Modellierung von Scheduling-Problemen mit zeitbehafteten Petri-Netzen

Bachelorarbeit

Fachbereich 03: Informatik und Mathematik

Eingereicht von **Jens Schlöter**
jschloet@uni-bremen.de

Erstgutachterin und Betreuerin Dr. Sabine Kuske
Zweitgutachter Prof. Dr. Rolf Drechsler

24. April 2018

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

1 Einleitung

Scheduling-Probleme sind kombinatorische Probleme, die auf die Erstellung von Zeitplänen abzielen. So fasst [15] Scheduling als Entscheidungsprozess zusammen, der sich mit dem Planen von Ressourcen und Aufgaben in festgelegten Zeitintervallen befasst, wobei eine Zielfunktion optimiert werden soll. Ein prominentes Beispiel für ein solches Ziel ist die Zeit, die zur Abarbeitung aller Aufgaben benötigt wird. [9] nennen weitere mögliche Ziele. Das Scheduling ist dabei von hoher praktischer Bedeutung, so wird es laut [15] in Herstellungs- und Service-Unternehmen eingesetzt und spielt für diese eine wichtige Rolle. Auch [13] listen weitere Anwendungsfälle auf und betonen die Relevanz des Scheduling:

> „Eine Zuordnung von begrenzten Ressourcen über eine gewisse Zeit, beispielsweise die Zuordnung von Personal, Maschinen, Werkzeug, etc. zur Erledigung von Aufträgen (Jobs) innerhalb einer bestimmten Zeit, ist in fast allen Bereichen unternehmerischen Handelns unverzichtbar, kritisch und schwierig."

Wie das aufgeführte Zitat schon andeutet, handelt es sich beim Scheduling nicht nur um eine wichtige, sondern auch um eine schwierige Aufgabe. So erwähnt [15], dass viele Scheduling-Probleme *NP-Hart*[1] sind, insbesondere ist auch das *Resource-constrained project scheduling problem*, welches im Verlauf dieser Arbeit thematisiert wird, *NP-Vollständig* (vgl. [9]). Obwohl viel Forschung und Analyse zu verschiedensten Scheduling-Problemen betrieben wurde (vgl. [15]), kann es gerade für schwierige Probleme von Vorteil sein, auch Ergebnisse aus anderen Forschungsbereichen zu verwenden. Zum Beispiel betont [1] die Vorteile, die daraus entstehen, Scheduling-Probleme mit Hilfe anderer Techniken darzustellen und präsentiert eine Möglichkeit, diese mit Hilfe von *timed Petri Nets* zu modellieren. Dabei werden die graphische Darstellung, das mathematische Fundament sowie die Existenz von Analysemethoden als Vorteile der Modellierung genannt.

Bei *Petri-Netzen* handelt es sich um ein mathematisch fundiertes Werkzeug zur Modellierung von nebenläufigen, parallelen und verteilten Systemen (vgl. [12]). Ähnlich zu den bereits erwähnten timed Petri Nets, erweitern auch *zeitbehaftete Petri-Netze* die klassischen Petri-Netze um zeitliche Abhängigkeiten und bieten dabei ebenfalls die Vorteile einer graphischen Darstellung sowie eines mathematischen Fundamentes. Darüber hinaus existieren eigene Analysetechniken für zeitbehaftete Petri-Netze, wie zum Beispiel in [16] aufgeführt wird. Neben den theoretischen Analyseansätzen existieren auch Softwaretools, die diese umsetzen. So führen zum Beispiel [4] ein Tool zur Analyse und Simulation von zeitbehaftete Petri-Netzen ein, das unter anderem einen Model-Checker für Eigenschaften dieser zur Verfügung stellt.

Um die erwähnten Analysemethoden und Softwaretools auch im Kontext von Scheduling-Problemen nutzbar zu machen und auch von den anderen genannten Vorteilen zu profitieren, wird in dieser Arbeit ein ähnlicher Ansatz zu [1] gewählt, indem eine Konstruktion angegeben wird, die Scheduling-Probleme in zeitbehaftete Petri-Netze transformiert, sodass diese zur Analyse der zugrunde liegenden Scheduling-Probleme verwendet werden können.

[1] Wir gehen dabei davon aus, dass die Begriffe NP-Härte und NP-Vollständigkeit bekannt sind. Für eine Einführung dieser verweisen wir auf [8] und [5].

Dabei besteht die Arbeit aus drei Teilen. Zu Beginn wird ein Scheduling-Problem formal eingeführt. Da laut [15] eine „erstaunliche“ Anzahl an Scheduling-Modellen existiert, wird mit dem Resource-constrained project scheduling problem (RCPSP) ein möglichst generelles ausgewählt (vgl. [9]). Anschließend werden klassische und auch zeitbehaftete Petri-Netze definiert, die im weiteren Verlauf zur Modellierung verwendet werden. Im Hauptteil der Arbeit wird schließlich eine Konstruktion angegeben, die RCPSPs in zeitbehaftete Petri-Netze umformt, um abschließend ein Fazit zu ziehen und einen Ausblick auf mögliche weiterführende Arbeit sowie Anwendungen der bisherigen Ergebnisse zu geben.

2 Scheduling-Probleme

Wie bereits erwähnt, wird in dieser Arbeit das Resource-constrained project scheduling problem betrachtet, da es sich dabei um ein eher generelles Scheduling-Problem handelt. Besonders vorteilhaft ist dabei, dass sich viele andere populäre Scheduling-Probleme, wie *Single-machine scheduling*, *Parallel machine scheduling*, *Job scheduling* und *Multi-processor task scheduling*, mit Hilfe des RCPSP modellieren lassen (vgl. [9]).

Neben der Möglichkeit, andere Scheduling-Probleme zu modellieren, existiert auch eine große Anzahl an Anwendungen, die mit Hilfe des RCPSP dargestellt werden können. Während wir uns in dieser Arbeit hauptsächlich auf abstrakte Beispiele für RCPSPs beschränken, listen [9] solche Anwendungen auf und erläutern, wie diese als RCPSPs abgebildet werden können.

Obwohl es sich bereits bei der Grundform des RCPSP um ein recht allgemeines Scheduling-Problem handelt, existiert eine Reihe von Erweiterungen, die es weiter generalisieren (siehe zum Beispiel [9], [2] und [7]).

Im Verlauf dieses Kapitels wird zunächst die Grundform des RCPSP eingeführt, woraufhin ausgewählte Erweiterungen erläutert werden.

2.1 Resource-constrained project scheduling problem

Wie die meisten Scheduling-Probleme behandelt das RCPSP die Planung von Aktionen in einem gegebenen Zeitintervall unter Einhaltung gewisser Einschränkungen. Im Folgenden wird das RCPSP angelehnt an [9] und [2] definiert, wobei wir die Definition in drei einzelne Teile aufgliedern:

1. Problem
2. Lösung
3. Optimalität

Der erste Teil definiert das RCPSP mit seinen einzelnen Komponenten.

Definition 1. Ein *Resource-constrained project scheduling problem* (RCPSP) ist ein 7-Tupel $SP = (D, [n], [e], R^e, p, r^e, V)$ mit

- $D \in \mathbb{N}$ ist der *Zeit-Horizont*.
- $[n]$ ist die Menge der *Aktionen* mit $n \in \mathbb{N}_{>0}$ [2].
- $[e]$ ist die Menge der *erneuerbaren Ressourcen* mit $e \in \mathbb{N}$.
- $R \in \mathbb{N}^e$ ist ein Vektor $R^e = (R^e_1, \ldots, R^e_e)$, wobei R^e_k mit $k \in [e]$ das *maximale Vorkommen* der Ressource k angibt.
- $p \in \mathbb{N}^n$ ist ein Vektor $p = (p_1, \ldots, p_n)$, wobei p_i mit $i \in [n]$ die *Bearbeitungszeit* für die Aktion i angibt.
- $r^e \in \mathbb{N}^{n \times e}$ ist eine $(n \times e)$-Matrix, wobei r^e_{ik} [3] mit $i \in [n]$ und $k \in [e]$ die *benötigte Anzahl an Einheiten* der Ressource k für die Aktion i angibt.
- $V \subseteq ([n] \times [n])$ ist die *Vorgängerrelation* der Aktionen. Für diese gelte dabei, dass keine Folge $i_1, \ldots, i_k$ mit $k \geq 2$ und $(i_j, i_{j+1}) \in V$ für alle $j \in [k-1]$ existiert, sodass $i_1 = i_k$.

Das RCPSP besteht also grundsätzlich aus Aktionen und Ressourcen, wobei Erstere eine Bearbeitungsdauer und eine festgelegte Anzahl an benötigten Ressourcen haben. Zusätzlich existieren vorgegebene Reihenfolgen zwischen ihnen. Letztere hingegen haben ein maximales Vorkommen. Mit Hilfe dieser Komponenten lässt sich nun auch festlegen, wie eine Lösung des Problems auszusehen hat.

Definition 2. Gegeben sei ein RCPSP $SP = (D, [n], [e], R^e, p, r^e, V)$, dann ist ein Vektor $S \in \mathbb{N}^n$, dessen Einträge die Startzeiten der Aktionen in $[n]$ angeben, eine *zulässige Lösung* bzw. ein *zulässiger Schedule* von SP, wenn gilt:

- Alle Aktionen werden innerhalb des Zeithorizonts D gestartet, also $S_i \leq D$ für alle $i \in [n]$.
- Die Vorgängerrelation V wird eingehalten, also für alle $(i, j) \in V$: $S_i + p_i \leq S_j$.
- Die Ressourcen-Kapazität wird nicht überschritten. Für alle $t \in \{0, \ldots, D\}$ sei dazu $A_t = \{i \in [n] | (t \geq S_i) \wedge (S_i + p_i > t)\}$ die Menge aller Aktionen i, die zu einem Zeitpunkt t aktuell durchgeführt werden. Nun muss für alle $k \in [e], t \in \{0, \ldots, D\}$ gelten, dass $\sum_{i \in A_t} r^e_{ik} \leq R^e_k$.

Eine zulässige Lösung ist also eine Zuordnung von Aktionen zu Startzeitpunkten innerhalb des Zeithorizonts, sodass die zeitlichen Reihenfolgen, welche durch die Vorgängerrelation gegeben sind, eingehalten und die Ressourcen-Kapazitäten nicht überschritten werden. Zu beachten ist dabei, dass die Ressourcen erneuerbar sind, d.h. die Aktionen verwenden diese zwar entsprechend ihrer Bedarfe, sie sind nach der jeweiligen Bearbeitungsdauer allerdings wieder verfügbar.

In Definition 1 werden „Kreise“ innerhalb der Vorgängerrelation ausgeschlossen. Wäre dies nicht der Fall, wäre es zum Beispiel möglich, dass für zwei Aktionen i, j gilt, dass $(i, j) \in V$

[2] Die Schreibweise $[n]$ sei dabei und im Folgenden definiert als die Menge von natürlichen Zahlen $\{1, ..., n\}$

[3] Im Verlauf dieser Arbeit werden bei Mehrfachindizierung die einzelnen Indizes nur dann mit Komma getrennt, wenn dadurch Mehrdeutigkeiten vermieden werden.

und $(j, i) \in V$. Betrachtet man die Einschränkungen für zulässige Lösungen eines RCPSP, dann fällt auf, dass in einem solchen Fall dann nur eine Lösung existiert, falls $p_i = p_j = 0$, weil ansonsten kein Schedule existiert, der die Vorgängerrelation einhält. Dieser Spezialfall scheint zum einen aus Anwendungsperspektive nicht besonders sinnvoll zu sein, denn Aktionen, die unmittelbar nach ihrem Start erledigt sind, bedürfen keiner besonderen Planung. Zum Anderen führt dieser Spezialfall im späteren Verlauf der Arbeit im Kontext der Konstruktion (siehe Abschnitt 4) zu Problemen. Aus diesen beiden Gründen werden „Kreise“ in der Vorgängerrelation ausgeschlossen.

Betrachtet man noch einmal die Definitionen 1 und 2, dann sieht man leicht, dass mehrere Lösungen für ein RCPSP existieren können. Um also die Einschätzung der Qualität dieser Lösungen zu ermöglichen, wird im Folgenden ein Optimalitätskriterium eingeführt, welches der Zielfunktion entspricht, die bereits in der Einleitung erwähnt wurde.

Definition 3. Gegeben seien ein RCPSP $SP = (D, [n], [e], R^e, p, r^e, V)$ und ein zulässiger Schedule S von SP, dann ist S *optimal*, wenn die benötigte Zeit $C_{max} = \max\{C_i | i \in [n]\}$ mit $C_i = S_i + p_i$ minimal ist unter alle zulässigen Schedules von SP.

Ein zulässiger Schedule eines RCPSP ist also optimal, wenn die Zeit bis zur Beendigung der letzten Aktion minimal ist. Zu beachten ist, dass dieses Optimalitätskriterium nur eine von vielen Möglichkeiten ist (siehe auch [9]).

Bevor im Folgenden das RCPSP anhand eines Beispiels anschaulich erläutert wird, sei der *Vorgängerrelationen-Graph* eines RCPSP definiert als ein gerichteter Graph $G([n], V)$. Die Knoten von G stellen also die Aktionen dar, wobei eine Kante zwischen zwei Knoten $i, j \in [n]$ von i nach j genau dann existiert, wenn $(i, j) \in V$.

Beispiel 1. Sei also [7] die Menge der Aktionen, [4] die Menge der Ressourcen und $D = 30$ der Zeit-Horizont. Die Vorgängerrelation sei gegeben durch Abbildung 1.

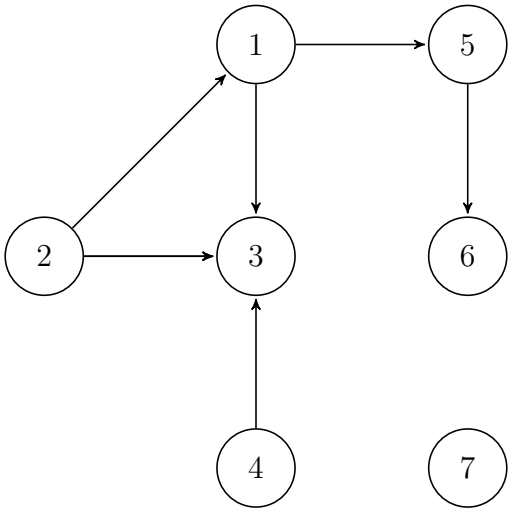


Abbildung 1: Vorgängerrelation zu Beispiel 1

Die Vorkommen der Ressourcen und Bearbeitungszeiten sowie Ressourcenbedarfe der Aktionen sind gegeben durch die Tabellen 1 und 2.

Ressourcen	1	2	3	4
Vorkommen	10	30	50	80

Tabelle 1: Vorkommen der Ressourcen in Beispiel 1

Aktionen		1	2	3	4	5	6	7
Bearbeitungszeit		5	3	2	10	2	4	12
Verbrauch Ressource	1	0	10	0	10	0	10	0
	2	20	0	0	0	0	15	0
	3	0	0	30	0	30	40	0
	4	0	0	0	80	0	0	80

Tabelle 2: Bearbeitungszeiten und Ressourcenbedarfe der Aktionen in Beispiel 1

Betrachtet man die Vorgängerrelation des Beispiels, dann fällt auf, dass die Aktionen 2, 4 und 7 prinzipiell zum Zeitpunkt 0 gestartet werden könnten, weil diese Aktionen keine Vorgänger haben. Allerdings können die Aktionen 2 und 4 nicht beide gleichzeitig zu Zeitpunkt 0 gestartet werden, weil beide jeweils 10 Einheiten von Ressource 1 benötigen und diese nur ein Vorkommen von 10 Einheiten hat.

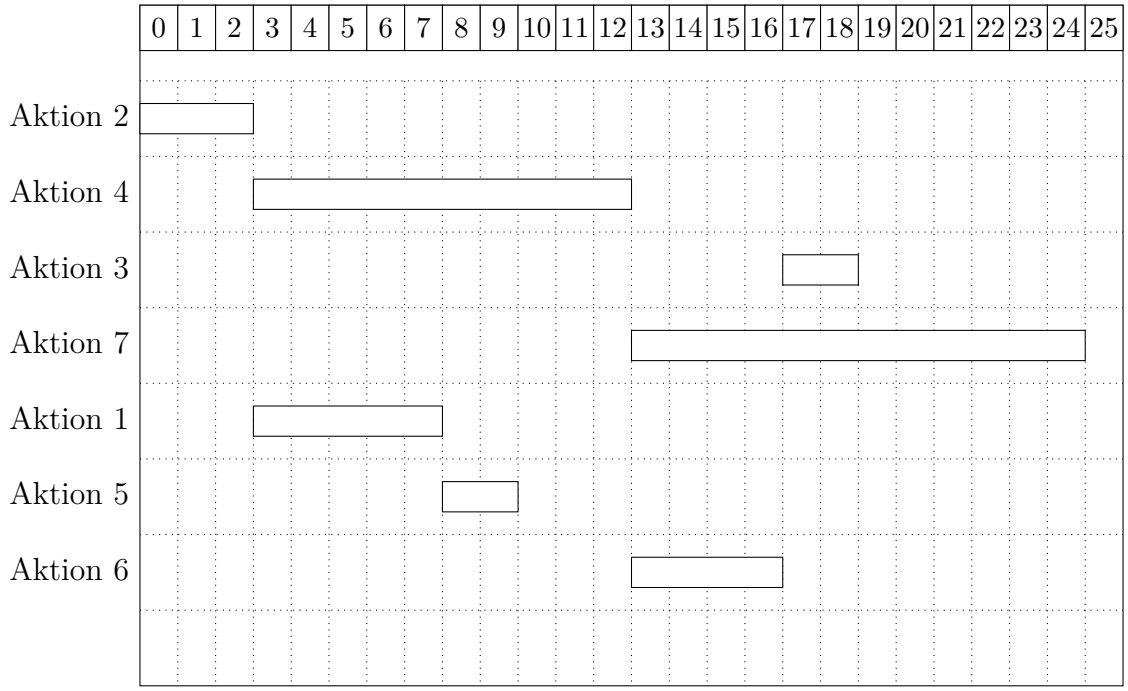


Abbildung 2: Schedule zu Beispiel 1

Ebenso können Aktion 4 und 7 nicht zeitgleich aktiv sein, da diese mit 80 Einheiten jeweils das komplette Vorkommen von Ressource 4 benötigen. Ein potentieller zulässiger Schedule startet also zum Beispiel mit Aktion 2. Nach Beendigung dieser kann die Aktion 1 gestartet werden, weil dann die Vorgängerrelation erfüllt ist. Außerdem kann Aktion 4 gestartet werden, da die Ressource 1 nicht mehr von Aktion 2 belegt wird. Weil die Aktionen 1 und 4 keine gemeinsamen Ressourcenbedarfe haben, können sie sogar Parallel gestartet werden, sobald Aktion 2 beendet ist, also zu Zeitpunkt 3. Abbildung 2 zeigt einen kompletten zulässigen (nicht zwangsläufig optimalen) Schedule zu Beispiel 1.

2.2 Modes

In den bisher eingeführten Definitionen ist die Bearbeitungszeit p_i einer Aktion i konstant. Es sind aber Anwendungsfälle denkbar, in denen eine Aktion schneller ausführbar ist, wenn mehr Ressourcen verwendet werden. Nehmen wir zum Beispiel an, ein Hausbau soll als RCPSP modelliert werden, wobei das Legen des Fundamentes eine Aktion und Arbeitskräfte eine Ressource sind, dann ist leicht vorstellbar, dass das Legen des Fundamentes weniger Zeit benötigt, wenn mehr Arbeitskräfte daran beteiligt sind. Um also solche Sachverhalte zu modellieren, wird das RCPSP im Folgenden um das Konzept der *Multiple Modes* erweitert. Die Definition orientiert sich dabei an [7].

