Scheduling.* August 1971.

[ISO97] ISO/IEC: *Information technology - Programming languages - FORTRAN - Part 1: Base language*, 1997. ISO/IEC 1539-1:1997.

[Jac56] JACKSON, J. R.: *An Extension of Johnson's Results on Job Lot Scheduling.* Naval Research Logistics Quarterly, 3:201–203, 1956.

[Joh54] JOHNSON, S. M.: *Optimal Two- and Three-Stage Production Schedules with Setup-Times Included.* Naval Research Logistics Quarterly, 1:61–68, 1954.

[Joh67] JOHNSON, T. J. R.: *An Algorithm for the Resource-Constrained Project Scheduling Problem.* Doktorarbeit, School of Management, M. I. T., August 1967.

[Joh74] JOHNSON, K. L.: *Grundlagen der Netzplantechnik.* Düsseldorf, 1974.

[Jon73] JONES, C. H.: *An Economic Evaluation of Job Shop Dispatching Rules.* Management Science, 20:293–307, 1973.

[KC91] Khattab, M. M. und F. Choobineh: *A New Approach for Project Scheduling with a Limited Resource.* International Journal of Production Research, 29(1):185–198, 1991.

[Kel63] Kelly, J. E. Jr.: *The Critical Path Method: Resource Planning and Scheduling, Ch 21 in Industrial Scheduling, Muth, J. F. AND Thompson, G. L. (eds.).* Prentice Hall, Englewood Cliffs, NJ, 1963.

[Kim94] Kim, P. S.: *Optimierung und Simplifikation von Anfragen an eine KL-ONE-Wissensbasis.* Diplomarbeit, Künstliche Intelligenz / Softwaretechnology, Fachbereich Informatik, Universität Frankfurt, Frankfurt, 1994.

[KR77] Kernighan, Brian W. und Dennis M. Ritchie: *The C Programming Language.* Prentice-Hall International, 1977.

[KS90] Kistner, Klaus-Peter und Marion Steven: *Produktionsplanung.* Physica-Verlag, Heidelberg, 1990.

[Küp82] Küpper, H. U.: *Ablauforganisation.* Stuttgart, New York, 1982.

[Kut00] Kutzner, A.: *Ein nichtdeterministischer call-by-need Lambda-Kalkül mit erratic Choice: Operationale Semantik, Programmtransformationen und Anwendungen.* Doktorarbeit, JWG-Universität in Frankfurt am Main, 2000.

[KWR93] König, W., O. Wendt und P. Rittgen: *Das Wirtschaftsinformatik-Schwerpunktprogramm "Verteilte DV-Systeme in der Betriebswirtschaft" der Deutschen Forschungsgemeinschaft -Frankfurt am Main, 1993-21 Bl.* Technischer Bericht13, Institut für Wirtschaftsinformatik, 1993.

[LLRK77] Lageweg, B. J., J. K. Lenstra und A. H. G. Rinnooy Kan: *Job-Shop Scheduling by Implicit Enumeration.* Management Science, 24:441–450, 1977.

[Loo90] Loogen, R: *Parallele Implementierung funktionaler Programmiersprachen.* Springer-Verlag, Berlin, Heidelberg, New York, London, Paris, Tokyo, Hong Kong, 1990.

[LRK78] Lenstra, J. K. und A. H. G. Rinnooy Kan: *Complexity of Scheduling under Precedence Constraints.* Operations Research, 26:22–35, 1978.

[LW77] Levy, J. und J. Wiest: *A Management Guide to PERT/CPM.* Prentice Hall, Englewood Cliffs, NJ, 2 Auflage, 1977.

[Man60] MANNE, A. S.: *On the Job-Shop Scheduling Problem.* Operations Research, 8:219–233, 1960.

[Mar91] MARTIN, J. C.: *Introduction to Languages and the Theory of Computation.* McGraw-Hill Inc., 1991.

[MM66] MOODIE, C. L. und D. E. MANDEVILLE: *Project Resource Balancing by Assembly Line Balancing Techniques.* Journal of Industrial Engineering, 17(7):377–383, July 1966.

[MM67] MÜLLER-MERBACH, H.: *Ein Verfahren zur Planung des optimalen Betriebsmitteleinsatzes bei der Terminierung von Großprojekten.* Zeitschrift für Wirtschaftliche Fertigung, 2, 3, February, March 1967.

[MM75] MÜLLER-MERBACH, H.: *Ganzzahlige Programmierung.* Grochla E., Wittmann W. HWB, Band 1, 4. Aufl., Stuttgart:3212–3217, 1975.

[MT63] MUTH, J. F. und G. L. THOMPSON: *Industrial Scheduling.* Prentice Hall, Englewood Cliffs, NJ, 1963.

[Neb90] NEBEL, B.: *Reasoning and Revision in Hybrid Representation Systems.* Lecture Notes in Artificial Intelligence 422. Springer Verlag, 1990.

[New75] NEW, C. C.: *Job Shop Scheduling: Is Manual Application of Dispatching Rules Feasible ?* Operational Research Quarterly, 26:35–43, 1975.

[NM93] NEUMANN, K. und M. MORLOCK: *Operations Research.* Hanser, München-Wien, 1993.

[NvL88] NEBEL, B. und K. VON LUCK: *Hybrid Reasoning in BACK.* In: RAS, Z. W. und L. SAITTA (Herausgeber): *Methodologies for Intelligent Systems*, Seiten 260–269. North Holland, Amsterdam, Netherlands, 1988.

[OB94] OGUZ, O. und H. BALA: *A Comparative Study of Computational Procedures for the Resource Constrained Project Scheduling Problem.* European Journal of Operational Research, 72:406–416, 1994.

[OH85] O'GRADY, P. J. und C. HARRISON: *A General Search Sequencing Rule for Job Shop Sequencing.* International Journal of Production Research, 23:961–973, 1985.

[Ohs98] OHSE, D.: *Quantitative Methoden in der Betriebswirtschaftslehre.* Verlag Vahlen, 1998.

[Oka98] Okasaki, C.: *Purely Functional Data Structures.* Cambridge University Press, 1998.

[OW90] Ottmann, T. und P. Widmayer: *Algorithmen und Datenstrukturen.* Wissenschaftsverlag, Mannheim/Wien/Zürich, 1990.

[Pat68] Patton, G. T.: *Optimal Scheduling of Resource Constrained Projects.* Doktorarbeit, Stanford University, August 1968.

[Pea84] Pearl, J.: *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading MA, 1984.

[PH74] Patterson, J. H. und W. D. Huber: *A Horizon-Varying, Zero-One Approach to Project Scheduling.* Management Science, 20(6):171–216, February, 1974.

[PHA+97] Peterson [ed.], J., K. Hammond [ed.], L. Augustsson, B. Boutel, W. Burton, J. Fasel, A. D. Gordon, J. Hughes, P. Hudak, Th. Johnsson, M. Jones, E. Meijer, S. Peyton Jones, A. Reid und P. Wadler: *Report on the Programming Language Haskell: A Non-strict, Purely Functional Language, Version 1.4*, 1997.

[PI77] Panwalkar, S. S. und W. Iskander: *A Survey of Scheduling Rules.* Operations Research, 25:45–61, 1977.

[Pin95] Pinedo, M.: *Scheduling - Theory, Algorithms, and Systems.* Prentice Hall, 1995.

[PR76] Patterson, J. H. und G. W. Roth: *Scheduling a Project Under Multiple Resource Constraints: A Zero-One Programming Approach.* AIIE Transactions, 8(4):449–455, December, 1976.

[PS92] Posthoff, C. und K. Schultz: *Grundkurs Theoretische Informatik.* B. G. Teubner Verlagsgesellschaft, Stuttgart; Leipzig, 1992.

[PW68] Pritsker, A. B. und L. J. Watters: *A Zero-One Programming Approach to Scheduling with Limited Resources.* Technischer BerichtRM-5561-PR, The RAND Corporation, January, 1968.

[PWW69] Pritsker, A. B., L. J. Watters und P. M. Wolfe: *Multiproject scheduling with limited resources: A zero-one programming approach.* Management Science, 16(1):93–108, September, 1969.

[Rit98] Rittgen, P.: *Prozeßtheorie der Ablaufplanung.* Teubner-Reihe Wirtschaftsinformatik. B. G. Teubner, Stuttgart, 1998.

[RL99] RABHI, FETHI und GUY LAPALME: *Algorithms: A functional programming approach.* Addison-Wesley, 1999.

[Row59] ROWE, A. J.: *Sequential Decision Rules in Production Scheduling.* Doktorarbeit, University of California, 1959.