Definition 4. Ein *RCPSP mit Multiple Modes* ist ein 8-Tupel

$$SPM = (D, [n], [e], R^e, M, p, r^e, V)$$

mit

- D, $[n]$, $[e]$, R^e und V sind wie in Definition 1 gegeben.
- $M \in \mathbb{N}_{>0}^n$ ist ein Vektor $M = (M_1, \ldots, M_n)$, wobei M_i mit $i \in [n]$ die Anzahl der verfügbaren *Modi* für die Aktion i angibt. Sei im Folgenden M_{max} der Wert des Eintrages von M mit dem maximalen Wert.
- $p \in \mathbb{N}^{n \times M_{max}}$ ist eine $(n \times M_{max})$-Matrix, wobei p_{im} mit $i \in [n]$ und $m \in [M_i]$ die *Bearbeitungszeit* von Aktion i in Modus m angibt. Die restlichen Werte der Matrix sind für das Problem nicht relevant und seien mit ∞ belegt.
- $r^e \in \mathbb{N}^{n \times e \times M_{max}}$ ist eine $(n \times e \times M_{max})$-Matrix, wobei r^e_{ikm} mit $i \in [n]$, $k \in [e]$ und $m \in [M_i]$ die *benötigte Anzahl an Einheiten* der Ressource k für die Aktion i im Modus m angibt. Die restlichen Werte der Matrix sind für das Problem nicht relevant und seien mit ∞ belegt.

Das RCPSP mit Multiple Modes erweitert also die Grundform des RCPSP um Modi für die Aktionen, in denen diese ausgeführt werden können. Führt man sich die Definition eines zulässigen Schedules für die Grundform des RCPSP vor Augen, dann fällt auf, dass diese nicht mehr ohne Weiteres anwendbar ist, denn Ressourcenbedarfe und Bearbeitungszeiten hängen nun auch von den Modi ab, in denen die Aktionen ausgeführt werden. Ein zulässiger Schedule für ein RCPSP mit Multiple Modes besteht also nicht nur aus einer Zuordnung

der Aktionen zu Startzeitpunkten, sondern ordnet die Aktivitäten ebenfalls den Modi zu, in denen sie ausgeführt werden. Dabei bleiben die restlichen Restriktionen analog zu Definition 2.

Definition 5. Gegeben sei ein RCPSP mit Multiple Modes $SPM = (D, [n], [e], R^e, M, p, r^e, V)$. Ein Tupel (S, m) mit $S, m \in \mathbb{N}^n$, wobei die Einträge von S die Startzeiten der Aktionen und die Einträge von m die Modi der Aktionen angeben, ist eine *zulässige Lösung* bzw. ein *zulässiger Schedule* von SPM, wenn gilt:

- Alle Aktionen werden innerhalb des Zeithorizonts D gestartet, also $S_i \leq D$ für alle $i \in [n]$.
- Es werden nur tatsächlich vorhandene Modi verwendet, also $m_i \leq M_i$ für alle $i \in [n]$.
- Die Vorgängerrelation V wird eingehalten, also $S_i + p_{im_i} \leq S_j$ für alle $(i, j) \in V$.
- Die Ressourcen-Kapazität wird nicht überschritten. Für alle $t \in \{0, \ldots, D\}$ sei dazu $A_t = \{i \in [n] | t \geq S_i \wedge S_i + p_{im_i} < t\}$ die Menge aller Aktionen i, die zu einem Zeitpunkt t durchgeführt werden. Nun muss gelten: $\forall k \in [e], t \in \{0, \ldots, D\} : \sum_{i \in A_t} r^e_{ikm_i} \leq R^e_k$.

Die Optimalität einer solchen Lösung wird analog zu Definition 3 definiert.

Definition 6. Gegeben seien ein RCPSP mit Multiple Modes

$$SPM = (D, [n], [e], M, R^e, p, r^e, V)$$

und ein zulässiger Schedule (S, m) von SPM, dann ist (S, m) *optimal*, wenn die benötigte Zeit $C_{max} = \max\{C_i | i \in [n]\}$ mit $C_i = S_i + p_{im_i}$ minimal ist unter alle zulässigen Schedules von SPM.

Beispiel 2. Um Beispiel 1 um Modi zu erweitern, muss zu jeder Aktion angegeben werden, welche Modi es gibt. Außerdem müssen Bearbeitungszeiten und Ressourcenbedarfe in Abhängigkeit der Modi angegeben werden. Wie in Beispiel 1 ist [7] die Menge der Aktionen, [4] die Menge der Ressourcen und $D = 30$ der Zeithorizont. Außerdem seien die Vorgängerrelation sowie Ressourcenvorkommen weiterhin durch Abbildung 1 bzw. Tabelle 1 gegeben.

Modus	Bearbeitungszeiten und Ressourcenbedarfe								
1	Aktionen		1	2	3	4	5	6	7
	Bearbeitungszeit		5	3	2	10	2	4	12
	Verbrauch Ressource	1	0	10	0	10	0	10	0
		2	20	0	0	0	0	15	0
		3	0	0	30	0	30	30	0
		4	0	0	0	80	0	0	80
2	Aktionen				3	4		6	7
	Bearbeitungszeit				3	12		8	18
	Verbrauch Ressource	1			0	10		10	0
		2			0	0		15	0
		3			20	0		15	0
		4			0	40		0	40

Tabelle 3: Bearbeitungszeiten und Ressourcenbedarfe der Aktionen in verschiedenen Modi aus Beispiel 2

Die Bearbeitungszeiten und Ressourcenbedarfe in Abhängigkeit von den Modi seien nun gegeben durch Tabelle 3, wobei sie für Modus 1 den Werten aus vorigem Beispiel entsprechen. Anders als im vorherigen Beispiel haben die Aktionen 3, 4, 6 und 7 nun einen weiteren Modus mit anderen Bearbeitungszeiten und Ressourcenbedarfen.

Eine Lösung dieses Beispieles muss zusätzlich zu den Startzeitpunkten der einzelnen Aktionen auch angeben, in welchem Modus die Aktionen ausgeführt werden sollen. Die Aktionen 1,2 und 5 haben dabei nur einen Modus und erfordern insofern keine Angabe. Betrachtet man die Aktionen 4 und 7, dann fällt auf, dass diese beiden Aktionen in Modus 1 nacheinander ausgeführt werden müssen, weil beide in Modus 1 jeweils das gesamte Vorkommen von Ressource 4 benötigen. In Modus 2 benötigen beide Aktionen jeweils nur die Hälfte des Vorkommens der Ressource, allerdings eine längere Bearbeitungszeit. Weil Aktion 7 schon zu Zeitpunkt 0 gestartet werden kann, wenn beide Aktionen im Modus 2 ausgeführt werden, ist Modus 2 in diesem Fall die bessere Wahl. Die Aktionen 3 und 6 können in Modus 2 ebenfalls parallel ausgeführt werden. Allerdings verhindert die zeitliche Abhängigkeit zwischen den Aktionen 4 und 3 in Kombination mit erhöhten Bearbeitungszeiten in Modus 2, dass ein Vorteil aus der möglichen Parallelisierung gezogen werden kann. Abbildung 3 zeigt einen zulässigen Schedule für Beispiel 2.

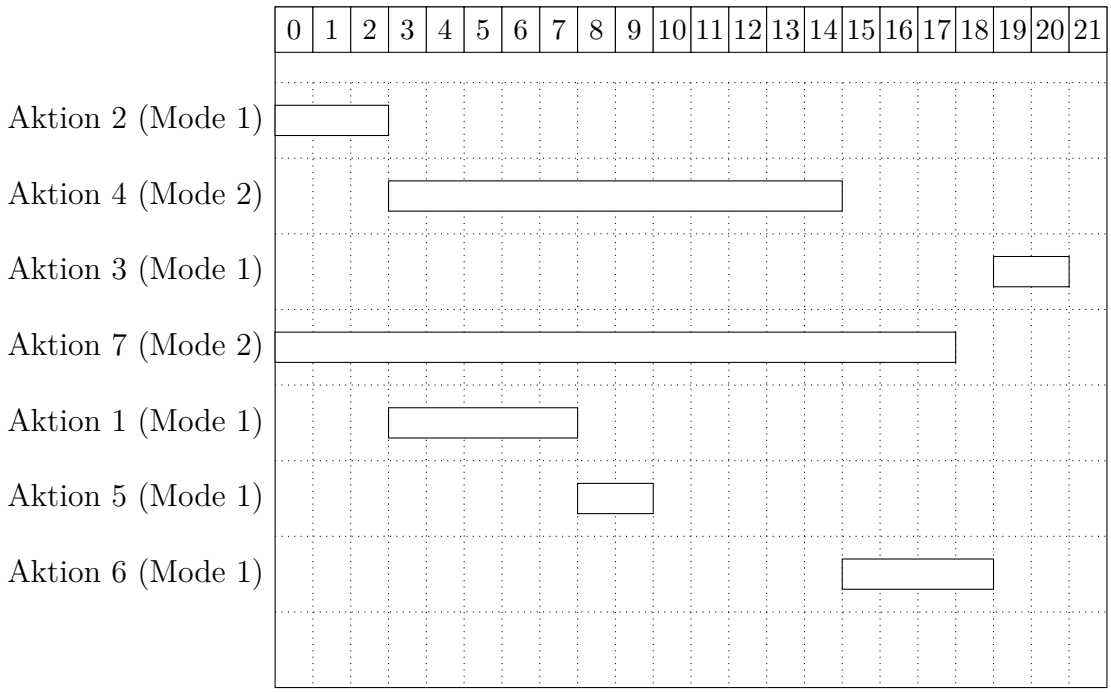


Abbildung 3: Schedule zu Beispiel 2

Zu beachten ist, dass auch der Schedule aus Abbildung 2 keine Restriktionen des RCPSP mit Multiple Modes verletzt und somit eine zulässige Lösung für Beispiel 2 ist. Allerdings benötigt der Schedule aus Abbildung 3 weniger Zeit zum Abarbeiten aller Aktionen und ist somit näher an einer optimalen Lösung.

2.3 Nicht erneuerbare Ressourcen

Die bisherigen RCPSP Modelle stellen genau eine Modellierung von Ressourcen zur Verfügung, in der diese von Aktionen verwendet und nach deren Bearbeitungsdauer freigegeben werden. Diese Modellierung ist für einige Anwendungsfälle nicht besonders gut geeignet. Betrachten wir dazu noch einmal das Modell eines Hausbaus als RCPSP mit dem Legen des Fundamentes als Aktion und Arbeitskräften als erneuerbare Ressource. Es erscheint durchaus sinnvoll, dass die Arbeitskräfte, die für das Legen des Fundamentes benötigt werden, nach Abschluss der Aktion wieder verfügbar für andere Aktivitäten sind. Möchte man nun allerdings den Beton, aus dem das Fundament gegossen wird, als Ressource modellieren, dann ist dieser nach dem Legen des Fundamentes fest verbaut und sollte nicht mehr verfügbar sein.

Um die Modellierung von Situationen dieser Art zu ermöglichen, wird im Folgenden eine weitere RCPSP Variante definiert, die sogenannte *nicht erneuerbare Ressourcen* enthält (siehe dazu auch [7]).

Definition 7. Ein *RCPSP mit Multiple Modes und nicht erneuerbaren Ressourcen* ist ein 11-Tupel $SPMN = (D, [n], [e], [v], R^e, R^v, M, p, r^e, r^v, V)$ mit

- $(D, [n], [e], R^e, M, p, r^e, V)$ ist ein RCPSP mit Multiple Modes nach Definition 4.
- $[v]$ ist die Menge von *nicht erneuerbaren Ressourcen* mit $v \in \mathbb{N}$.
- $R^v \in \mathbb{N}^v$ ist ein Vektor $R^v = (R^v_1, \ldots, R^v_v)$, wobei R^v_q mit $q \in [v]$ das *maximal verfügbare Vorkommen* von Ressource q angibt.
- $r^v \in \mathbb{N}^{n \times v \times M_{max}}$ ist eine $(n \times v \times M_{max})$-Matrix, wobei r^v_{iqm} mit $i \in [n]$, $q \in [v]$ und $m \in [M_i]$ die *benötigte Anzahl an Einheiten* der Ressource q für Aktion i im Modus m angibt. Die restlichen Werte der Matrix sind für das Problem nicht relevant und seien mit ∞ belegt.

In einer zulässigen Lösung eines RCPSP mit Multiple Modes und nicht erneuerbaren Ressourcen dürfen dementsprechend insgesamt nicht mehr Einheiten einer erneuerbaren Ressource verbraucht werden, als ursprünglich verfügbar sind.

Definition 8. Gegeben sei ein RCPSP mit Multiple Modes und nicht erneuerbaren Ressourcen $SPMN = (D, [n], [e], [v], R^e, R^v, M, p, r^e, r^v, V)$. Ein Tupel (S, m) mit $S, m \in \mathbb{N}^n$ ist ein *zulässiger Schedule* von $SPMN$, wenn gilt:

- (S, m) ist eine zulässige Lösung für das RCPSP mit Multiple Modes $SPM = (D, [n], [e], R^e, M, p, r^e, V)$
- Die Kapazität der erneuerbaren Ressourcen wird nicht überschritten, also $\forall q \in [v]$: $\sum_{i=1}^{n} r^v_{iqm_i} \leq R^v_q$.

Das Hinzufügen der nicht erneuerbaren Ressourcen hat keinerlei Auswirkungen auf das Optimalitätskriterium für RCPSPs mit Multiple Modes, sodass dieses auch für Probleme mit nicht erneuerbaren Ressourcen verwendet werden kann.

Beispiel 3. Seien Aktionen, Bearbeitungszeiten, Ressourcenvorkommen, Vorgängerrelation und Ressourcenbedarfe wie in Beispiel 2. Allerdings sei nun Ressource 4 eine nicht erneuerbare Ressource. In Beispiel 2 wurde erläutert, dass die Abbildungen 2 und 3 jeweils zulässige Schedules darstellen. Ist nun 4 eine nicht erneuerbare Ressource, dann stellt Abbildung 2 keinen Schedule mehr dar, denn die Aktionen 4 und 7 verbrauchen jeweils 80 Einheiten der Ressource. Da diese allerdings nur ein Vorkommen von 80 hat, wird dieses überschritten, weil insgesamt 160 Einheiten verbraucht werden. Wäre Ressource 4 nach wie vor erneuerbar, würde dies kein Problem darstellen, weil zu keinem Zeitpunkt mehr als 80 Einheiten gleichzeitig verwendet werden.

2.4 Allgemeinere Vorgängerrelation

Mit der Vorgängerrelation aus Definition 1 (bzw. 4) lassen sich nur Einschränkungen der Form $S_i + p_i \leq S_j$ für zwei Aktionen $i, j \in [n]$ formulieren. In diesem Abschnitt wird ein weiteres Konzept zur zeitlichen Restriktion der Startzeitpunkte zweier Aktionen $(i, j) \in V$ erläutert, das es ermöglicht, den zeitlichen Abstand zwischen dem Beenden von Aktion i und dem Start von Aktion j einzuschränken.

Zur Veranschaulichung dieses Konzeptes ziehen wir noch einmal das Hausbau-Beispiel heran. Seien mit dem Aufstellen der Wände und dem Platzieren eines Stützpfeilers nun weitere Aktionen gegeben. Die beiden Aktivitäten müssen nach dem Legen des Fundamentes gestartet werden, was sich mit Hilfe der Vorgängerrelation modellieren lässt. Es ist allerdings zu beachten, dass die Wände erst aufgestellt werden können, wenn der Beton getrocknet ist, sodass ein gewisser zeitlicher Abstand zwischen beiden Aktionen bestehen muss. Der Pfeiler dagegen muss im flüssigen Beton platziert werden, sodass der Beton noch nicht getrocknet sein darf und somit ein gewisser zeitlicher Abstand zwischen den Aktionen nicht überschritten werden darf.

Das Konzept der *Verzögerungen*, mit dem sich solche Situation modellieren lassen, wird in der folgenden Definition angelehnt an [7] eingeführt.

Definition 9. Ein *RCPSP mit Multiple Modes, erneuerbaren Ressourcen und Verzögerungen* ist ein 13-Tupel $SPMNV = (D, [n], [e], [v], R^e, R^v, M, p, r^e, r^v, V, d, \hat{d})$ mit

- $(D, [n], [e], [v], R^e, R^v, M, p, r^e, r^v, V)$ ist ein RCPSP mit Multiple Modes und erneuerbaren Ressourcen.
- $d \in \mathbb{N}^{n \times n}$ ist eine $(n \times n)$-Matrix, wobei d_{ij} mit $i, j \in [n]$ und $(i, j) \in V$ angibt, wie viel Zeit zwischen dem Endzeitpunkt von i und dem Startzeitpunkt von j mindestens vergehen muss. Dieser Wert wird auch *minimale Verzögerung* zwischen i und j genannt. Die restlichen Werte der Matrix sind für das Problem nicht relevant und seien mit ∞ belegt
- $\hat{d} \in \mathbb{N}^{n \times n}$ ist eine $(n \times n)$-Matrix, wobei $\hat{d}_{ij}$ mit $i, j \in [n]$ und $(i, j) \in V$ angibt, wie viel Zeit zwischen dem Endzeitpunkt von i und dem Startzeitpunkt von j maximal vergehen darf. Dieser Wert wird auch *maximale Verzögerung* zwischen i und j genannt. Die restlichen Werte der Matrix sind für das Problem nicht relevant und seien mit ∞ belegt.
- $\hat{d}_{ij} \geq d_{ij}$ für alle $(i, j) \in V$.

Zusätzlich muss auch die Definition einer zulässigen Lösung um die Verzögerungen erweitert werden, sodass eine solche weder minimale noch maximale Verzögerungen verletzt.

Definition 10. Gegeben sei ein RCPSP mit Multiple Modes, nicht erneuerbaren Ressourcen und Verzögerungen $SPMNV = (D, [n], [e], [v], R^e, R^v, M, p, r^e, r^v, V, d, \hat{d})$. Ein Tupel (S, m) mit $S, m \in \mathbb{N}^n$ ist ein *zulässiger Schedule* von $SPMNV$, wenn gilt:

- (S, m) ist ein zulässiger Schedule für das RCPSP mit Multiple Modes und nicht erneuerbaren Ressourcen $SPMN = (D, [n], [e], [v], R^e, R^v, M, p, r^e, r^v, V)$.
- Für alle $(i, j) \in V$ wird die Vorgängerrelation und die minimale Verzögerung eingehalten, also $\forall (i, j) \in V : S_i + p_i + d_{ij} \leq S_j$
- Für alle $(i, j) \in V$ wird die Vorgängerrelation und die maximale Verzögerung eingehalten, also $\forall (i, j) \in V : S_i + p_i + \hat{d}_{ij} \geq S_j$

Eine Aktion j kann nun also frühestens d_{ij} und muss spätestens $\hat{d}_{ij}$ Zeiteinheiten nach Beendigung einer Vorgängeraktivität i gestartet werden. Die Optimalität einer zulässigen Lösung für ein RCPSP mit Multiple Modes, nicht erneuerbaren Ressourcen und Verzögerungen wird wieder entsprechend Definition 6 definiert.

Im weiteren Verlauf der Arbeit werden ausschließlich RCPSPs mit Multiple Modes, nicht erneuerbaren Ressourcen und Verzögerungen betrachtet, die entsprechend Definition 9 definiert sind, deshalb werden solche Probleme im Folgenden als RCPSP bezeichnet.

Beispiel 4. In diesem Beispiel wird das vorige um Verzögerungen ergänzt. Aktionen, Ressourcenvorkommen, Ressourcenbedarfe und Bearbeitungszeiten seien wie in Beispiel 3.

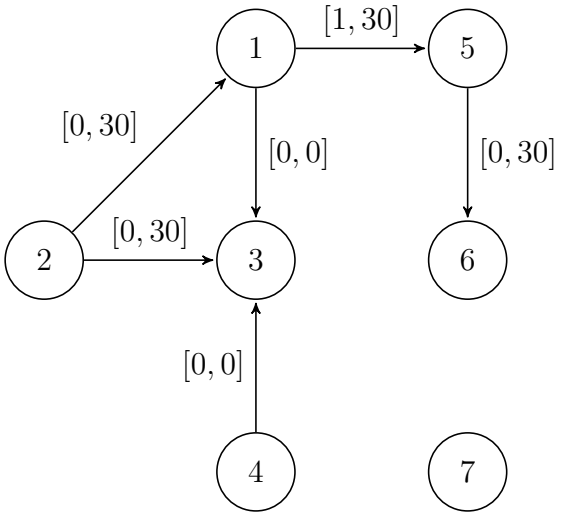


Abbildung 4: Vorgängerrelation zu Beispiel 4

Die Vorgängerrelation wird nun um Verzögerungen ergänzt. Dazu wird diese analog zu den vorigen Beispielen als Graph dargestellt, dessen Kanten durch Beschriftungen der Form $[x, y]$ ergänzt werden, wobei x der minimalen Verzögerung und y der maximalen Verzögerung entspricht. Abbildung 4 zeigt diesen Graph.

Nachdem die neue Vorgängerrelation aus Abbildung 4 hinzugefügt wurde, stellt die Lösung zu Beispiel 3, die in Abbildung 3 zu sehen ist, keinen zulässigen Schedule mehr dar. Zum einen wird in diesem Schedule die Aktion 3 erst mehrere Zeiteinheiten nach Beendigung von Aktivität 1 gestartet, was gegen die maximale Verzögerung zwischen den Aktionen 1 und 3 verstößt. Außerdem wird Aktion 5 unmittelbar nach Beendigung von Aktivität 1 gestartet, was gegen die minimale Verzögerung zwischen den beiden Aktionen verstößt. Abbildung 5 zeigt einen aktualisierten Schedule, der diese Aspekte berücksichtigt.

Weil Aktion 3 unmittelbar nach dem Beenden der Aktionen 1 und 4 gestartet werden muss, müssen die Aktionen 1 und 4 zum gleichen Zeitpunkt aufhören. Aktion 5 wird nun erst eine Zeiteinheit nach Beendigung von Aktion 1 gestartet. Weil die beiden Aktionen 4 und 5 nun nicht mehr parallel ausgeführt werden können, wird in diesem Schedule Aktion 3 in Modus 2 gestartet, damit die Aktionen 3 und 5 parallel ausgeführt werden können.

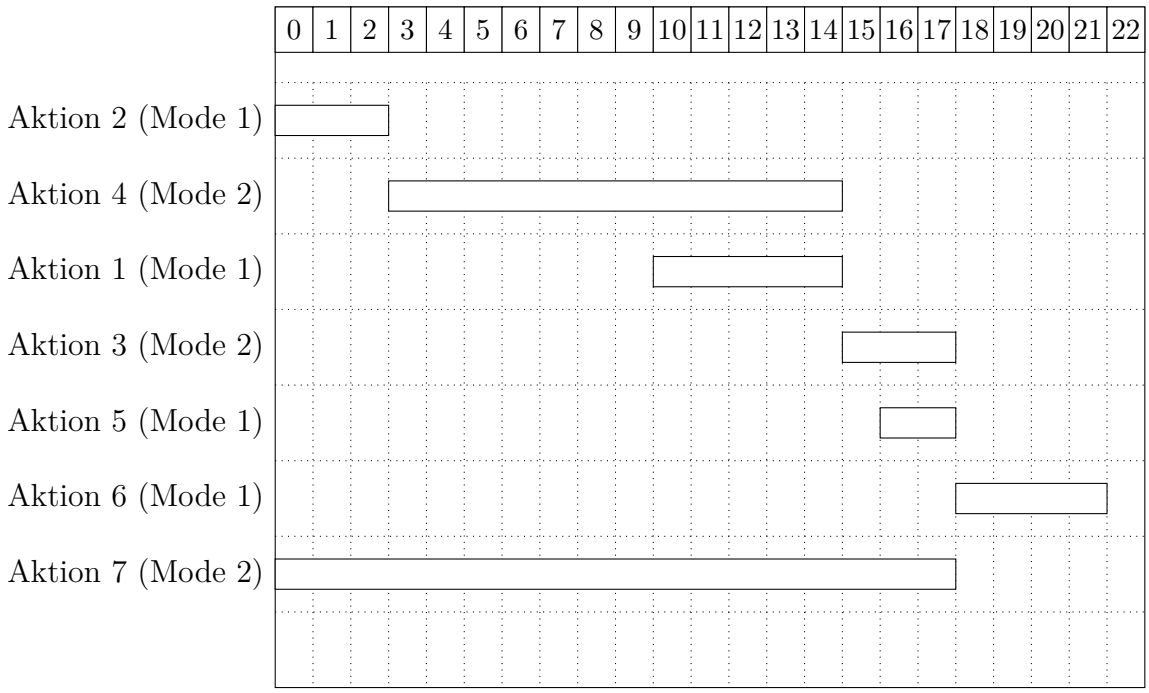


Abbildung 5: Schedule zu Beispiel 3

3 Petri-Netze

Die von Carl Adam Petri in seiner Doktorarbeit entwickelten Petri-Netze (vgl. [14]) sind ein Werkzeug, um nebenläufige Systeme zu modellieren und zu analysieren. [12] bezeichnet sie als graphisches und mathematisch fundiertes Tool, das sowohl in der Praxis, als auch in der Theorie einsetzbar ist. Gerade die Möglichkeit auch parallele Systeme zu modellieren (siehe auch [12]) erscheint in Hinblick auf das Ziel dieser Arbeit ein sehr nützlicher Aspekt zu sein.

Während die ursprünglichen Petri-Netze keine zeitlichen Abhängigkeiten beinhalteten (vgl. [3]), wurden mit der Zeit Erweiterungen dieser entwickelt, welche die originalen Petri-Netze um zeitliche Aspekte ergänzen (siehe zum Beispiel [16]). Da Zeit gerade im Kontext von Scheduling-Problemen eine große Rolle spielt, sind diese Erweiterungen für das Ziel dieser Arbeit besonders relevant.

In diesem Kapitel werden zunächst die klassischen Petri-Netze eingeführt, bevor mit den zeitbehafteten Petri-Netzen eine ausgewählte Erweiterung erläutert wird.

3.1 Klassische Petri-Netze

Wie bereits erwähnt, werden Petri-Netze verwendet, um Systeme zu modellieren, wobei sie es unter anderem ermöglichen, drei Aspekte solcher Systeme zu beschreiben (siehe auch [16]):

1. Statische Struktur (Netze).
2. Zustände (Markierungen).
3. Zustandsübergänge (Schalten).

Im Verlauf dieses Abschnittes werden anhand dieser drei Aspekte die klassischen Petri-Netze formal eingeführt.

3.1.1 Netze

Zunächst wird der erstgenannte Aspekt näher erläutert und die statische Struktur eines Petri-Netzes eingeführt. Diese wird auch *Netz* genannt und im Folgenden nach [17] definiert.

Definition 11. Ein *Netz* ist 3-Tuple $N = (P, T, F)$. Dabei ist

- P die endliche Menge von *Stellen*
- T die endliche Menge von *Transitionen*
- $F \subseteq (P \times T) \cup (T \times P)$ die *Flussrelation*

Für die Mengen P und T gilt dabei $T \cap P = \emptyset$ und $T \cup P \neq \emptyset$

Bei einem Netz handelt es sich um einen gerichteten Graphen, der Knoten in zwei unterschiedlichen „Farben“ enthält, wobei die Mengen P und T die verschieden gefärbten Knoten sind und F die Menge der Kanten ist. Abbildung 6 zeigt ein Beispiel für die graphische Darstellung eines Netzes, in der die Knoten aus P rund und die Knoten aus T eckig dargestellt sind.

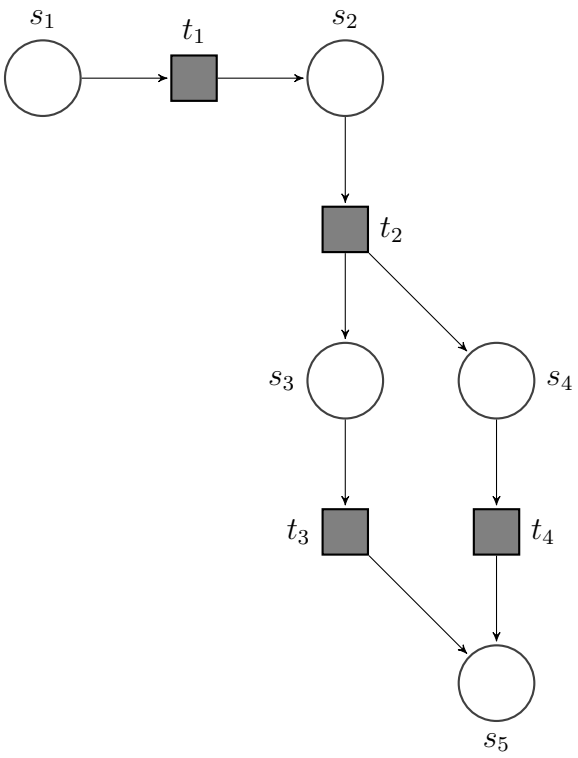


Abbildung 6: Graphische Repräsentation eines Netzes

Nun bleibt noch die Frage offen, wie mit Hilfe von Netzen die statische Struktur eines Systems modelliert werden kann. Um eine Intuition dafür zu bekommen, hilft es, sich das Prinzip der Dualität von Petri-Netzen vor Augen zu führen (vgl. [6]). Dieses Prinzip besagt, dass passive Elemente der realen Welt (wie zum Beispiel Ressourcen und Bedingungen) mit Hilfe von Stellen dargestellt werden, während aktive Elemente der realen Welt (wie zum Beispiel Aktionen und Events) mit Hilfe von Transitionen dargestellt werden.

Nehmen wir beispielsweise an, dass mit Stellen Ressourcen und mit Transitionen Aktionen modelliert werden, dann bedeutet dies für Abbildung 6, dass die Ressource, welche mit Stelle s_1 modelliert wird, notwendig ist, um die Aktion, welche durch Transition t_1 modelliert wird, auszuführen. Durch das Ausführen jener Aktion wird wiederum die Ressource verfügbar, die durch Stelle s_2 repräsentiert wird. Diese Intuition wird in Abschnitt 3.1.3 formalisiert werden. Zu beachten ist dabei, dass die Modellierung von Ressourcen und Aktionen lediglich ein Beispiel für die Anwendung von Netzen (bzw. auch Petri-Netzen) darstellt.

Vor diesem Hintergrund lässt sich auch verstehen, warum Transitionen nur Kanten zu Stellen haben und umgekehrt. Grob beschrieben müssen Bedingungen erfüllt sein, damit Aktionen durchgeführt werden können. Umgekehrt werden Bedingungen durch das Ausführen von Aktionen erfüllt.

Eine Erweiterung der Netze ist es, zusätzlich zur Netz-Struktur eine Gewichtungsfunktion für die Flussrelation anzugeben. Dahinter steckt die Idee, dass es zum Beispiel durchaus nützlich sein kann zu modellieren, dass eine Aktion mehrere Einheiten einer bestimmten

Ressource benötigt bzw. produziert. Diese Erweiterung nennt sich *gewichtetes Netz* und wird im Folgenden angelehnt an [6] definiert.

Definition 12. Ein *gewichtetes Netz* ist ein 4-Tuple $N = (P, T, F, W)$ für das gilt:

- (P, T, F) ist ein Netz.
- $W \colon F \to \mathbb{N}_{>0}$ ist eine (Gewichtungs-) Funktion

Für die graphische Darstellung hat dies zur Auswirkung, dass nun ein zweifach gefärbter, gerichteter und gewichteter Graph vorliegt. Abbildung 7 zeigt die graphische Darstellung eines gewichteten Netzes. Dabei wird für nicht beschriftete Kanten eine implizite Gewichtung von 1 angenommen.

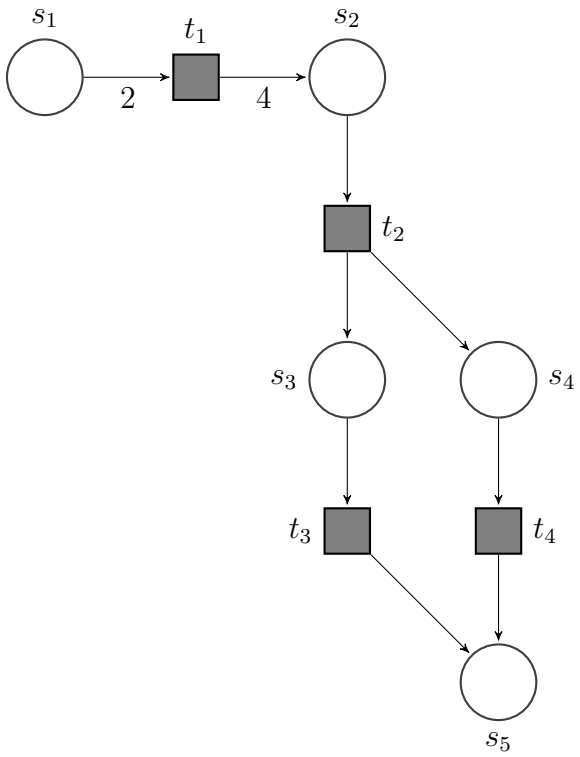


Abbildung 7: Graphische Repräsentation eines gewichteten Netzes

Nehmen wir wieder an, dass Stellen Ressourcen und Transitionen Aktionen modellieren, dann bedeutet dies, dass jetzt die Aktion, welche durch Transition t_1 modelliert wird, zwei Einheiten der Ressource, die durch s_1 repräsentiert wird, benötigt und vier Einheiten der Ressource, welche durch Stelle s_2 modelliert wird, produziert.

3.1.2 Markierungen

Mit Hilfe von gewichteten Netzen ist es also möglich, die statische Struktur eines Systems zu beschreiben. Wir haben zum Beispiel gesehen, dass es möglich ist zu modellieren, welche

Aktionen und Ressourcen existieren und welche Zusammenhänge zwischen ihnen bestehen. Dabei war es allerdings nicht möglich zu modellieren, wie viele Einheiten der Ressourcen, die mit Hilfe von Stellen modelliert werden, vorliegen.

Um diese Lücke zu füllen, werden nun Token eingeführt. Jede Stelle kann eine Anzahl von Token enthalten. Dabei realisiert die Anzahl der Token pro Stelle die Zustände eines gewichteten Netzes, welche auch *Markierungen* genannt werden. Formal ist eine solche Markierung eine Abbildung, die jeder Stelle eine Anzahl an Token zuweist (vgl. [16]).

Definition 13. Sei $N = (P, T, F, W)$ ein gewichtetes Netz. Dann ist eine Markierung von N eine Abbildung $m\colon P \to \mathbb{N}$.

Ein gewichtetes Netz mit einer *initialen Markierung* ist ein Petri-Netz (vgl. [16]).

Definition 14. Ein *Petri-Netz* ist ein 5-Tuple $PN = (P, T, F, W, m_0)$ mit

- $N = (P, T, F, W)$ ist ein gewichtetes Netz
- $m_0\colon P \to \mathbb{N}$ ist die *initiale Markierung* von PN (auch *Anfangsmarkierung* genannt).

Auch die graphische Repräsentation von gewichteten Netzen kann um Markierungen erweitert werden, wobei die Token einer Stelle durch schwarze „Punkte“ in eben dieser repräsentiert werden, wie Abbildung 8 exemplarisch darstellt.

Kehrt man zum Beispiel aus den Abbildungen 6 und 7 zurück, dann ist es mit Markierungen möglich zu modellieren, wie viele Einheiten einer Ressource vorhanden sind. So wären in der Markierung, die in Abbildung 8 zu sehen ist, zum Beispiel 8 Einheiten der Ressource, die durch Stelle s_1 repräsentiert wird, verfügbar.

Eine weitere Möglichkeit Markierungen eines Petri-Netzes zu notieren ist es, ohne Beschränkung der Allgemeinheit anzunehmen, dass die Stellen des Petri-Netzes durchnummeriert sind und die Markierungen als Vektor zu notieren. Der Vektor repräsentiert dabei die Stellen des Petri-Netzes und die Einträge enthalten die Anzahl der Token in den jeweiligen Stellen. Im Beispiel aus Abbildung 8 hat die erste Stelle acht, die dritte zwei und alle andere Stellen haben keine Token, deshalb wäre die Markierung $M = (8, 0, 2, 0, 0)$.

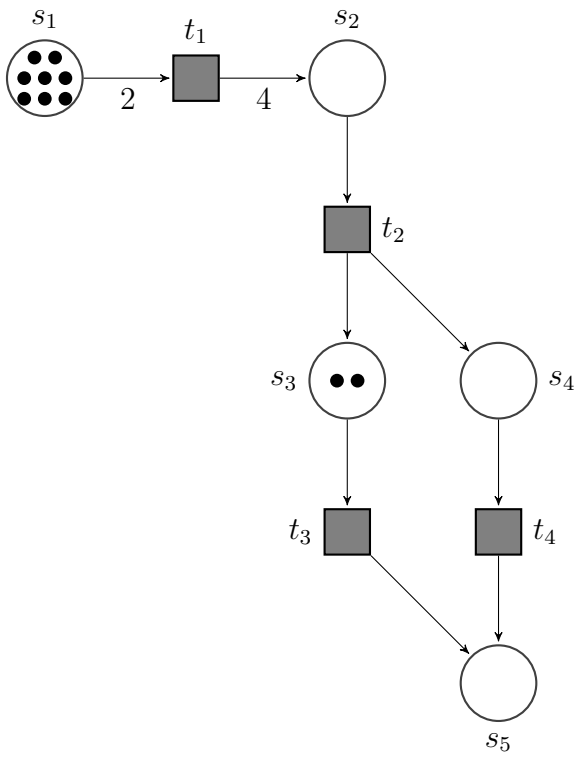


Abbildung 8: Graphische Repräsentation eines *Petri-Netz*

3.1.3 Schalten

Im vorigen Abschnitt wurde ein Petri-Netz als gewichtetes Netz mit initialer Markierung eingeführt. Dieser Abschnitt widmet sich nun dem dynamischen Verhalten von Petri-Netzen. Das dynamische Verhalten von Petri-Netzen wird realisiert, indem Markierungen unter bestimmten Regeln in Nachfolgemarkierungen überführt werden können. Ausgehend von der initialen Markierung kann so das dynamische Verhalten eines Systems aus der realen Welt modelliert werden.

Gemäß dem Prinzip der Dualität [6] werden aktive Elemente der realen Welt mit Hilfe der Transitionen modelliert. Dementsprechend ist auch das dynamische Verhalten von Petri-Netzen an die Transitionen geknüpft. Diese können nach bestimmten Regeln *schalten* und verändern dabei die Markierung des Petri-Netzes. Bevor im Folgenden erwähnte Regeln definiert werden, werden zunächst einige Notationen eingeführt, die dafür erforderlich sind. Wie wir im weiteren Verlauf dieses Abschnittes sehen werden, verändert eine Transition nur die Stellen, mit denen sie direkt über eine Kante verbunden ist (siehe auch „Prinzip der Lokalität“ in [6]). Die Auswirkungen des Schaltens unterscheiden sich dabei für Stellen, die eine eingehende Kante von der Transition haben, und Stellen, die eine ausgehende Kante zur Transition haben. Erstere werden als *Vorstellen* und Letztere als *Nachstellen* einer Transition bezeichnet (vgl. [17]). Die folgende Definition führt Vorstellen und Nachstellen formal nach [17] ein. Die restlichen Aspekte des Schaltens werden orientiert an [16] eingeführt.

Definition 15. Seien $PN = (P, T, F, W, m_0)$ ein Petri-Netz und $t \in T$ eine Transition. Dann heißt:

- ${}^\bullet t := \{p \in P \mid (p,t) \in F\}$ die Menge der *Vorstellen* von t.
- $t^\bullet := \{p \in P \mid (t,p) \in F\}$ die Menge der *Nachstellen* von t.

Wenn nun eine Transition schaltet, dann entfernt sie Token aus ihren Vorstellen und fügt ihren Nachstellen Token hinzu. Die Anzahl der Token, die entfernt bzw. hinzugefügt werden, entspricht dabei den jeweiligen Kantengewichten. Um die Veränderung einer Markierung nach dem Schalten einer Transition besser beschreiben zu können, werden nun einige Abbildungen eingeführt.

Definition 16. Seien $PN = (P, T, F, W, m_0)$ ein Petri-Netz und $t \in T$ eine Transition. Dann seien t^-, t^+ und Δt definiert durch:

- $t^-\colon P \to \mathbb{N}_0$ mit $t^-(p) := \begin{cases} W(p,t) & \text{, falls } (t,p) \in F \\ 0 & \text{sonst} \end{cases}$ für alle $p \in P$.
- $t^+\colon P \to \mathbb{N}_0$ mit $t^+(p) := \begin{cases} W(t,p) & \text{, falls } (p,t) \in F \\ 0 & \text{sonst} \end{cases}$ für alle $p \in P$.
- $\Delta t\colon P \to \mathbb{N}_0$ mit $\Delta t(p) := t^+(p) - t^-(p)$ für alle $p \in P$.

Mit Hilfe der definierten Funktionen lassen sich nun Schaltregeln und Schaltverhalten von Petri-Netzen definieren. Dabei entspricht die Schaltregel der Intuition, dass eine Transition nur schalten kann, wenn ausreichend Token in ihren Vorstellen vorhanden sind.