[Row60] ROWE, A. J.: *Toward a Theory of Scheduling.* The Journal of Industrial Engineering, 11:125–136, 1960.

[RS64] ROY, S. und B. SUSSMAN: *Les problèmes d'ordonnancement avec constraintes disjonctives.* SEMA, Note D.S. No. 9 (décembre 1964).

[Sch70] SCHRAGE, L.: *Solving Resource-Constrained Network Problems by Implicit Enumeration-Nonpreemptive Case.* Operations Research, 10:263–278, 1970.

[Sch72] SCHRAGE, L.: *Solving resource-constrained network problems by implicit enumeration, preemptive case.* Operations Research, 20(3):668–677, 1972.

[Sch91] SCHMIDT, R. A.: *Algebraic Terminological Representation.* Technischer BerichtTechnical Report MPI-I-91-216, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1991.

[Sch92] SCHÖNING, UWE: *Logik für Informatiker.* Wissenschaftsverlag, 3 Auflage, 1992.

[Sch94] SCHWARZE, J.: *Netzplantechnik: eine Einführung in das Projektmanagement.* Herne, Berlin: Neue Wirtschafts-Briefe, 7 Auflage, 1994.

[SD67] SMITH, R. D. und R. A. DUDEK: *A General Algorithm for Solution of the n-Job, M-Machine Sequencing Problem of the Flow Shop.* Operations Research, 15:71–82, 1967.

[SDK78] STINSON, J. P., E. W. DAVIS und B. M. KHUMAWALA: *Multiple Resource-Constrained Scheduling Using Branch and Bound.* AIIE Transactions, 10(3):252–259, 1978.

[See75] SEELBACH, H.: *Ablaufplanung.* Physica, Würzburg, 1975.

[SH96] SPERSCHNEIDER, V. und B. HAMMER: *Theoretische Informatik.* Springer, 1996.

[Slo80] SLOWINSKI, R.: *Two approaches to problems of resources allocation among project activities: A comparative study.* Journal of the Operations Research Society, 31(8):711–723, 1980.

[Slo81] Slowinski, R.: *Multiobjective network scheduling with efficient use of renewable and non-renewable resources.* European Journal of Operational Research, 7(3):265–273, 1981.

[SS76] Specker, E. und V. Strassen: *Komplexität von Entscheidungsproblemen, Lecture notes in computer science; 43.* Springer, 1976.

[SSS88] Schmidt-Schauss, M. und G. Smolka: *Attributive Concept Descriptions with Unions and Complements.* Technischer BerichtSEKI Report SR-88-21, FB Informatik, Universität Kaiserslautern, D-6750, Germany, 1988.

[Sti75] Stinson, J.: *A Branch and Bound Algorithm for a General Class of Resource-Constrained Scheduling Problems.* In: *AIIE Conf. Proc.*, Seiten 337–342, Las Vegas, Nevada, 1975.

[Tal82] Talbot, F. B.: *Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive Case.* Management Science, 28(10):1197–1210, 1982.

[Tho99] Thompson, S.: *Haskell: The Craft of Functional Programming.* Addison-Wesley, zweite Auflage, 1999.

[Tow77] Townsend, W.: *Sequencing n Jobs on m Machines to Minimise Maximum Tardiness: A Branch-and-Bound Solution.* Management Science, 23:1016–1019, 1977.

[TP78] Talbot, F. B. und J. H. Patterson: *An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems.* Management Science, 24(11):1163–1174, 1978.

[Wag59] Wagner, H.: *An Integer Linear Programming Model for Machine Scheduling.* Naval Research Logistics Quarterly, 6(2):131–140, June, 1959.

[WBCS77] Weglarz, J., J. Blazewicz, W. Cellary und R. Slowinski: *Algorithm 520 - An automatic revised simplex method for constrained resources network scheduling.* ACM Transactions Mathmatical Software, 3(3):295–300, 1977.

[WBCS79] Weglarz, J., J. Blazewicz, W. Cellary und R. Slowinski: *New Models and Procedures for Resource Allocation Problems.* In: *Proc. 6 INTERNET Congress 2*, Seiten 521–530, Düsseldorf, 1979. VDI-Verlag.

[WBCS80] WEGLARZ, J., J. BLAZEWICZ, W. CELLARY und R. SLOWINSKI: *Control in Resource Allocation Systems.* Foundations Control Engineering, 5(3):159–180, 1980.