Definition 17. Seien $PN = (P, T, F, W, m_0)$ ein Petri-Netz, m eine Markierung dieses Netzes und $t \in T$ eine Transition. t heißt *m-aktiviert* (Notation: $m\ [t >$), wenn $t^- \leq m$.[4]

Betrachtet man zum Beispiel die Markierung m, welche in Abbildung 8 zu sehen ist, dann gilt $m\ [t_1 >$, denn t_1 hat genau die Vorstelle s_1 und es gilt $t_1^-(s_1) = 2 \leq 8 = m(s_1)$. Es gilt aber nicht $m\ [t_2 >$, denn $t_2^-(s_2) = 1 \not\leq 0 = m(s_2)$.

Eine Transition darf also in einer Markierung m eines Petri-Netzes PN schalten, wenn sie m-aktiviert ist, wobei das Schalten PN in eine neue Markierung überführt. Entsprechend des Prinzips der Lokalität verändert sich dabei nur die Zahl der Token in Vor- und Nachstellen der Transition. Die folgende Definition konkretisiert dies.

Definition 18. Seien $PN = (P, T, F, W, m_0)$ ein Petri-Netz, m eine Markierung dieses Netzes und $t \in T$ eine Transition mit $m[t >$, dann kann t von m nach $m' = \Delta t + m$ schalten (Notation $m\ [t > m'$).[5]

[4] Dabei gelte $a \leq b$ für zwei Abbildungen $a, b\colon D \to \mathbb{N}$ genau dann, wenn $a(d) \leq b(d)$ für alle $d \in D$. Die anderen Vergleichsoperationen zwischen Abbildungen seien analog definiert.

[5] Dabei sei die Summe c zweier Abbildungen $a, b\colon D \to \mathbb{N}$ definiert als $c\colon D \to \mathbb{N}$ mit $c(d) = a(d) + b(d)$ für alle $d \in D$.

Für die Markierung m, die in Abbildung 8 zu sehen ist, gilt $m\ [t_1 >$, weshalb t_1 in m schalten kann. Abbildung 9 zeigt Markierung m', die durch das Schalten von Transition t_1 entstanden ist.

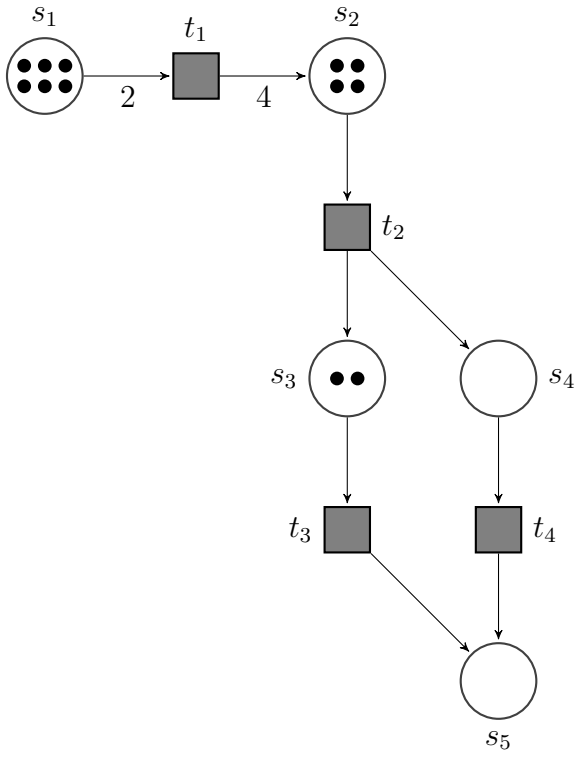


Abbildung 9: Das Petri-Netz aus Abbildung 8 nach dem Schalten der Transition t_1

Wir haben also gesehen, unter welchen Bedingungen ein Petri-Netz eine Markierung in eine Folgemarkierung überführen kann. Führt man sich erneut vor Augen, dass die Markierungen von Petri-Netzen zur Modellierung von Systemen dienen, dann erscheint die Frage, ob ein bestimmter Zustand dieses Systems auftreten kann, höchst relevant zu sein. Gehen wir zum Beispiel davon aus, dass ein Softwaresystem modelliert wird, dann ist es wünschenswert zu wissen, ob eine Markierung, die einen fehlerhaften Zustand repräsentiert, auftreten kann.

Wie wir später bei der Modellierung eines RCPSP sehen werden, existiert dort eine Markierung, die repräsentiert, dass jenes vollständig abgeschlossen wurde. In dem Zusammenhang ist dann ebenfalls von Interesse, ob diese Markierung auftreten kann.

Eine zentrale Frage im Zusammenhang der Analyse von Petri-Netzen ist also die, ob eine gegebene Markierung m in einem Petri-Netz PN ausgehend von der Anfangsmarkierung *erreichbar* ist. Die folgenden Definitionen dienen dazu, den Begriff der Erreichbarkeit zu präzisieren. Zunächst wird der Begriff der Schaltfolge induktiv definiert.

Definition 19. Seien $PN = (P, T, F, W, m_0)$ ein Petri-Netz, m, m' Markierungen von PN und $w = t_1 \cdots t_n$ eine Folge von Transitionen. Dann heißt w eine Schaltfolge von m nach m' (Notation: $m[t_1 \cdots t_n > m'$), falls gilt:

- $m' = m$ für $n = 0$
- $m\ [t_1 \cdots t_{n-1} > m''\ [t_n > m'$ für eine Markierung m'' und $n > 0$.

In den Abbildungen 8 und 9 haben wir bereits gesehen, dass $m_0\ [t_1 > m'$ für die Markierung $m' = (6, 4, 2, 0, 0)$ gilt. Außerdem sieht man leicht, dass $m'\ [t_2 > m''$ für die Markierung $m'' = (6, 3, 3, 1, 0)$ gilt. Demnach gilt auch $m_0\ [t_1 t_2 > m''$.

Mit Hilfe von Definition 19 lässt sich nun der Begriff der Erreichbarkeit definieren.

Definition 20. Seien $PN = (P, T, F, W, m_0)$ ein Petri-Netz und $m, m' \colon P \to \mathbb{N}$ Markierungen von PN. Die Markierung m' heißt *erreichbar ausgehend von* m, falls $\exists w \in T^* : m\ [w > m'$. [6]

Die Menge aller Markierungen von PN, die ausgehend von m erreichbar sind, ist definiert als $R_{PN}(m) = \{m' \mid m\ [* >\ m'\}$. [7]

Die Menge aller Markierungen, die in PN erreichbar sind, ist definiert als $R(PN) = R_{PN}(m_0)$.

3.2 Petri-Netze und Zeit

Das Ziel dieser Arbeit ist es, RCPSPs mit Hilfe von Petri-Netzen zu modellieren, sodass diese zur Analyse des ursprünglichen Scheduling-Problems verwendet werden können. Wir haben gesehen, dass ein Anwendungsfall von Petri-Netzen ist, Ressourcen und Aktionen zu modellieren. Wie der weitere Verlauf der Arbeit zeigen wird, lassen sich die bisher eingeführten Konzepte auch dazu verwenden, Ressourcen-bedingte Einschränkungen sowie bestimmte Reihenfolgen zwischen Aktionen zu modellieren. Außerdem kann das Erreichbarkeits-Problem zur Analyse herangezogen werden.

Andere Aspekte von Scheduling-Problemen lassen sich mit Hilfe der bereits eingeführten Petri-Netze nicht ohne Weiteres modellieren. So gibt es zum Beispiel keine Möglichkeit das Schalten einer Transition zeitlich zu beschränken, um die Bearbeitungszeiten in Scheduling-Problemen zu modellieren. Zudem ist nicht offensichtlich, ob und wie Verzögerungen modelliert und mögliche Startzeitpunkte analysiert werden können. Aus diesen Gründen wird in diesem Kapitel eine Möglichkeit eingeführt, klassische Petri-Netze um zeitliche Abhängigkeiten zu erweitern.

Eine erste intuitive Möglichkeit, Zeit in das Petri-Netz-Modell mit einzubeziehen ist es, jeder Transition eine Dauer zuzuweisen, die sie benötigt, um zu schalten. Dieses Prinzip wird zum Beispiel in *Timed Petri-Nets* verwendet (vgl. [16]). Eine weitere Möglichkeit ist es, jeder Transition ein Zeitintervall zuzuweisen, in dem die Transition schalten muss. Dieses Prinzip wird zum Beispiel in *zeitbehafteten Petri-Netzen*[8] verwendet (vgl. [16]). Auf dem ersten Blick erscheint ersteres Prinzip möglicherweise besser geeignet zu sein, um Scheduling-Probleme zu modellieren, weil sich die Dauer einer Transition relativ leicht mit der Bearbeitungszeit

[6] $w \in T^*$ bedeutet, dass w eine beliebige (möglicherweise leere) Konkatenation von Transitionen aus T ist.
[7] $*$ ist in diesem Kontext eine beliebige (möglicherweise leere) Konkatenation von Transitionen.
[8] Im Englischen *Time Petri-Nets*

einer Aktion in Verbindung bringen lässt. Im Prinzip sind allerdings beide Modelle geeignet, um Scheduling-Probleme zu modellieren. Tatsächlich haben beide Modelle die gleiche Ausdrucksmächtigkeit und sind jeweils Turingmächtig (vgl. [16]).

Wie allerdings bereits in der Einleitung erwähnt wurde, existieren einige Analysetechniken für zeitbehaftete Petri-Netze (siehe zum Beispiel [16] und [4]), die auch für die Analyse von Scheduling-Problemen genutzt werden könnten. Um diese Anwendung zu ermöglichen, werden in dieser Arbeit zeitbehaftete Petri-Netze zur Modellierung verwendet. Diese werden im Folgenden orientiert an [16] eingeführt.

3.2.1 Zeitbehaftete Petri-Netze

Zeitbehaftete Petri-Netze erweitern Petri-Netze nach Definition 14 , indem für jede Transition t ein Intervall $[a_t, b_t]$ angegeben. Die Bedeutung dessen ist, dass eine Transition frühestens a_t Zeiteinheiten nachdem sie aktiviert wurde schalten kann und spätestens b_t Zeiteinheiten nach Aktivierung schalten muss, sofern sie dann immer noch aktiviert ist. Die formale Definition nach [16] lautet wie folgt.

Definition 21. Ein zeitbehaftetes Petri-Netz ist ein 6-Tupel $Z = (P, T, F, W, m_0, I)$, sodass gilt:

- das Tupel $S(Z) = (P, T, F, W, m_0)$ ist ein Petri-Netz und wird *Skelett* von Z genannt.
- $I : T \to \mathbb{N}_0 \times (\mathbb{N}_0 \cup \infty)$, sodass für alle $t \in T$ gilt, dass $I(t) = (\text{eft}(t), \text{lft}(t)) \Rightarrow \text{eft}(t) \leq \text{lft}(t)$ [9].

Definition 14 wurde also um eine Abbildung ergänzt, die jeder Transition die Grenzen eines Zeitintervalls zuweist, wobei diese Grenzen natürliche Zahlen sind bzw. im Falle der oberen Grenze eine natürliche Zahl oder positiv Unendlich. Wie bereits angemerkt, muss eine Transition schalten, wenn die Zeiteinheiten entsprechend der oberen Intervall-Grenze vergangen sind. Soll für eine Transition keine solche Restriktion existieren, kann positiv Unendlich als hintere Intervall-Grenze gewählt werden.

Abbildung 10 zeigt die graphische Darstellung eines zeitbehafteten Petri-Netzes, wobei die Zeitintervalle der Transitionen in eckigen Klammern angegeben sind.

Die Intervallgrenzen könnten dabei prinzipiell auch als rationale Zahlen vorliegen. Laut [16] haben zeitbehaftete Petri-Netze, die nur natürliche Zahlen (sowie positiv Unendlich) als Grenzen der Intervalle verwenden, allerdings die gleiche Ausdrucksmächtigkeit wie solche, die rationale Zahlen als Intervallgrenzen verwenden. Weil deshalb einige Analyseverfahren für zeitbehaftete Petri-Netze nur natürliche Zahlen als Intervallgrenzen einbeziehen, werden in dieser Arbeit nur zeitbehaftete Petri-Netze betrachtet, die Intervall-Funktionen der Form $I : T \to \mathbb{N}_0 \times (\mathbb{N}_0 \cup \infty)$ haben.

Die bisher eingeführten Definitionen betreffen die statischen Aspekte von zeitbehafteten Petri-Netzen. Um deren Verhalten vollständig zu definieren, werden analog zu Abschnitt 3, Markierungen und Schaltregeln eingeführt.

[9] eft und lft sind dabei die Abkürzungen für *earliest firing time* bzw. *latest firing time*

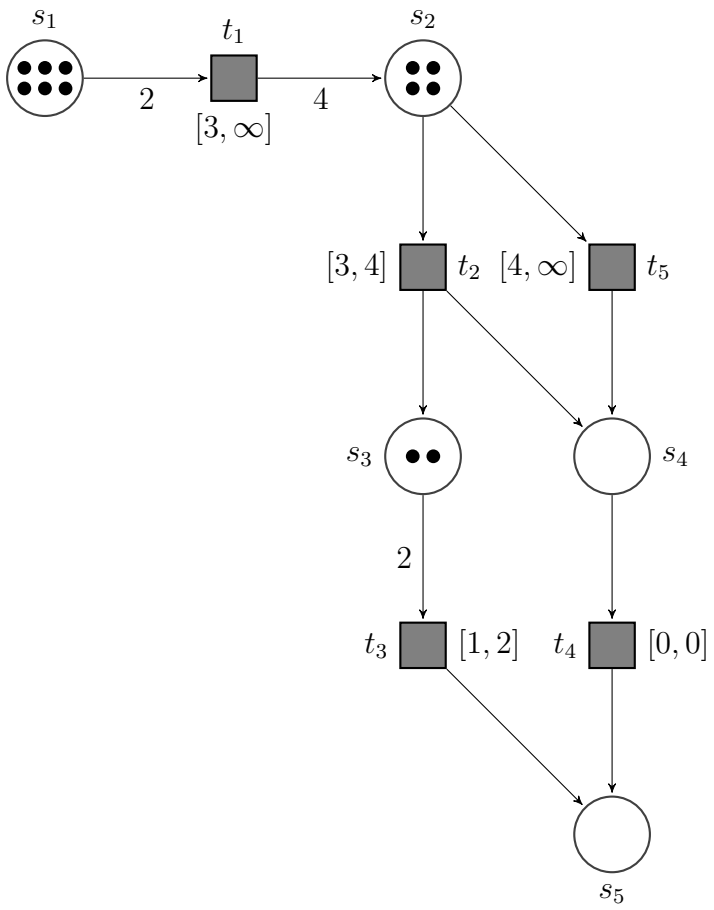


Abbildung 10: Graphische Darstellung eines zeitbehafteten Petri-Netzes

3.2.2 Markierungen und Zustände

Zeitbehaftete Petri-Netze unterscheiden sich von klassischen Petri-Netzen dadurch, dass jede Transition nach ihrer Aktivierung nur in einem gegebenen Zeitintervall schalten kann und sogar schalten muss. Der Zustand eines zeitbehafteten Petri-Netzes ist also nicht mehr alleine durch die Verteilung der Token auf die Stellen definiert, sondern vielmehr muss für jede Transition bekannt sein, wie viel Zeit seit Aktivierung dieser Transition vergangen ist. Zu diesem Zweck unterscheidet man bei zeitbehafteten Petri-Netzen zwischen *p-Markierungen* und *t-Markierungen*, wobei *p*-Markierungen den Markierungen von klassischen Petri-Netzen aus Definition 13 entsprechen. Im Folgenden werden beide Arten von Markierungen und *Zustände* nach [16] definiert.

Definition 22. Sei $Z = (P, T, F, W, m_0, I)$ ein zeitbehaftetes Petri-Netz. Eine *p-Markierung* von Z ist eine Abbildung $m : P \to \mathbb{N}$.

Während *p*-Markierungen wie gehabt angeben, wie viele Token in den einzelnen Stellen vorhanden sind, weisen *t*-Markierungen jeder Transition die Zeit zu, die seit Aktivierung der Transition vergangen ist.

Definition 23. Sei $Z = (P, T, F, W, m_0, I)$ ein zeitbehaftetes Petri-Netz. Eine *t-Markierung* von Z ist eine Abbildung $h : T \to \mathbb{R}_0^+ \cup \{\#\}$.

Zu beachten ist dabei, dass auch Transitionen vorliegen können, die nicht aktiviert sind. Für diese existiert dann keine Zeit, die seit ihrer Aktivierung vergangen ist, deshalb wird der Wert # für solche Transitionen verwendet.

Der Zustand eines zeitbehafteten Petri-Netzes setzt sich aus einer *p*- sowie einer *t*-Markierung zusammen.

Definition 24. Seien $Z = (P, T, F, W, m_0, I)$ ein zeitbehaftetes Petri-Netz, m eine p-Markierung und h eine t-Markierung von Z. Ein Tupel (m, h) ist ein *Zustand* von Z, wenn gilt:

1. $(t^- \not\leq m) \Rightarrow h(t) = \#$ für alle $t \in T$
2. $(t^- \leq m) \Rightarrow (h(t) \in \mathbb{R}_0^+ \wedge h(t) \leq lft(t))$ für alle $t \in T$

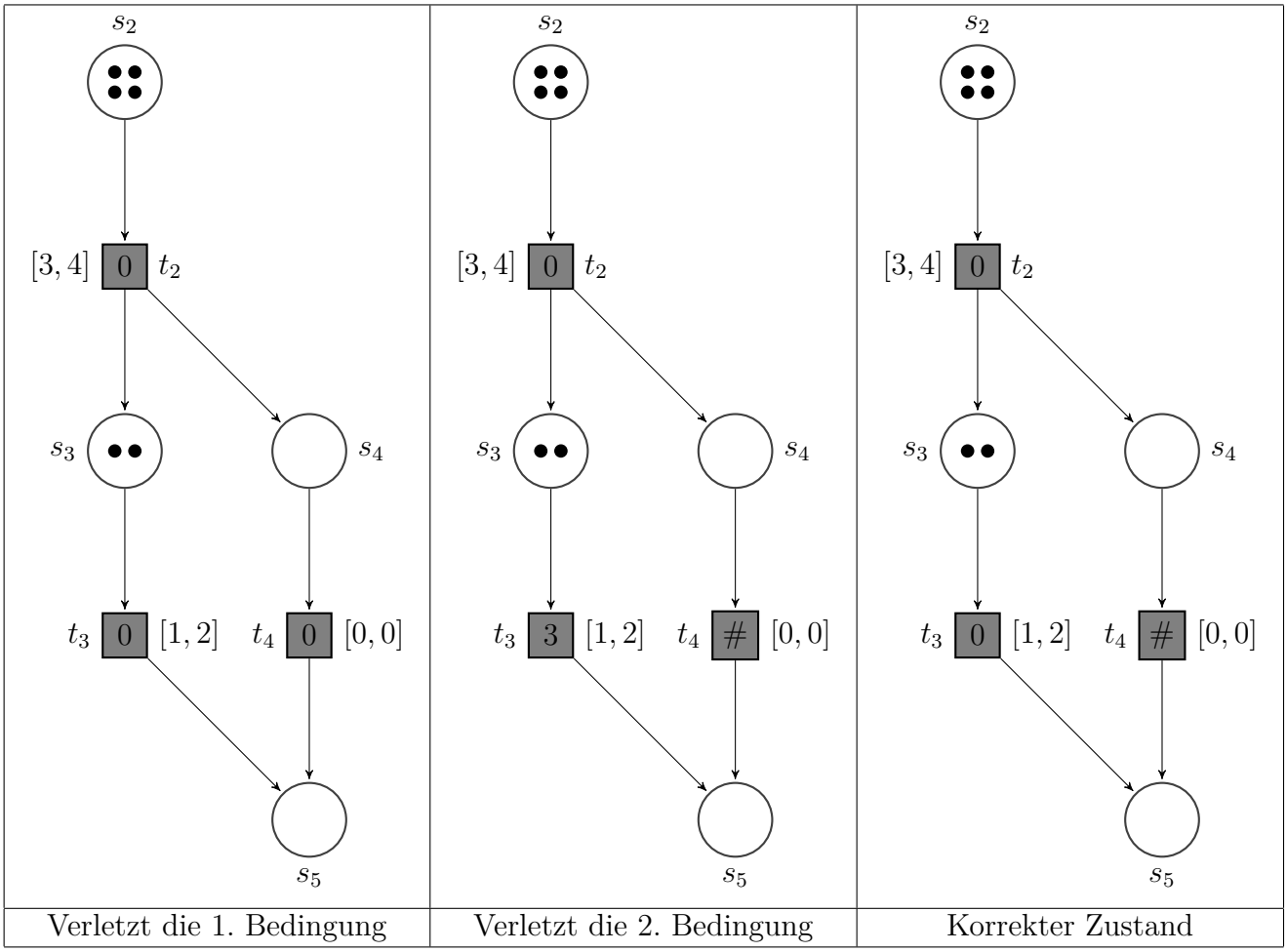
Definition 24 besagt, dass ein solches Tupel aus *p*- und *t*-Markierung nur dann einen Zustand darstellt, wenn es zwei Einschränkungen erfüllt. Erstere bezieht ein, dass für nicht aktivierte Transitionen keine Anzahl von Zeiteinheiten existiert, die seit Aktivierung der Transitionen vergangen ist, weshalb die *t*-Markierung sie auf den Wert # abbilden muss. Die zweite Bedingung sorgt dafür, dass keine Zustände existieren, in denen eine Transition innerhalb ihres Zeitintervalls nicht geschaltet hat.

Der *initiale Zustand* eines zeitbehafteten Petri-Netzes sei dabei wie folgt definiert:

Definition 25. Sei $Z = (P, T, F, W, m_0, I)$ ein zeitbehaftetes Petri-Netz, dann ist der *initiale Zustand* z_0 von Z definiert als:

$$z_0 = (m_0, h_0) \text{ mit } h_0(t) = \begin{cases} 0 & , \text{ falls } t^- \leq m_0 \\ \# & , \text{ falls } t^- \not\leq m_0 \end{cases} \quad \forall t \in T$$

Abbildung 11 zeigt einige graphische Darstellungen von zeitbehafteten Petri-Netzen. Die Beschriftung innerhalb der Transitionen sei dabei die Darstellung der *t*-Markierung im aktuellen Zustand. Die linke graphische Darstellung ist dabei kein Zustand eines zeitbehafteten Petri-Netzes, denn t_4 ist nicht aktiviert und müsste daher mit # beschriftet sein. Die mittlere graphische Darstellung ist ebenfalls kein Zustand eines zeitbehafteten Petri-Netzes, denn $\text{lft}(t_3) < h(t_3)$. Die rechte Darstellung zeigt einen Zustand eines zeitbehafteten Petri-Netzes.



Verletzt die 1. Bedingung	Verletzt die 2. Bedingung	Korrekter Zustand

Abbildung 11: „Legale“ und „illegale“ Zustände eines zeitbehafteten Petri-Netzes.

3.2.3 Schalten und Ablauf von Zeit

Nachdem Zustände von zeitbehafteten Petri-Netzen definiert wurden, wird in diesem Abschnitt ihr dynamisches Verhalten thematisiert. Während klassische Petri-Netze ihre Markierungen ausschließlich durch das Schalten von Transitionen verändern, verändert sich der Zustand eines zeitbehafteten Petri-Netzes zusätzlich auch dadurch, dass Zeit vergeht.

Zunächst betrachten wir allerdings das Schalten von Transitionen. In klassischen Petri-Netzen kann eine Transition t in einer Markierung m schalten, wenn sie m-aktiviert ist. Auch in zeitbehafeten Petri-Netzen ist m-Aktivierung identisch definiert.

Definition 26. Sei $Z = (P, T, F, W, m_0, I)$ ein zeitbehaftetes Petri-Netz im Zustand $z = (m, h)$. Eine Transition $t \in T$ ist *m-aktiviert* in Z, wenn sie m-aktiviert in $S(Z) = (P, T, F, W, m_0)$ ist.

Damit eine Transition schalten kann, muss neben der Aktivierung auch eine zeitliche Bedingung erfüllt sein. Wie bereits angedeutet wurde, darf eine Transition t nur schalten, wenn seit ihrer Aktivierung mindestens $\text{eft}(t)$ Zeiteinheiten vergangen sind. Im Folgenden wird

definiert wann eine Transition eines zeitbehafteten Petri-Netz *bereit zu schalten* ist (siehe auch [16]).

Definition 27. Sei $Z = (P, T, F, W, m_0, I)$ ein zeitbehaftetes Petri-Netz im Zustand $z = (m, h)$. Eine Transition $t \in T$ ist *bereit zu schalten* (Notation: $z \xrightarrow{t}$), wenn gilt:

- t ist m-aktiviert in Z.
- $h(t) \geq \text{eft}(t)$.

Abbildung 12 zeigt den Zustand $z = (m, h)$ eines zeitbehafteten Petri-Netzes. Dabei ist t_1 nicht bereit zu schalten, denn t_1 ist zwar m-aktiviert, allerdings gilt $h(t_1) < eft(t_1)$. t_4 ist nicht bereit zu schalten, weil sie nicht m-aktiviert ist. t_2, t_3 und t_5 sind dagegen bereit zu schalten, denn sie sind m-aktiviert im Skelett des zeitbehafteten Petri-Netzes und es gilt $h(t_2) \geq eft(t_2)$, $h(t_3) \geq eft(t_3)$ und $h(t_5) \geq eft(t_5)$.

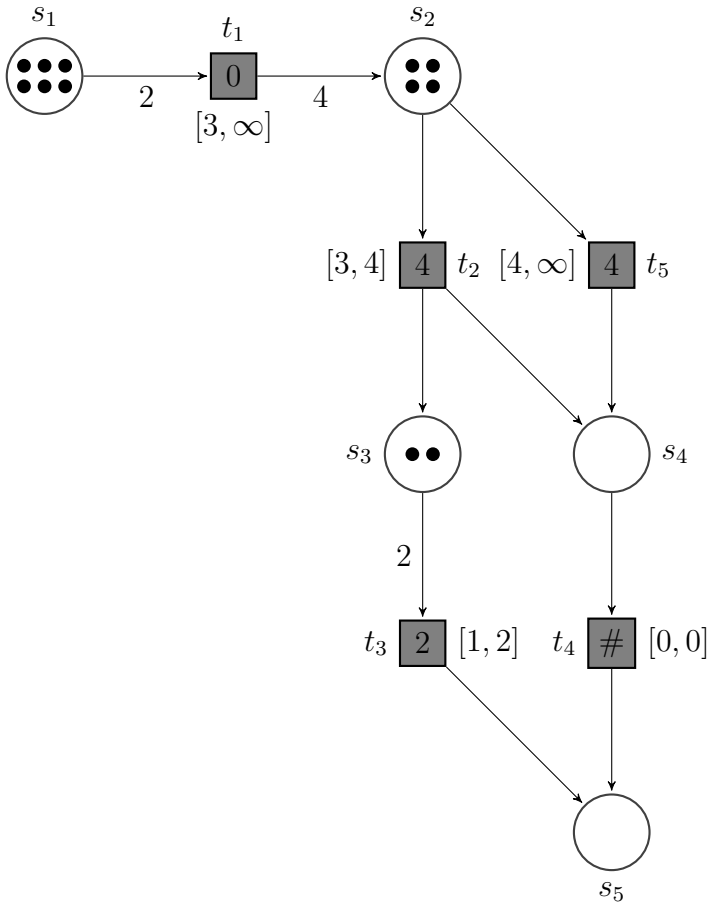


Abbildung 12: Graphische Darstellung eines zeitbehafteten Petri-Netzes

Mit Hilfe von Definition 27 kann nun das Schalten einer Transition in einem zeitbehafteten Petri-Netz definiert werden(vgl. [16]). Dabei verändert sich die p-Markierung identisch

zum Schalten von klassischen Petri-Netzen. Auch die t-Markierung wird durch das Schalten aktualisiert. Laut Definition 24 müssen in einem Zustand eines zeitbehafteten Petri-Netz genau die Transitionen, die nicht aktiviert sind, auf den Wert # abgebildet werden, weshalb Transitionen, die ihre Aktivierung durch das Schalten verlieren, im neuen Zustand auf # abgebildet werden. Ebenso dürfen die Transitionen, die durch das Schalten aktiviert werden, nicht mehr auf # abgebildet werden, da nun null Zeiteinheit seit ihrer Aktivierung vergangen sind. Zusätzlich wird der Wert der Transition, die geschaltet hat, sowie der Transitionen, die gemeinsame Vorstellen mit jener haben, auf 0 zurückgesetzt, sollten diese vor und nach dem Schalten aktiviert sein. Alle andere Transitionen behalten den Wert, den sie bereits vor dem Schalten hatten.

Definition 28. Seien $Z = (P, T, F, W, m_0, I)$ ein zeitbehaftetes Petri-Netz, $\hat{t} \in T$ eine Transition und $z = (m, h)$ ein Zustand in Z. Dann kann $\hat{t}$ in z schalten, wenn gilt $z \xrightarrow{\hat{t}}$. Nach dem Schalten von $\hat{t}$ verändert sich der Zustand von Z zu $z' = (m', h')$ (Notation: $z \xrightarrow{\hat{t}} z'$), mit

- $m' = m + \Delta\hat{t}$
- $\forall t \in T : h'(t) = \begin{cases} \# & , \text{ falls } t^- \not\leq m' \\ h(t) & , \text{ falls } t^- \leq m \wedge t^- \leq m' \wedge^\bullet t \cap^\bullet \hat{t} = \emptyset \wedge t \neq \hat{t} \\ 0 & \text{sonst} \end{cases}$

Abbildung 13 zeigt die zeitbehafteten Petri-Netze TPN_1, TPN_2 und TPN_3. Dabei ist TPN_2 durch Schalten der Transition t_3 aus TPN_1 entstanden. TPN_3 ist durch Schalten der Transition t_2 aus TPN_2 entstanden.

Betrachtet man TPN_1 und TPN_2 fällt auf, dass t_3 nach dem Schalten nicht mehr aktiviert ist, deshalb bekommt es den Wert # zugewiesen. t_4 ist nach dem Schalten nicht aktiviert und bekommt deshalb ebenfalls den Wert # zugewiesen. Die Transitionen t_2 und t_5 waren vor dem Schalten von t_3 aktiviert, sind nach dem Schalten von t_3 aktiviert und haben keine gemeinsamen Vorstellen mit t_3, deshalb behalten sie nach dem Schalten von t_3 ihren Wert. Betrachtet man TPN_2 und TPN_3 bemerkt man, dass t_2 und t_5 vor und nach dem Schalten von t_2 aktiviert sind. Weil t_2 die Transition ist, die geschaltet hat, und t_5 gemeinsame Vorstellen mit t_2 hat, bekommen t_2 und t_5 nach dem Schalten den Wert 0 zugewiesen. t_3 ist nach dem Schalten von t_2 nicht aktiviert und bekommt deshalb den Wert # zugewiesen. t_4 ist erst durch das Schalten von t_2 aktiviert worden und bekommt deshalb den Wert 0 zugewiesen.

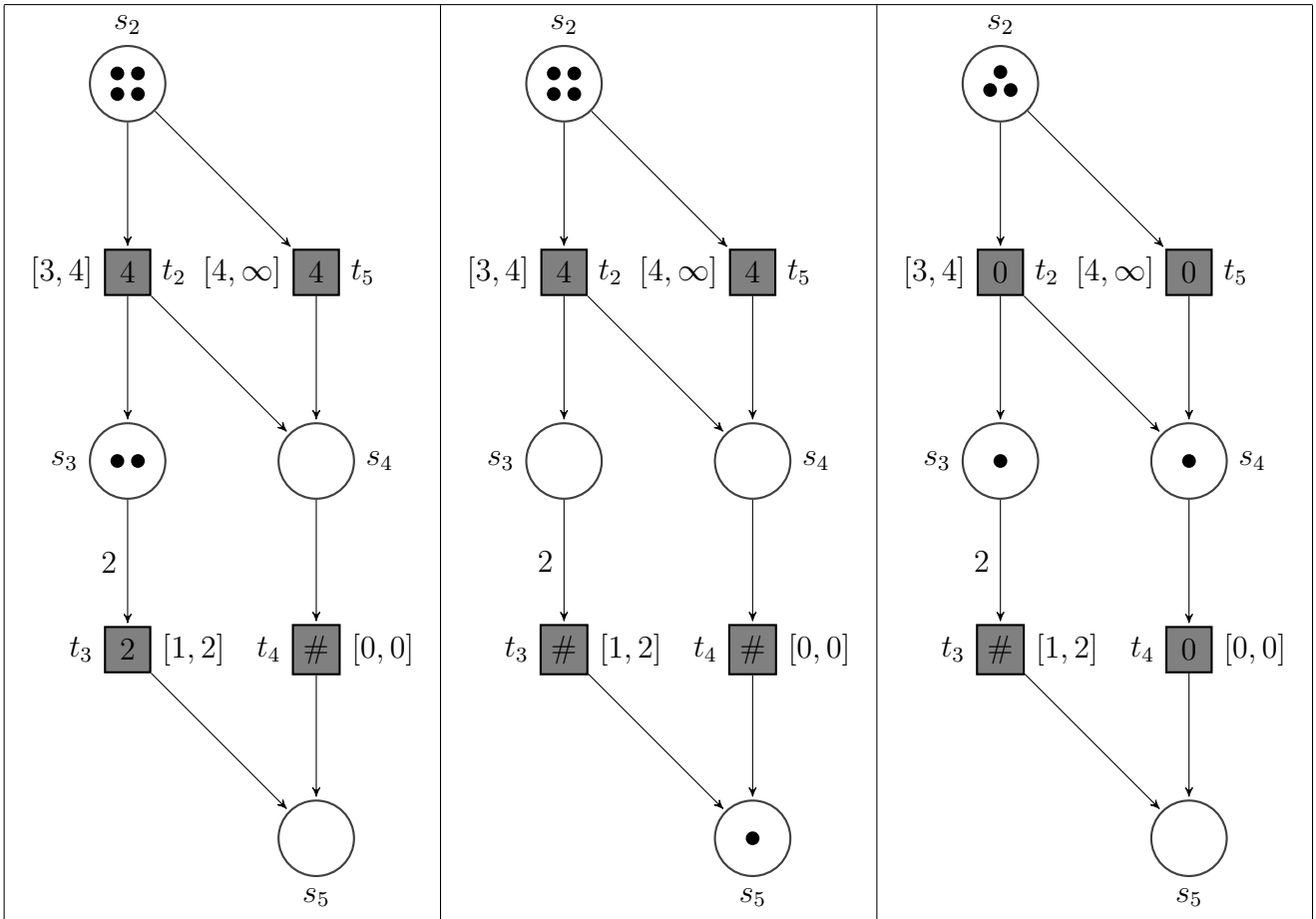


Abbildung 13: Schalten in einem zeitbehafteten Petri-Netz

Durch das Schalten nimmt die t-Markierung für keine Transition einen höheren Wert an, sodass durch das Schalten keine Zeit vergeht. Um das Vergehen von Zeit zu modellieren existiert in zeitbehafteten Petri-Netzen eine weitere Möglichkeit zur Überführung von Zuständen, die im Folgenden nach [16] eingeführt wird.

Definition 29. Seien $Z = (P, T, F, W, m_0, I)$ ein zeitbehaftetes Petri-Netz und $z = (m, h)$ ein Zustand in Z. Sei $\tau \in \mathbb{R}_{\geq 0}$. Dann kann *Zeit entsprechend τ in z vergehen* (Notation: $z \xrightarrow{\tau}$), wenn

- $\forall t \in T\colon h(t) \neq \# \Rightarrow h(t) + \tau \leq \mathrm{lft}(t)$

Nachdem τ Zeit vergangen ist, ändert sich der Zustand zu $z' = (m', h')$ (Notation $z \xrightarrow{\tau} z'$), mit

- $m' = m$
- $\forall t \in T : h'(t) = \begin{cases} \# & , \text{ falls } t^- \not\leq m' \\ h(t) + \tau & , \text{ falls } t^- \leq m' \end{cases}$

Das *Vergehen von Zeit* verändert dabei nur die t-Markierung eines Zustandes. Da in einem Zustand eines zeitbehafteten Petri-Netz für jede Transition genau die Zeit relevant ist, die seit der Aktivierung der Transition vergangen ist, bleibt der Wert für Transitionen mit dem Wert # gleich, während alle anderen Transitionen einen aktualisierten Wert erhalten. Laut Definition 24 muss in einem Zustand eines zeitbehafteten Petri-Netz der Wert der h-Markierung kleiner gleich dem lft-Wert sein. Damit diese Restriktion durch das Vergehen von Zeit nicht verletzt wird, kann nur soviel Zeit vergehen, dass der lft-Wert für keine Transition überschritten wird.

In Abbildung 14 sind zum Beispiel die zeitbehafteten Petri-Netze TPN_4 und TPN_5 zu sehen, wobei TPN_5 durch das Vergehen von einer Zeiteinheit aus TPN_4 entstanden ist. Der Wert von t_4 ist dabei gleich geblieben, weil die Transition nicht aktiviert ist und somit den Wert # hat. Alle anderen Transitionen sind aktiviert und haben deshalb einen aktualisierten Wert bekommen, der dem alten Wert addiert mit der vergangenen Zeit entspricht. Zu beachten ist, dass es nicht möglich ist, dass in TPN_4 zwei Zeiteinheiten vergehen, weil dann t_2, t_3 und t_5 jeweils ihre rechte Intervallgrenze überschreiten würden.

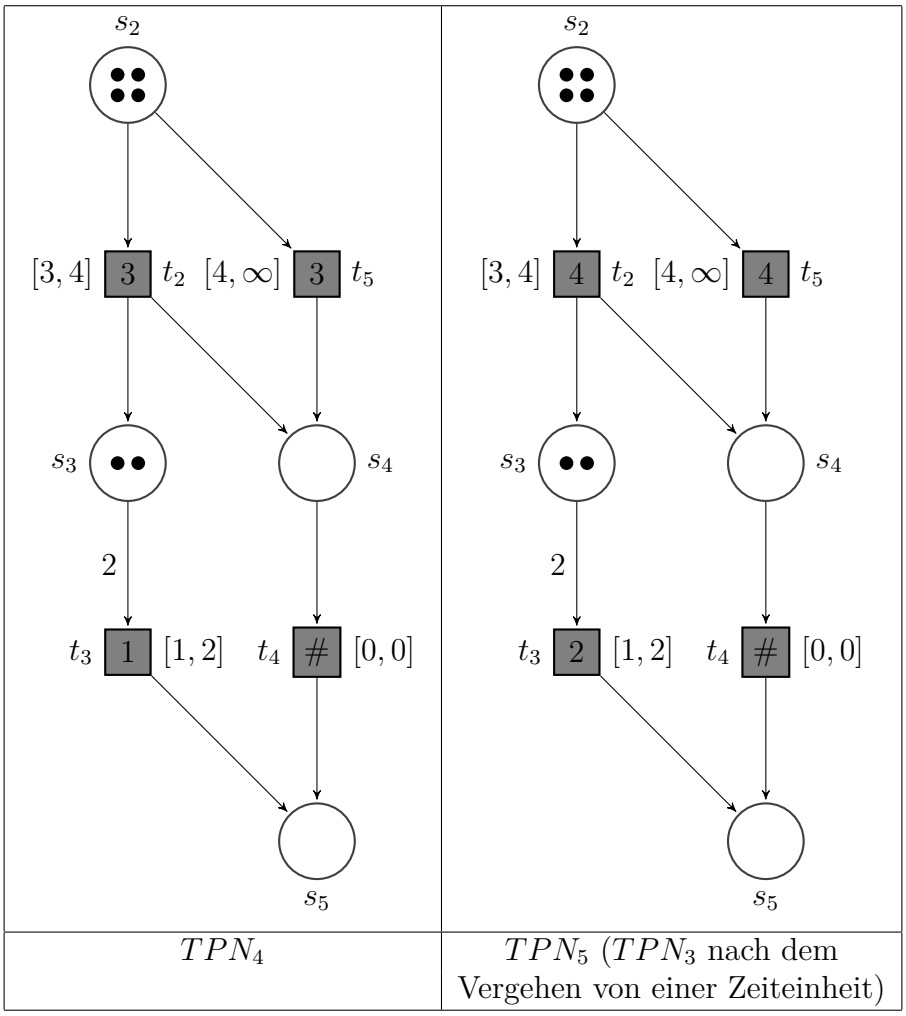


Abbildung 14: Das Vergehen von Zeit in einem zeitbehafteten Petri-Netz

Im Kontext von klassischen Petri-Netzen wurde das Erreichbarkeits-Problem eingeführt und erwähnt, dass dieses interessant für die Analyse der Petri-Netze ist. Gleiches gilt auch für die zeitbehaftete Variante, weshalb im Folgenden Erreichbarkeit in dieser thematisiert wird. Im Gegensatz zu den klassischen Petri-Netzen existiert neben dem Schalten mit dem Vergehen von Zeit eine weitere Möglichkeit, Zustände in Folgezustände zu überführen, die miteinbezogen werden muss. Aus diesem Grund werden im Folgenden *Läufe* von zeitbehafeten Petri-Netzen eingeführt, die Folgen von Transitionen und Zeiten kombinieren.