[Weg93] WEGENER, I.: *Theoretische Informatik.* B. G. Teubner, Stuttgart, 1993.

[Weg96] WEGENER, I.: *Kompendium Theoretische Informatik-eine Ideensammlung.* B. G. Teubner, Stuttgart, 1996.

[Wie63] WIEST, J. D.: *The Scheduling of Large Projects with Limited Resources.* Doktorarbeit, Carnegie Institute of Technology, 1963.

[Wie67] WIEST, J. D.: *A Heuristic Model for Scheduling Large Projects with Limited Resources.* Management Science, 13(6), February 1967.

[Wit93] WITTMANN, W.: *Enzyklopädie der Betriebswirtschaftslehre, Bd. 1: Handwörterbuch der Betriebswirtschaft.* Schäffer-Poeschel, Stuttgart, 1993.

[Woo84] WOOD, D.: *An Isothetic View of Computational Geometry.* Technischer BerichtCS-84-01, Department of Computer Science, University of Waterloo, Jan. 1984.

[WR90] WHITE, K. P. und R. V. ROGERS: *Job-Shop Scheduling: Limits of the Binary Disjunctive Formulation.* International Journal of Production Research, 28:2187–2200, 1990.

[WR93] WENDT, O. und P. RITTGEN: *ANDORI Graphs - an Extension of AND/OR Graphs for Distributed Problem Solving and Reasoning.* Technischer BerichtForschungsbericht IWI 93-09, Institut für Wirtschaftsinformatik, Universität Frankfurt, Frankfurt, 1993.

[WRK96] WENDT, O., P. RITTGEN und W. KÖNIG: *Solving Decision Problems by Distributed Decomposition and Delegation - Foundations of a Theory and its Application within a Normative Group Decision Support System framework.* König, Kurbel, Mertens, Preßmar, Seiten 55–72, 1996.

[WW95] WERNER, F. und A. WINKLER: *Insertion Techniques for the Heuristic Solution of the Job Shop Problem.* Discrete Applied Mathematics, 58:191–211, 1995.

[Zim97] ZIMMERMANN, W.: *Operations Research-Quantitative Methoden der Entscheidungsvorbereitung.* Oldenbourg, München, Wien, 8 Auflage, 1997.

Tabellarischer Lebenslauf

Pok-Son Kim
Am Alten See 21
60489 Frankfurt/Main

24.09.1964	geboren in Pohang/Republik Korea (Süd-Korea)
1971-1976	Besuch der Grundschule in Pohang
1977-1979	Besuch der Middle-School in Pohang
1980-1982	Besuch der High-School zuerst in Deagu ab 1981 in Pohang, Abitur
1983-1986	Studium der Mathematik an der Kook-Min Universität in Seoul/Süd-Korea. Das Studium wurde von der Kook-Min Universität durch ein spezielles Stipendium gefördert. Abschluß des Studiums mit Erreichen des Bachelor of Science
1988-1995	Studium der Informatik mit Nebenfach Mathematik an der Johann Wolfgang Goethe-Universität Frankfurt/Main Diplomarbeit unter Prof. Dr. M. Schmidt-Schauß mit dem Titel *Optimierung und Simplifikation von Anfragen an eine KL-ONE-Wissensbasis* Abschluß als Diplom-Informatikerin
1995-1996	Erwerben einer Gleichwertigkeit zu einem wirtschaftswissenschaftlichen Universitätsabschluss an der Johann Wolfgang Goethe-Universität Frankfurt/Main (Ablegung von Prüfungen in Allg. VWL und Allg. BWL)
1996-2000	Promotion an der Professur für Betriebswirtschaftslehre insbesondere Wirtschaftsinformatik und Informationsmanagement unter Prof. Dr. Wolfgang König an der Johann Wolfgang Goethe-Universität Frankfurt/Main. Die Promotion wurde in den Jahren 1996 bis 1998 durch ein Stipendium des Landes Hessen gefördert.

Erklärung

Ich habe die vorgelegte Dissertation selbst verfaßt und dabei nur die von mir angegebenen Quellen und Hilfsmittel benutzt. Alle Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Schriften entnommen sind, sowie alle Angaben, die auf mündlichen Auskünften beruhen, sind als solche kenntlich gemacht.