Definition 30. Seien $Z = (P, T, F, W, m_0, I)$ ein zeitbehaftetes Petri-Netz, $\sigma = t_1 \cdots t_n$ eine Folge von Transitionen und $\tau = \tau_0\tau_1 \cdots \tau_n$ mit $\tau_j \in \mathbb{R}_{\geq 0}$ eine Folge von Zeitangaben. Dann ist $\sigma(\tau) = \tau_0 t_1 \tau_1 \cdots t_n \tau_n$ ein Lauf von σ.

Ein Lauf drückt aus, dass zwischen dem Schalten von Transitionen Zeit vergehen kann, bezieht allerdings nicht ein, ob ein abwechselnden Vergehen von Zeit und Schalten entsprechend des Laufes tatsächlich *durchführbar* ist, ohne dass die Restriktionen aus den Definitionen 28 und 29 verletzt werden. So wäre $3t_43t_20$ zwar ein Lauf für TPN_3 aus Abbildung 13, jedoch würde bereits das Vergehen von drei Zeiteinheiten dazu führen, dass t_4 nicht innerhalb ihres Zeitintervalls schaltet. Um solche Läufe auszuschließen, führt Definition 31 *durchführbare Läufe* ein.

Definition 31. Seien $Z = (P, T, F, W, m_0, I)$ ein zeitbehaftetes Petri-Netz, $z = (m, h)$ ein Zustand in Z und $\sigma(\tau) = \tau_0 t_1 \tau_1 \cdots t_n \tau_n$ ein Lauf von $\sigma = t_1, \ldots, t_n$, dann schaltet $\sigma(\tau)$ von z nach z' (Notation: $z \overset{\sigma(\tau)}{\to} z'$), falls gilt:

- $z = z'$, falls $n = 0$
- $z \overset{\tau_0 t_1 \tau_1 \cdots t_{n-1}\tau_{n-1}}{\to} z_1$, $z_1 \overset{t_n}{\to} z_2$ und $z_2 \overset{\tau_n}{\to} z'$ für zwei Zustände z_1 und z_2 , falls $n > 0$

Ein Lauf $\sigma(\tau)$ heißt *durchführbar in Z ausgehend von* z, falls ein Zustand z' existiert, sodass $z \overset{\sigma(\tau)}{\to} z'$.

Ein Lauf $\sigma(\tau)$ heißt *durchführbar* in Z, falls $\sigma(\tau)$ ein durchführbarer Lauf ausgehend von z_0 ist.

Betrachtet man noch einmal TPN_3 aus Abbildung 13, dann ist $3t_43t_20$ kein durchführbarer Lauf in TPN_3, weil $z_0 \overset{3}{\to}$ nicht gilt. $0t_43t_21t_30$ ist dagegen ein durchführbarer Lauf.

Zu beachten ist, dass in einem durchführbaren Lauf auch keine Zeiteinheiten zwischen dem Schalten von zwei Transitionen vergehen können. Da durch das Schalten an sich keine Zeit vergeht, schalten dann beide Transitionen zum gleichen Zeitpunkt, wodurch sich Parallelität ausdrücken lässt. Diese Eigenschaft wird sich im folgenden Kapitel als nützlich erweisen, um Parallelität in Lösungen zu Scheduling-Problemen zu modellieren.

Auf Basis der durchführbaren Läufe können jetzt auch *Schaltfolgen* von zeitbehafteten Petri-Netzen eingeführt werden.

Definition 32. Seien $Z = (P, T, F, W, m_0, I)$ ein zeitbehaftetes Petri-Netz und z, z' Zustände von Z, dann ist $\sigma = t_1 \cdots t_n$ mit $t_i \in T$ eine *Schaltfolge* von z nach z' (Notation: $z \overset{\sigma}{\to} z'$), wenn eine Folge von Zeiten $\tau = \tau_0 \cdots \tau_n$ existiert, sodass $z \overset{\sigma(\tau)}{\to} z'$.

Schaltfolgen können verwendet werden, um Erreichbarkeit analog zu klassischen Petri-Netzen einzuführen.

Definition 33. Seien $Z = (P, T, F, W, m_0, I)$ ein zeitbehaftetes Petri-Netz und z ein Zustand in Z. Dann ist z erreichbar in Z, falls eine Schaltfolge σ existiert, sodass $z_0 \xrightarrow{\sigma} z$.

Im Gegensatz zur Erreichbarkeit in klassischen Petri-Netzen, bezieht sich die Variante für zeitbehaftete Petri-Netze auf Zustände anstelle von Markierungen. Wie wir im folgenden Kapitel noch bemerken werden, kann die t-Markierung irrelevant für Analyseanwendungen des Erreichbarkeits-Problems sein, deshalb wird abschließend die Erreichbarkeit von p-Markierungen definiert.

Definition 34. Seien $Z = (P, T, F, W, m_0, I)$ ein zeitbehaftetes Petri-Netz und m eine p-Markierung in Z. m heißt erreichbar in Z, wenn eine t-Markierung h existiert, sodass der Zustand $z = (m, h)$ erreichbar in Z ist.

4 Modellierung von Resource-Constrained Project Scheduling Problems mit zeitbehafteten Petri-Netzen

In den Kapiteln 2 und 3 wurden das RCPSP mit diversen Erweiterungen sowie zeitbehaftete Petri-Netze eingeführt. In diesem Kapitel wird nun erläutert, wie ein beliebiges RCPSP mit Hilfe von zeitbehafteten Petri-Netzen modelliert werden kann. Ziel ist es also, eine Konstruktion anzugeben, die ein formal definiertes RCPSP in ein zeitbehaftetes Petri-Netz transformiert. Für das Petri-Netz soll dabei gelten, dass es bestimmte Eigenschaften erfüllt genau dann, wenn das ursprüngliche RCPSP eine zulässige Lösung hat. Dies wird gefordert, damit durch Überprüfung eben jener Eigenschaften nachvollzogen werden kann, ob das ursprüngliche RCPSP eine Lösung hat. Außerdem sollen mit Hilfe des konstruierten zeitbehafteten Petri-Netzes auch konkrete zulässige Lösungen für das ursprüngliche RCPSP ermittelbar sein. Diese beiden Aspekte können dann zur Analyse der RCPSPs herangezogen werden.

4.1 Ziele der Konstruktion

Eine Eigenschaft, die im Kontext von Petri-Netzen sowie zeitbehafteten Petri-Netzen eingeführt wurde, ist die Erreichbarkeit. Die Erreichbarkeit beantwortet die Frage, ob in einem gegebenen zeitbehafteten Petri-Netz eine gegebene Markierung ausgehend vom initialen Zustand erreichbar ist. Mit Hilfe dieser Eigenschaft lassen sich die Ziele der Konstruktion konkretisieren:

1. Aus jedem RCPSP soll ein zeitbehaftetes Petri-Netz Z sowie eine p-Markierung m_{goal} konstruiert werden, sodass m_{goal} in Z genau dann erreichbar ist, wenn das ursprüngliche RCPSP eine zulässige Lösung hat.
2. Mit Hilfe von Z und m_{goal} sollen zulässige Lösungen des ursprünglich gegebenen RCPSP berechenbar sein. Dies soll mit Hilfe von durchführbaren Läufen, die zu einem Zustand $z = (m_{goal}, h)$ führen, wobei h eine t-Markierung ist, möglich sein.

Zu beachten ist, dass ein solcher durchführbarer Lauf existiert, wenn m_{goal} erreichbar ist, sodass mit Hilfe einer Konstruktion, die beide Ziele erfüllt, für jedes RCPSP eine zulässige Lösung berechnet werden kann, sofern eine existiert.

Bevor die Konstruktion angegeben und erläutert wird, führt der folgende Abschnitt die grundlegenden Ideen der Konstruktion ein, mit denen die eben formulierten Ziele erreicht werden sollen. Eine detailliertere Beschreibung und Erklärung der einzelnen Bestandteile der Konstruktion folgt dann in Unterabschnitt 4.3.

4.2 Ideen der Konstruktion

Für die Beschreibung der Konstruktionsidee nehmen wir an,

$$X = (D, [n], [e], [v], R^e, R^v, M, p, r^e, r^v, V, d, \hat{d})$$

sei ein RCPSP und (Z, m_{goal}) das Resultat der Konstruktion, wobei Z ein zeitbehaftetes Petri-Netz und m_{goal} eine p-Markierung von Z ist. Im Verlauf der Konstruktion wird für jede Aktion $i \in [n]$ eine Transition $start_i$, die den Start der Aktion modellieren soll, eingefügt. Des Weiteren werden für jede Aktion i und jeden Modus $m \in [M_i]$ die Transitionen $modus_{im}$ und $finish_{im}$ hinzugefügt, welche die Auswahl des Modus bzw. das Beenden der Aktion im entsprechenden Modus modellieren. Außerdem wird eine Stelle $finished_i$ eingefügt, die von den $finish_{im}$-Transitionen befüllt wird. Der konkrete Zusammenhang zwischen den eingeführten Elementen wird im späteren Verlauf erläutert. Abbildung 15 skizziert diesen allerdings schon einmal.

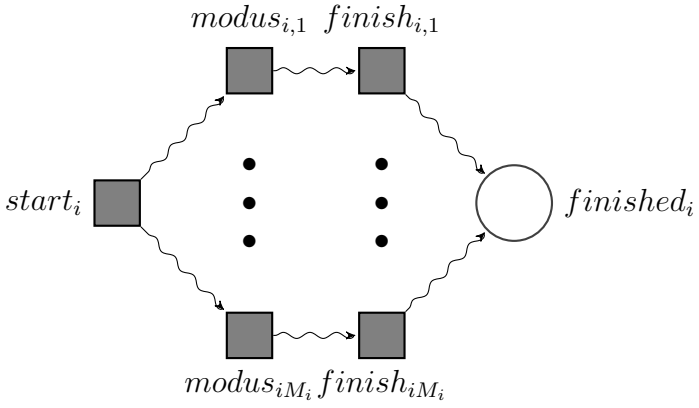


Abbildung 15: Skizze der Konstruktion einer Aktion. Die geschlängelten Linien repräsentieren dabei weitere Petri-Netz Bestandteile, wobei die Pfeilspitzen anzeigen, in welche Richtung Token „weitergegeben“ werden können.

Für jede Aktion wird also ein Gebilde eingefügt, wie es in Abbildung 15 dargestellt wird. Dabei werden die einzelnen Gebilde für zwei Aktionen i, j verbunden, wenn $(i, j) \in V$. Diese

Verbindung wird dann von $finished_i$ nach $start_j$ führen (siehe auch Abbildung 16). Betrachtet man die einzelnen Gebilde, die Aktionen repräsentieren, als Knoten und die Verbindungen zwischen den Gebilden als Kanten, ergibt sich daraus der Vorgängerrelationen-Graph des RCPSP.

Die Idee ist nun, dass in m_{goal} die $finished_i$-Stellen genau dann ein Token enthalten, wenn i keine Nachfolge-Aktionen in der Vorgängerrelation V besitzt, d.h. $m_{goal}(finished_i) = 1$ genau dann, wenn $(i, j) \notin V$ für alle $j \in [n]$. Alle anderen Stellen sollen keine Token enthalten. Die Intuition dahinter ist, dass das Ausführen einer Aktion i durch das Schalten von Transitionen modelliert wird, wobei nach dem Ausführen der Aktion Token in die Stelle $finished_i$ gelegt werden, welche dann wiederum von möglichen Nachfolgeaktionen entnommen werden können, damit diese dann ihrerseits ausgeführt werden. Wenn alle Aktionen abgeschlossen wurden, liegen somit nur Token in den $finished_i$-Stellen der Aktionen, die keine Nachfolger haben.

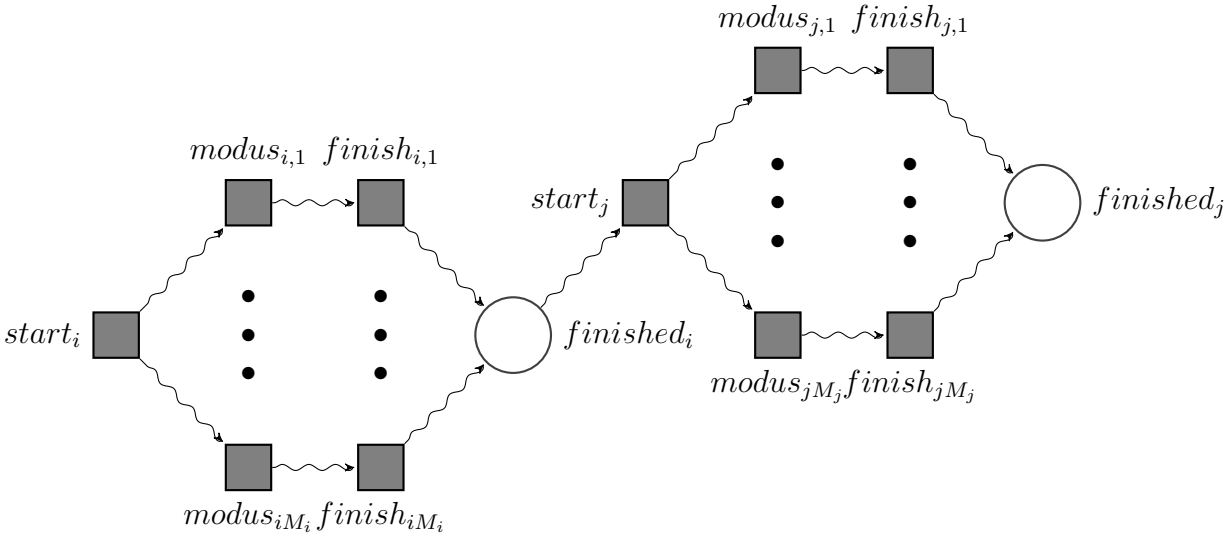


Abbildung 16: Skizze der Verbindung zwischen zwei Aktionen i, j mit $(i, j) \in V$.

Das zweite Ziel der Konstruktion besagt, dass aus durchführbaren Läufen, die zu m_{goal} führen, zulässige Lösungen für das ursprüngliche RCPSP berechenbar sein sollen. Zur Erläuterung der Idee zur Umsetzung dieses Zieles nehmen wir an, $\sigma = t_1 \cdots t_k$ sei eine Folge von Transitionen und $\tau = \tau_0 \cdots \tau_k$ eine Folge von Zeiten, sodass $\sigma(\tau) = \tau_0 t_1 \tau_1 \cdots t_k \tau_k$ ein durchführbarer Lauf ist, der zu m_{goal} führt, also $z_0 \overset{\sigma(\tau)}{\rightarrow} (m_{goal}, h)$ für eine t-Markierung h. Wie die Skizze der Konstruktion bereits andeutet, soll in einem solchen Lauf zu m_{goal} für jede Aktion i die Transition $start_i$ genau einmal vorkommen. Ebenso soll für jede Aktion i genau eine Transition $modus_{im}$ mit $m \in [M_i]$ vorkommen. Diese Eigenschaft soll das Berechnen einer zulässigen Lösung ermöglichen. Dazu verwende für eine Aktion i die eindeutig bestimmten Indizes $l \in \{1, \ldots, k\}$ und $m \in \{1, \ldots, M_i\}$, sodass einerseits $t_l = start_i$ gilt und andererseits $modus_{im}$ in der Folge σ vorkommt. Berechne dann die Summe $S_i := \Sigma_{w=0}^{l-1} \tau_w$. Die Werte S_i und $m_i := m$ sollen nun einen zulässigen Schedule von X ergeben. Abbildung 17 veranschaulicht das Vorgehen.

					$S_i := \sum_{w=0}^{l-1} \tau_w$				$m_i := m$			
Durchführbarer Lauf	τ_0	t_1	τ_1	...	τ_{l-1}	$start_i$	τ_l	...	$modus_{im}$	...	t_k	τ_k
Index	0	1	1	...	$l-1$	l	l	...	b	...	k	k

Abbildung 17: Berechnung eines Schedules anhand eines durchführbaren Laufes des konstruierten zeitbehafteten Petri-Netz für eine Aktion i und einen Modus $m \in [M_i]$.

Der folgende Abschnitt widmet sich der konkreten Konstruktion mit dem Ziel, die eben beschriebenen Ideen umzusetzen.

4.3 Konstruktion

Zur Erläuterung der Konstruktion wird diese der Übersichtlichkeit halber in einzelne Teile aufgegliedert, die am Ende dieses Abschnittes zur vollständigen Konstruktion zusammengefügt werden, wobei sich die Aufteilung an den einzelnen Komponenten eines RCPSP orientiert.

Gegeben sei also ein beliebiges RCPSP $X = (D, [n], [e], [v], R^e, R^v, M, p, r^e, r^v, V, d, \hat{d})$. Das Ergebnis der Konstruktion ist ein zeitbehaftetes Petri-Netz $Z = (P, T, F, W, m_0, I)$ sowie eine p-Markierung m_{goal} von Z.

Zu Beginn der Konstruktion werden die Komponenten des zu konstruierenden zeitbehafteten Petri-Netzes Z wie im folgenden Pseudocode mit leeren Mengen bzw. Abbildungen initialisiert [10].

Algorithm 1: Initialization()

Data: Das RCPSP X
Result: Das initialisierte zeitbehaftete Petri-Netz Z
begin
  $P := \emptyset$
  $T := \emptyset$
  $F := \emptyset$
  $W := \emptyset$ [a]
  $m_0 := \emptyset$
  $I := \emptyset$

[a] Im Kontext von Abbildungen sei $\emptyset$ dabei die Abbildung von der leeren Menge in eine Zielmenge.

[10] Genau genommen liegt an dieser Stelle noch kein zeitbehaftetes Petri-Netz vor, da $P \cup T = \emptyset$ (siehe dazu auch Definition 11)

4.3.1 Modellierung von Ressourcen

In diesem Abschnitt werden die Ressourcen und ihre Vorkommen modelliert. Die Idee ist, für jede Ressource eine Stelle in das bereits konstruierte zeitbehaftete Petri-Netz einzufügen, deren initiale Tokenzahl den verfügbaren Einheiten der Ressource entspricht (siehe auch Abbildung 18).

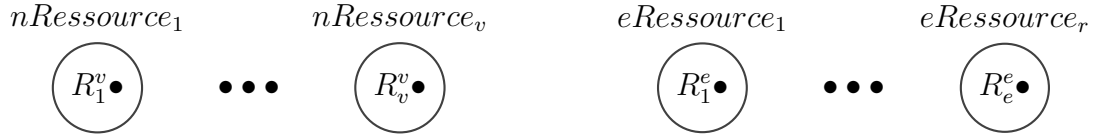


Abbildung 18: Konstruktion der Ressourcen sowie ihrer Vorkommen. Die Beschriftung $R_1^v\bullet$ meint dabei, dass initial R_1^v Token in der entsprechenden Stelle liegen. Die Anfangsbuchstaben n und e der Stellennamen stehen für „nicht erneuerbar“ bzw. „erneuerbar“.

Die Pseudocode Ausschnitte 2 und 3 veranschaulichen die Konstruktion für eine nicht erneuerbare Ressource $q \in [v]$ bzw. eine erneuerbare Ressource $k \in [e]$. Wie in Abschnitt 2 definiert, bezeichnet R_q^v die Anzahl der verfügbaren Einheiten von Ressource q und R_k^e die Anzahl der verfügbaren Einheiten von Ressource k. Zu beachten ist, dass diese Ausschnitte in der vollständigen Konstruktion für jede Ressource $q \in [v]$ bzw. $k \in [e]$ ausgeführt werden müssen.

Algorithm 2: ModelNonRenewableRessource(NonRenewableRessource q)

Input: Eine nicht erneuerbare Ressource $q \in [v]$
Data: Das RCPSP X, das bisher konstruierte zeitbehaftete Petri-Netz Z
Result: Das zeitbehaftete Petri-Netz Z erweitert um die Modellierung der nicht erneuerbaren Ressource q

begin

$$P := P \cup \{nRessource_q\}$$
$$m_0(p) := \begin{cases} R_q^v & \text{, falls } p = nRessource_q \\ m_0(p) & \text{, sonst} \end{cases}$$

Algorithm 3: ModelRenewableRessource(RenewableRessource k)

Input: Eine erneuerbare Ressource $k \in [e]$
Data: Das RCPSP X, das bisher konstruierte zeitbehaftete Petri-Netz Z
Result: Das zeitbehaftete Petri-Netz Z erweitert um die Modellierung der erneuerbaren Ressource k

begin

$$P := P \cup \{eRessource_k\}$$
$$m_0(p) := \begin{cases} R_k^e & \text{, falls } p = eRessource_k \\ m_0(p) & \text{, sonst} \end{cases}$$

Im späteren Verlauf der Konstruktion werden Start- und Endzeitpunkt einer Aktion mit jeweils einer Transition modelliert. Diese Transitionen werden Ressourcen entsprechend ihres Bedarfs entnehmen und – im Falle von erneuerbaren Ressourcen – wieder zurücklegen. Der Gebrauch der Ressourcen durch Aktionen wird in Abschnitt 4.3.3 näher erläutert. Die Beschränkung der Tokenzahl auf das maximale verfügbare Vorkommen der jeweiligen Ressource sorgt dann dafür, dass die ressourcenbedingten Restriktionen eingehalten werden.

4.3.2 Modellierung von Aktionen

Nach den Ressourcen soll nun die Modellierung der Aktionen erläutert werden. Wie bereits erwähnt, soll das Starten einer Aktion i durch eine Transition $start_i$, die Auswahl eines Modus $m \in [M_i]$ durch eine Transition $modus_{im}$ und der Abschluss der Aktion i im Modus m durch eine Transition $finish_{im}$ modelliert werden. Dabei soll aus durchführbaren Läufen des konstruierten zeitbehafteten Petri-Netzes, die zu m_{goal} führen, ein Schedule berechenbar sein. Damit die eingeführte Methode eindeutige und vollständige Ergebnisse erzielt, darf für eine Aktion i die Transition $start_i$ nur genau einmal in so einem Lauf vorkommen. Analog darf für jede Aktion i auch nur eine Transition $modus_{im}$ für einen Modus $m \in [M_i]$ vorkommen.

Zusätzlich ergibt sich eine weitere Anforderung für die berechneten Startzeitpunkte. Damit der berechnete Schedule (S, m) zulässig ist, muss $S_i \leq D$ für jeden Eintrag S_i mit $i \in [n]$ gelten. Vor dem Schalten von $start_i$ dürfen also höchstens D Zeiteinheiten vergehen, damit die berechneten Startzeitpunkte im Zeithorizont liegen (siehe auch Abbildung 19). Damit die ermittelten Schedules auch zulässig sind, ist es zusätzlich erforderlich, dass zwischen dem Schalten der $start_i$- und $finish_{im}$- Transition genau p_{im} Zeiteinheiten vergehen. Der genaue Grund dafür wird in Abschnitt 4.3.4 deutlich werden. Abbildung 19 konkretisiert diese Anforderungen.

Bei der Konstruktion der Aktionen müssen also die eben formulierten Anforderungen berücksichtigt werden.

	$\overbrace{\quad\quad\quad\quad\quad\quad\quad\quad}^{\sum_{w=0}^{l-1} \tau_w = S_i \leq D}$					$\overbrace{\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}^{\sum_{w=l}^{a-1} \tau_w = p_{im}}$					
Durchführbarer Lauf	τ_0	t_1	τ_1	...	$start_i$	τ_l	...	$modus_{im}$	...	$finish_{im}$	...
Index	0	1	1	...	l	l	...	b	...	a	...

Abbildung 19: Anforderungen an die Konstruktion von Aktionen am Beispiel einer Aktion i und eines Modus $m \in [M_i]$.

Betrachten wir dazu eine beliebige Aktion i mit den Modi $[M_i]$ und den Bearbeitungszeiten p_{im} in den Modi $m \in [M_i]$, dann werden der Konstruktion die Transitionen $start_i$, $modus_{im}$ und $finish_{im}$ für alle $m \in [M_i]$ hinzugefügt.

Um die ersten beiden Anforderung zu erfüllen, wird eine Stelle $started_i$ hinzugefügt, die Nachstelle von $start_i$ sowie Vorstelle der Transitionen $modus_{im}$ ist, wobei die Kantengewichtung jeweils auf 1 und das Zeitintervall von $start_i$ auf $[0, \infty]$ gesetzt wird. Außerdem wird eine Stelle $notStarted_i$ hinzugefügt, die initial ein Token enthält und Vorstelle von $start_i$ ist. Weil diese Stelle keine eingehenden Kanten hat und somit auch keine Token hinzugefügt bekommen kann, ist dafür gesorgt, dass $start_i$ in jedem durchführbaren Lauf nur einmal schalten kann. Weil somit auch nur höchstens einmal genau ein Token in die Stelle $started_i$ gelegt wird, kann nur eine Transition $modus_{im}$ im durchführbaren Lauf vorkommen (siehe auch Abbildung 20).

Um die Erfüllung der dritten Anforderung zu garantieren, also dass in einem durchführbaren Lauf, der nach m_{goal} führt, vor dem Schalten von $start_i$ höchstens D Zeiteinheiten vergehen, wird eine weitere Transition $timeHorizon_i$ hinzugefügt, die $notStarted_i$ als Vorstelle hat und dieser genau ein Token entzieht. Für das Zeitintervall dieser Transition soll gelten $I(timeHorizon_i) = (D, D)$ (siehe auch Abbildung 20), sie soll also genau D Zeiteinheiten nach ihrer Aktivierung schalten müssen. Weil diese Transition nur eine Vorstelle hat, die initial genügend Token enthält, ist die Transition m_0-aktiviert. Wenn also in einem durchführbaren Lauf, der von z_0 ausgeht, $start_i$ nach dem Vergehen von mehr als D Zeiteinheiten noch nicht geschaltet hat, dann kann sie auch im weiteren Verlauf nicht mehr schalten, weil dann $timeHorizon_i$ das Token aus $notStarted_i$ nach genau D Zeiteinheiten entzieht, sodass $start_i$ im Zustand, der dann vorliegt, nicht mehr aktiviert ist. Wie wir im weiteren Verlauf der Konstruktion sehen werden, kann m_{goal} dann nicht mehr erreicht werden, sodass der Startzeitpunkt, der aus einem durchführbaren Lauf, der nach m_{goal} führt, berechnet wird, kleiner gleich dem Zeithorizont D sein muss.

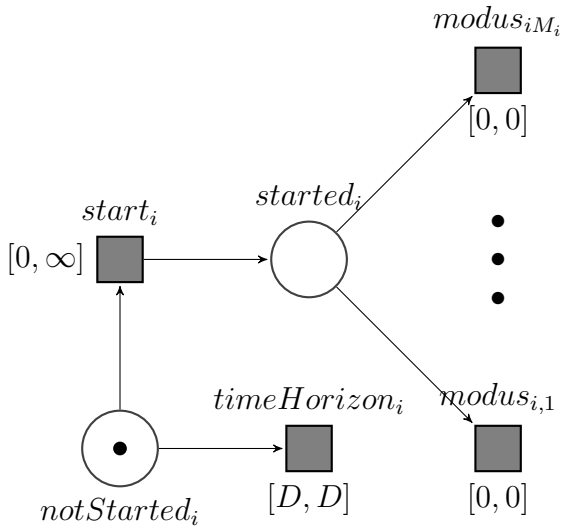


Abbildung 20: Konstruktion des Starts einer Aktion.

Schließlich muss noch dafür gesorgt werden, dass die vierte Anforderung erfüllt wird, dass also zwischen dem Schalten von $start_{im}$ und $finish_{im}$ genau p_{im} Zeiteinheiten vergehen. Dazu wird für jede Aktion $i \in [n]$ und jeden Modus $m \in [M_i]$ jeweils eine Stelle $inProgress_{im}$

eingefügt, die Vorstelle von $finish_{im}$ und Nachstelle von $modus_{im}$ ist. Dabei wird das Zeitintervall der Transition $finish_{im}$ auf $[p_{im}, p_{im}]$ und das von $modus_{im}$ auf $[0, 0]$ gesetzt. Dadurch wird erreicht, dass zwischen dem Schalten von $start_i$ und $modus_{im}$ keine Zeit vergehen kann und zwischen dem Schalten von $modus_{im}$ und $finish_{im}$ genau p_{im} Zeiteinheiten vergehen müssen, wodurch genannte Anforderung erfüllt ist. Um in einer p-Markierung repräsentieren zu können, dass eine Aktion i abgeschlossen wurde, wird außerdem eine Stelle $finished_i$ eingeführt, die jeweils Nachstelle von $finish_{im}$ ist. Abbildung 21 zeigt die vollständige Konstruktion einer Aktion, wobei $Post_i = \{b \in [n] | (i, b) \in V\}$ die Menge der Nachfolgeaktionen der Aktion i ist. Die Transitionen $finish_{im}$ legen also für jede Nachfolgeaktion der Aktion i ein Token in die Stelle $finished_i$. Dies stellt einen Vorgriff auf die Modellierung der Vorgängerrelationen dar und wird in Abschnitt 4.3.4 näher erläutert.

Anhand der Konstruktion der Aktionen sowie der Ermittlung eines Schedules aus einem durchführbaren Lauf in Z, der zu m_{goal} führt, lässt sich eine weitere Eigenschaft des konstruierten zeitbehafteten Petri-Netz erkennen.

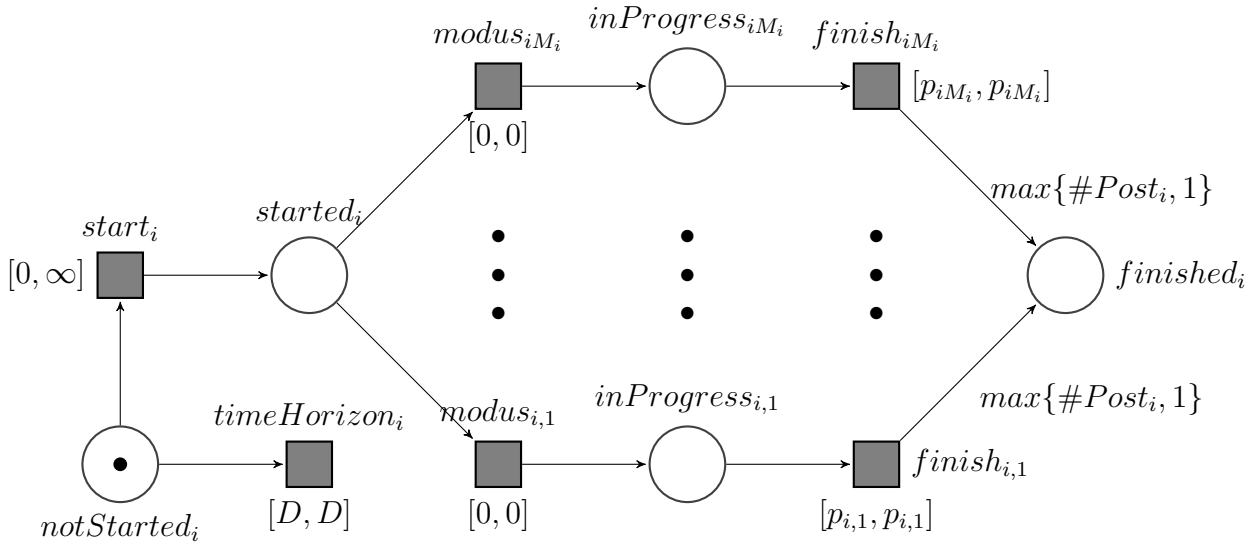


Abbildung 21: Konstruktion einer Aktion

Aus der Ermittlung eines Schedules (siehe auch Abbildung 17) folgt, dass in dem durchführbaren Lauf aus dem der Schedule berechnet wird, der berechnete Startzeitpunkt S_i einer Aktion i der Anzahl an Zeiteinheiten entspricht, die vor dem Schalten von Transition $start_i$ vergangen sind. Außerdem ist bekannt, dass anschließend genau p_{im} Zeiteinheiten vergehen, bevor $finish_{im}$ für einen Modus $m \in [M_i]$ schalten kann. Daraus folgt, dass in dem durchführbaren Lauf, aus dem S_i berechnet wurde, genau $S_i + p_{im}$ Zeiteinheiten vergehen, bevor $finish_{im}$ schaltet, was dem Endzeitpunkt C_i der Aktion i entspricht (siehe auch Abbildung 22). Somit lassen sich also auch die Endzeitpunkte von Aktionen aus einem durchführbaren Lauf im konstruierten zeitbehafteten Petri-Netz, der zu m_{goal} führt, berechnen. Hier wird auch deutlich, weshalb für eine Aktion i für jeden Modus $m \in [M_i]$ eine Transition $finish_{im}$ eingefügt wird, denn der Endzeitpunkt C_i hängt vom ausgewählten Modus ab.

$S_i + p_{im} = C_i$

$S_i = \sum_{w=0}^{l-1} \tau_w$ $\qquad$ $\sum_{w=l}^{a-1} \tau_w = p_{im}$

Durchführbarer Lauf	τ_0	t_1	τ_1	...	$start_i$	τ_l	...	$modus_{im}$	...	$finish_{im}$	...
Index	0	1	1	...	l	l	...	b	...	a	...

Abbildung 22: Berechnen des Endzeitpunkt C_i einer Aktion $i \in [n]$ im Modus $m \in [M_i]$ anhand eines durchführbaren Laufs des konstruierten zeitbehafteten Petri-Netz.

Weil die Restriktionen, die aus der Vorgängerrelation des ursprünglichen RCPSP resultieren, abhängig vom Endzeitpunkt der Aktionen sind, ist diese Eigenschaft sehr nützlich im weiteren Verlauf der Konstruktion.

Der Pseudocode 4 stellt die Konstruktion algorithmisch für eine gegebene Aktion i dar. In der vollständigen Konstruktion muss das Vorgehen für jede Aktion ausgeführt werden.

Algorithm 4: ModelAction(Action i)

Input: Eine Aktion $i \in [n]$
Data: Das RCPSP X, das bisher konstruierte zeitbehaftete Petri-Netz Z
Result: Das zeitbehaftete Petri-Netz Z erweitert um die Modellierung der Aktion i
begin

$P := P \cup \{started_i, finished_i, notStarted_i\}$
$T := T \cup \{start_i, timeHorizon_i\}$
$F := F \cup \{(start_i, started_i), (notStarted_i, start_i), (notStarted_i, timeHorizon_i)\}$

$$m_0(p) := \begin{cases} 1 & \text{, falls } p = notStarted_i \\ 0 & \text{, falls } p = started_i \\ 0 & \text{, falls } p = finished_i \\ m_0(p) & \text{, sonst} \end{cases}$$

$$I(t) := \begin{cases} (0, \infty) & \text{, falls } t = start_i \\ (D, D) & \text{, falls } t = timeHorizon_i \\ I(t) & \text{, sonst} \end{cases}$$

$$W(f) := \begin{cases} 1 & \text{, falls } f = (start_i, started_i) \\ 1 & \text{, falls } f = (notStarted_i, start_i) \\ 1 & \text{, falls } f = (notStarted_i, timeHorizon_i) \\ W(f) & \text{, sonst} \end{cases}$$

for $m \in [M_i]$ **do**

$T := T \cup \{modus_{im}, finish_{im}\}$
$P := P \cup \{inProgress_{im}\}$
$F := F \cup \{(started_i, modus_{im}), (modus_{im}, inProgress_{im})\} \cup$
$\{(inProgress_{im}, finish_{im}), (finish_{im}, finished_i)\}$

$$W(f) := \begin{cases} 1 & \text{, falls } f = (started_i, modus_{im}) \\ 1 & \text{, falls } f = (modus_{im}, inProgress_{im}) \\ 1 & \text{, falls } f = (inProgress_{im}, finish_{im}) \\ max\{\#Post_i, 1\} & \text{, falls } f = (finish_{im}, finished_i) \\ W(f) & \text{, sonst} \end{cases}$$

$$I(t) := \begin{cases} (0,0) & \text{, falls } t = modus_{im} \\ (p_{im}, p_{im}) & \text{, falls } t = finish_{im} \\ I(t) & \text{, sonst} \end{cases}$$

$$m_0(p) := \begin{cases} 0 & \text{, falls } p = inProgress_{im} \\ m_0(p) & \text{, sonst} \end{cases}$$

4.3.3 Modellierung der Ressourcenbedarfe

Im vorigen Abschnitt wurden Aktionen modelliert, wobei das Starten der Aktion sowie die Auswahl des Modus berücksichtigt wurden. Allerdings wurde nicht einbezogen, dass Modi erneuerbare und nicht erneuerbare Ressourcen belegen bzw. verbrauchen. Wichtig ist, dass ein Schedule, der aus einem durchführbaren Lauf in Z, der nach m_{goal} führt, berechnet wurde, die ressourcenbedingten Restriktionen eines zulässigen Schedules für ein RCPSP nicht verletzt.

Um dies zu erreichen betrachten wir zunächst die zwei Arten von ressourcenbedingten Einschränkungen an einen zulässigen Schedule des gegebenen RCPSP X.

Die erste Einschränkung betrifft die nicht erneuerbaren Ressourcen $[v]$ und besagt, dass das maximale Vorkommen R_q^v einer Ressource $q \in [v]$ nicht von den Aktionen im gesamten Verlauf des Schedules überschritten werden darf. Dabei hängt der Verbrauch einer Aktion ausschließlich vom ausgewählten Modus ab. Die Idee ist daher, dass die $modus_{im}$-Transitionen beim Schalten Token entsprechend r_{iqm}^v aus jeder Stelle $nRessource_q$ abziehen, sodass die Transitionen nur schalten können, wenn ausreichend Token in den entsprechenden Stellen liegen. Weil keine Transition Token in die $nRessource_q$-Stellen legt, können die Modi in einem durchführbaren Lauf also nur so ausgewählt werden, dass die maximalen Vorkommen der nicht erneuerbaren Ressourcen nicht überschritten werden. Es werden also Kanten von jeder Stelle $nRessource_q$ zu jeder Transition $modus_{im}$ gezogen und mit r_{iqm}^v gewichtet. Kanten, die mit 0 gewichtet wären, hätten keinen Effekt und werden deshalb nicht eingefügt (siehe auch Abbildung 23).

Die zweite Einschränkung besagt, dass zu einem beliebigen Zeitpunkt t das Vorkommen einer erneuerbaren Ressource nicht von Aktionen überschritten werden darf, die zu Zeitpunkt t aktiv sind. Im Gegensatz zu nicht erneuerbaren Ressourcen dürfen allerdings die kumulierten Ressourcenbedarfe aller Aktionen das Vorkommen überschreiten. In Bezug auf die Konstruktion ist eine Aktion i aktiv, wenn ihre $start_i$-Transition geschaltet hat, allerdings noch keine ihrer $finish_{im}$-Transitionen mit $m \in [M_i]$. Die Idee ist nun, dass jede Aktion i Ressourcen entsprechend ihres Verbrauches für die Dauer ihrer Bearbeitungszeit „blockiert“. Wie in Abbildung 22 zu sehen ist, vergehen zwischen dem Schalten der $start_i$-Transition und einer $finish_{im}$-Transition genau p_{im} Zeiteinheiten. Dabei ist zu beachten, dass zwischen dem Schalten von $start_i$ und $modus_{im}$ keine Zeiteinheit vergehen kann, weil $modus_{im}$ durch das Schalten von $start_i$ aktiviert wird und wegen $I(modus_{im}) = (0, 0)$ direkt schalten muss. D.h. zwischen $modus_{im}$ und $finish_{im}$ vergehen genau p_{im} Zeiteinheiten. Um die Ressourcen also für die Bearbeitungszeit von Aktion i zu blockieren, können sie von den $modus_{im}$ Transitionen mit $m \in [M_i]$ entnommen und von den Transitionen $finish_{im}$ mit $m \in [M_i]$ zurückgelegt werden. Dabei sollen die Transitionen $modus_{im}$ jeweils nur schalten können, wenn genügend Token in den Stellen liegen, die die erneuerbaren Ressourcen repräsentieren, sodass die Restriktion bezüglich dieser eingehalten wird. Um dies zu modellieren, wird von jeder Stelle $eRessource_k$ jeweils eine Kante zu den Transitionen $modus_{im}$ gezogen, die mit r_{ikm}^e gewichtet ist. Außerdem wird jeweils eine Kante von $finish_{im}$ zu den Stellen $eRessource_k$ gezogen, die mit r_{ikm}^e gewichtet ist. Kanten, die mit 0 gewichtet wären, hätten keinen Effekt und werden deshalb nicht eingefügt. Abbildung 23 veranschaulicht dies, wobei die Funktion der Transition $NoRessource_i$ im weiteren Verlauf dieses Abschnittes erläutert wird. Außerdem werden nur die Ressourcen dargestellt, die auch von der entsprechenden Aktion im entsprechendem Modus benötigt werden.

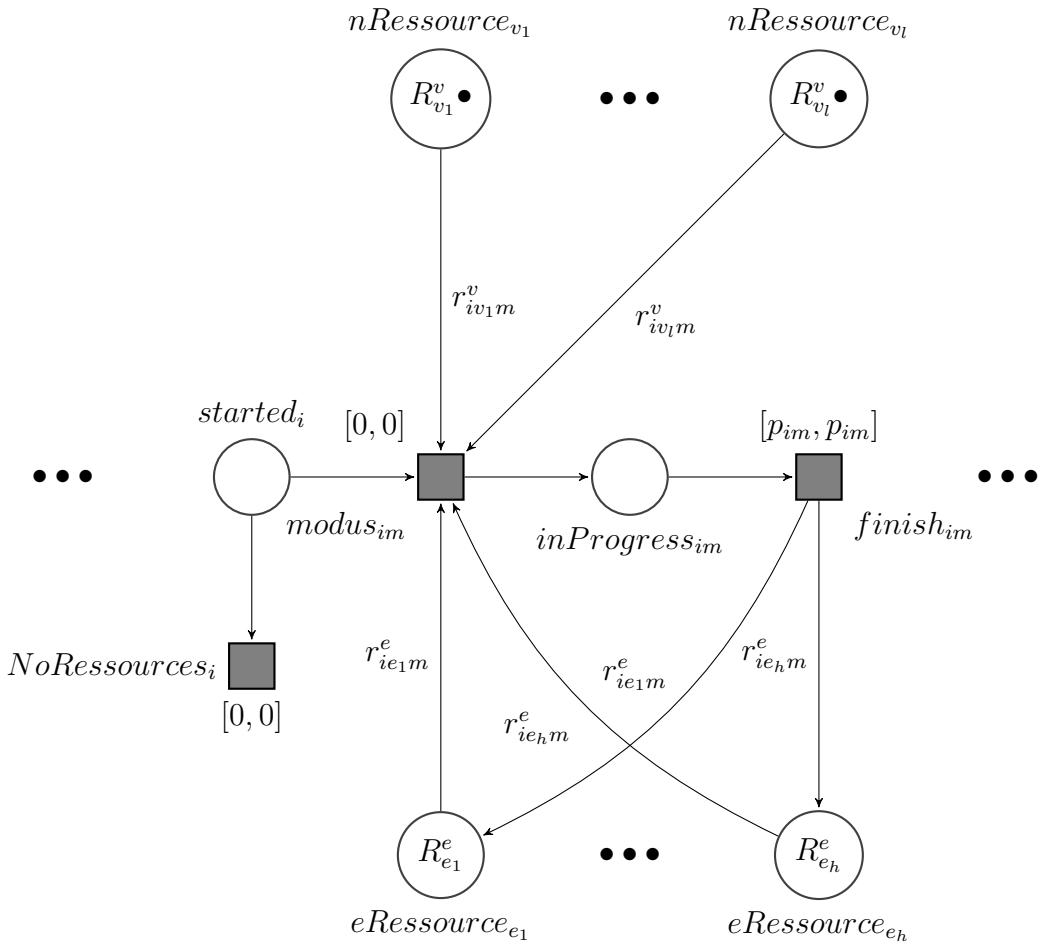


Abbildung 23: Konstruktion der Ressourcenbedarfe einer Aktion i am Beispiel des Modus m. Dabei seien $e_1, \ldots, e_h$ die erneuerbaren und $v_1, \ldots, v_l$ die nicht erneuerbaren Ressourcen, die Aktion i im Modus m benötigt.

An dieser Stelle wird nun auch deutlich, weshalb in Abschnitt 4.3.2 für jede Aktion $i \in [n]$ und jeden Modus $m \in [M_i]$ die beiden Transitionen $modus_{im}$ und $finish_{im}$ eingefügt wurden, denn mit Hilfe dieser Transitionen werden die Ressourcenbedarfe der Aktionen während ihrer Bearbeitungszeit p_{im} modelliert.

Betrachtet man die genannte Modellierung ohne $noRessources_i$-Transition, dann lässt sich damit bereits der Ressourcenverbrauch von Aktionen modellieren. Allerdings werden die Ressourcen, die eine Aktion i benötigt, erst von den Transitionen $modus_{im}$ mit $m \in [M_i]$ entnommen. D.h. Transition $start_i$ kann schalten, ohne Ressourcen zu entnehmen. Auf Grund der Zeitintervalle der Transitionen $modus_{im}$ müssen diese zwar unmittelbar nach ihrer Aktivierung schalten, sodass die Entnahme der Ressourcen zum gleichen Zeitpunkt wie das Schalten der Transition $start_i$ stattfindet. Dies ist allerdings nur der Fall, wenn mindestens eine Transition $modus_{im}$ aktiviert ist. Es ist also möglich, dass die Aktion gestartet wird, ohne dass genügend Ressourcen für das Ausführen eines Modus vorhanden sind. In diesem Fall wäre es möglich, dass durchführbare Läufe im konstruierten zeitbehafteten Petri-Netz exis-

tieren, die zu m_{goal} führen, allerdings die Eigenschaft, dass zwischen dem Schalten der $start_i$ und einer $finish_{im}$ Transition genau p_{im} Zeiteinheiten vergehen (siehe auch Abbildung 22), nicht erfüllen. Abbildung 24 zeigt eine Situation, aus der ein durchführbarer Lauf entstehen kann, in dem zwischen dem Schalten von $start_i$ und $finish_{im}$ mehr als p_{im} Zeiteinheiten vergehen.

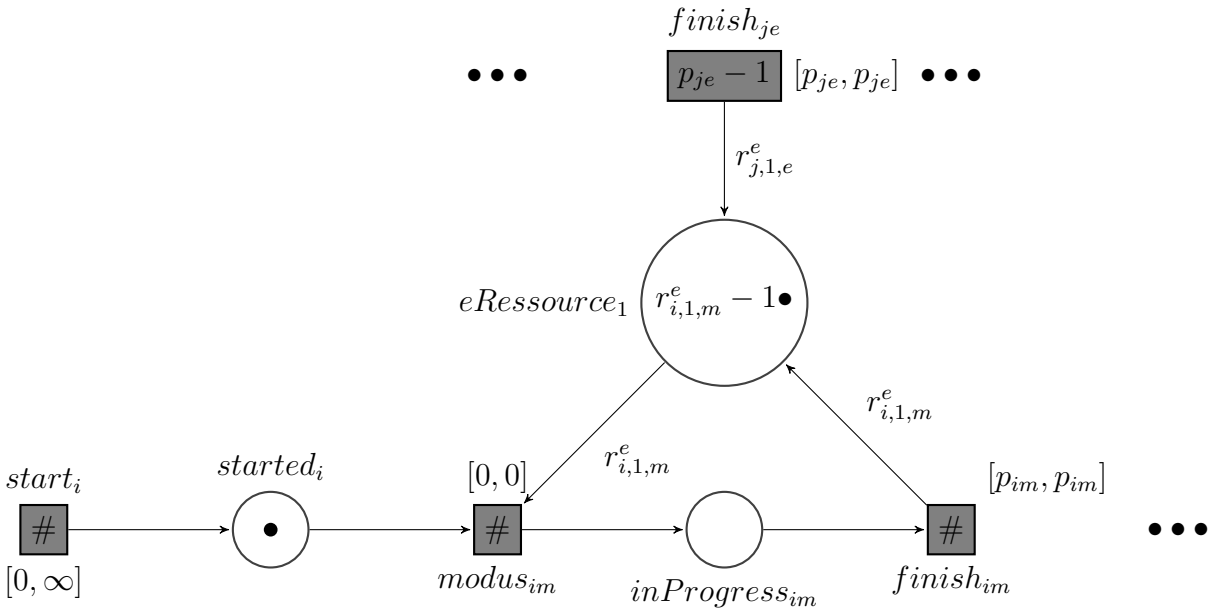


Abbildung 24: Ausschnitt eines Zustand eines konstruierten zeitbehafteten Petri-Netzes, der zu Problemen bezüglich der Ressourcenbedarfe führen kann.

Zu sehen ist ein Ausschnitt der Konstruktion einer Aktion i mit Modus $m \in [M_i]$, die sich eine erneuerbare Ressource 1 mit einer Aktion j im Modus $e \in [M_j]$ teilt. Nehmen wir an, dass Transition $start_i$ gerade geschaltet hat, weshalb sich ein Token in $started_i$ befindet. $modus_{im}$ ist nicht aktiviert, weil die Stelle $eRessource_1$ zu wenige Token hat. $finish_{je}$ ist nicht bereit zu schalten, weil sie noch nicht p_{je} Zeiteinheiten lang aktiviert ist. D.h. in diesem Ausschnitt müsste eine Zeiteinheit vergehen, damit $finish_{je}$ schalten kann. Anschließend kann $modus_{im}$ schalten, weil dann genügend Token in $eRessource_1$ liegen. Daraufhin müssen p_{im} Zeiteinheiten vergehen, bis Transition $finish_{im}$ schalten kann. Damit vergehen zwischen dem Schalten von $start_i$ und $finish_{im}$ dann insgesamt $p_{im} + 1$ Zeiteinheiten, was einen Verstoß gegen die Eigenschaft, die in Abbildung 22 zu sehen ist, darstellt.

Falls also die $start_i$-Transition schaltet, ohne dass genügend Ressourcen vorhanden sind, um eine der $modus_{im}$-Transitionen zu schalten, muss verhindert werden, dass die Markierung m_{goal} erreicht werden kann. Diese Funktion erfüllt die $noRessources_i$-Transition, die das Token aus der $started_i$-Stelle entzieht. Diese Transition bekommt ein Zeitintervall zugewiesen, das bei ihrer Aktivierung nur das sofortige Schalten erlaubt. Sind nun genügend Ressourcen vorhanden, kann stattdessen eine $modus_{im}$-Transition schalten, ansonsten muss die Transition $noRessources_i$ schalten, sodass die $modus_{im}$-Transitionen im weiteren Verlauf nicht mehr schalten können. Abbildung 23 veranschaulicht das Vorgehen.

Der folgende Pseudocode veranschaulicht die Konstruktion für eine gegebene Aktion i. In der vollständigen Konstruktion muss das Vorgehen für jede Aktion ausgeführt werden.

Algorithm 5: ModelRessourceUsage(Action i)

Input: Eine Aktion $i \in [n]$
Data: Das RCPSP X, das bisher konstruierte zeitbehaftete Petri-Netz Z
Result: Das zeitbehaftete Petri-Netz Z erweitert um die Modellierung des Ressourcenverbrauch von i

begin

$T := T \cup \{noRessources_i\}$

$F := F \cup \{(started_i, noRessources_i)\}$

$$W(f) := \begin{cases} 1 & \text{, falls } f = (started_i, noRessources_i) \\ W(f) & \text{, sonst} \end{cases}$$

$$I(t) := \begin{cases} (0,0) & \text{, falls } t = noRessources_i \\ I(t) & \text{, sonst} \end{cases}$$

for $k \in [e]$ **do**

  for $m \in [M_i]$ **do**

    if $r^e_{ikm} > 0$ **then**

      $F := F \cup \{(eRessource_k, modus_{im}), (finish_{im}, eRessource_k)\}$

      $$W(f) := \begin{cases} r^e_{ikm} & \text{, falls } f = (eRessource_k, modus_{im}) \\ r^e_{ikm} & \text{, falls } f = (finish_{im}, eRessource_k) \\ W(f) & \text{, sonst} \end{cases}$$

for $q \in [v]$ **do**

  for $m \in [M_i]$ **do**

    if $r^v_{ikm} > 0$ **then**

      $F := F \cup \{(nRessource_q, modus_{im})\}$

      $$W(f) := \begin{cases} r^v_{ikm} & \text{, falls } f = (nRessource_q, modus_{im}) \\ W(f) & \text{, sonst} \end{cases}$$

4.3.4 Modellierung der Vorgängerrelation

Nachdem nun die Aktionen mitsamt ihrer Ressourcenbedarfe modelliert wurden, sollen jetzt die Vorgängerrelationen sowie maximale und minimale Verzögerungen betrachtet werden. Jede Aktion $i \in [n]$ hat die Menge von Vorgängeraktionen $Pre_i = \{a \in [n] | (a, i) \in V\}$, sodass jede Aktion aus Pre_i abgeschlossen sein muss, bevor die Aktion i gestartet werden darf, d.h. $C_j \leq S_i$ für alle $i, j \in [n]$ mit $(j, i) \in V$. In Bezug auf die Konstruktion bedeutet dies, dass $start_i$ in jedem durchführbaren Lauf des konstruierten zeitbehafteten Petri-Netzes, der nach m_{goal} führt, schalten muss, nachdem eine $finish_{jm}$-Transition für jede Aktion $j \in Pre_i$ mit $m \in [M_j]$ geschaltet hat (siehe auch Abbildung 25).

$$\overbrace{\overbrace{C_j = (\sum_{w=0}^{l-1} \tau_w) + (\sum_{w=l}^{a-1} \tau_w) = S_j + p_{jm}}^{\sum_{w=0}^{c-1} \tau_w = S_i \geq C_j}}$$

Durchführbarer Lauf	τ_0	t_1	τ_1	...	$start_j$	τ_l	...	$modus_{jm}$	...	$finish_{jm}$	...	$start_i$
Index	0	1	1	...	l	l	...	b	...	a	...	c

Abbildung 25: Berechnen des Endzeitpunkt C_j einer Aktion j im Modus m und des Startzeitpunktes S_i einer Aktion i. Wenn $start_i$ nach $finish_{jm}$ schaltet, gilt $S_i \geq C_j$.

.

Dies ließe sich einfach modellieren, indem Kanten von den Stellen $finished_j$ der Aktionen $j \in Pre_i$ zur Transition $start_i$ gezogen werden (siehe auch Abbildung 26).

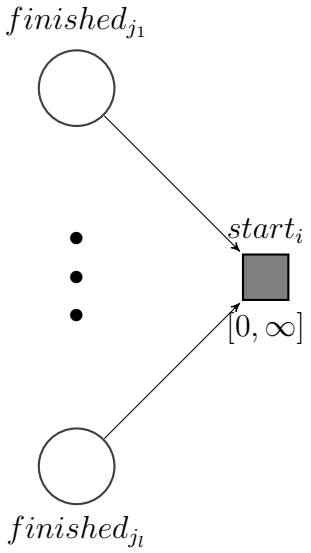


Abbildung 26: Modellierung der Vorgängerrelation einer Aktion i ohne Berücksichtigung der Verzögerungen. Dabei sind $j_1, \ldots, j_l$ die Vorgängeraktionen von i.

Diese Modellierung berücksichtigt allerdings nicht die minimalen bzw. maximalen Verzögerungen, die zwischen Aktion i und ihren Vorgängeraktionen bestehen.

Um dies zu ändern, sei für jedes Tupel (j, i) mit $i \in [n]$ und $j \in Pre_i$ d_{ji} die minimale und $\hat{d}_{ji}$ die maximale Verzögerung. Für einen zulässigen Schedule (S, m) muss dann für jedes Tupel $(j, i) \in V$

$$S_j + p_{jm_j} + d_{ji} \leq S_i \text{ und } S_j + p_{jm_j} + \hat{d}_{ji} \geq S_i$$

gelten. Für die Konstruktion bedeutet dies, dass in einem durchführbaren Lauf des konstruierten zeitbehafteten Petri-Netzes, der nach m_{goal} führt, zwischen dem Schalten von Transition $finish_{jm}$, mit $j \in [n]$ und $m \in [M_j]$, und Transition $start_i$, mit $i \in [n]$ und $(j,i) \in V$, Zeiteinheiten der Anzahl x vergehen müssen mit $d_{ji} \leq x \leq \hat{d}_{ji}$. Abbildung 27 illustriert dies.

$$\overbrace{C_j + \hat{d}_{ji} \geq C_j + x = S_i = C_j + x \geq C_j + d_{ji}}$$

$$\overbrace{C_j = S_j + p_{jm}} \qquad \overbrace{d_{ji} \leq x \leq \hat{d}_{ji}}$$

Durchführbarer Lauf	τ_0	t_1	τ_1	...	$start_j$	τ_l	...	$modus_{jm}$	...	$finish_{jm}$	...	$start_i$
Index	0	1	1	...	l	l	...	b	...	a	...	c

Abbildung 27: Berechnen des Endzeitpunkt C_j einer Aktion j im Modus m und des Startzeitpunktes S_i einer Aktion i. Wenn zwischen dem Schalten von $start_i$ und $finish_{jm}$ $x = \sum_{w=a}^{c-1} \tau_w$ Zeiteinheiten vergehen, mit $d_{ji} \leq x \leq \hat{d}_{ji}$, dann werden die Verzögerungen im resultierenden Schedule eingehalten.

Um die Einhaltung der Verzögerungen zu modellieren, wird daher für jedes Tupel (j,i) mit $i \in [n]$ und $j \in Pre_i$ eine Stelle $legalDelay_{ji}$ eingeführt. Außerdem werden keine direkten Kanten mehr zwischen $finished_j$ und $start_i$ gezogen, sondern zwischen $legalDelay_{ji}$ und $start_i$. Das Ziel ist dabei, dass die Stelle $legalDelay_{ji}$ genau dann ein Token enthält, wenn die Verzögerungen zwischen Aktion j und Aktion i nicht verletzt sind, sodass $start_i$ dann schalten kann, wenn die Verzögerungen zu allen Vorgängeraktionen eingehalten werden.

Aus der minimalen Verzögerung ergibt sich, dass $legalDelay_{ji}$ erst dann ein Token bekommen darf, wenn d_{ji} Zeiteinheiten nach Beendigung der Aktion j vergangen sind. D.h. wenn d_{ji} Zeiteinheiten vergangen sind seitdem $finished_j$ ein Token bekommen hat, muss dieses Token aus $finished_j$ entnommen und $legalDelay_{ji}$ hinzugefügt werden. Dazu wird für jedes Tupel (j,i) mit $i \in [n]$ und $j \in Pre_i$ zusätzlich eine Transition $minDelay_{ji}$ hinzugefügt, die eben jene Funktionalität erfüllt (siehe auch Abbildung 28).

Aus der maximalen Verzögerung ergibt sich, dass $legalDelay_{ji}$ ein Token entnommen werden muss, wenn $\hat{d}_{ji}$ Zeiteinheiten seit Beendigung der Aktion j vergangen sind. Sobald also $\hat{d}_{ji}$ Zeiteinheiten vergangen sind, seitdem $finished_j$ ein Token erhalten hat, muss ein Token aus $legalDelay_{ji}$ entfernt werden. Dabei ist zu beachten, dass, nachdem $finished_j$ ein Token erhält, wegen der minimalen Verzögerung zunächst d_{ji} Zeiteinheiten vergehen, bevor $legalDelay_{ji}$ ein Token erhält. D.h. nachdem $legalDelay_{ji}$ ein Token erhält, dürfen noch $\hat{d}_{ji} - d_{ji}$ Zeiteinheiten vergehen, bevor das Token aus $legalDelay_{ji}$ entfernt werden muss. Um dies zu erreichen, wird für jedes Tupel (j,i) mit $i \in [n]$ und $j \in Pre_i$ zusätzlich eine Transition $maxDelay_{ji}$ hinzugefügt, die genau $\hat{d}_{ji} - d_{ji}$ Zeiteinheiten nach ihrer Aktivierung schalten muss und ein Token aus $legalDelay_{ji}$ entzieht. Abbildung 28 veranschaulicht die Konstruktion für eine Aktion i und ihre Vorgängeraktionen $j_1, \ldots, j_l$. Zu beachten ist dabei,

dass laut Definition 9 $d_{ji} \leq \hat{d}_{ji}$ für alle $(j,i) \in V$ gilt, weshalb $(\hat{d}_{ji} - d_{ji}) \in \mathbb{N}_0$ gilt und die Verwendung dieses Wertes in einem Zeitintervall erlaubt ist.

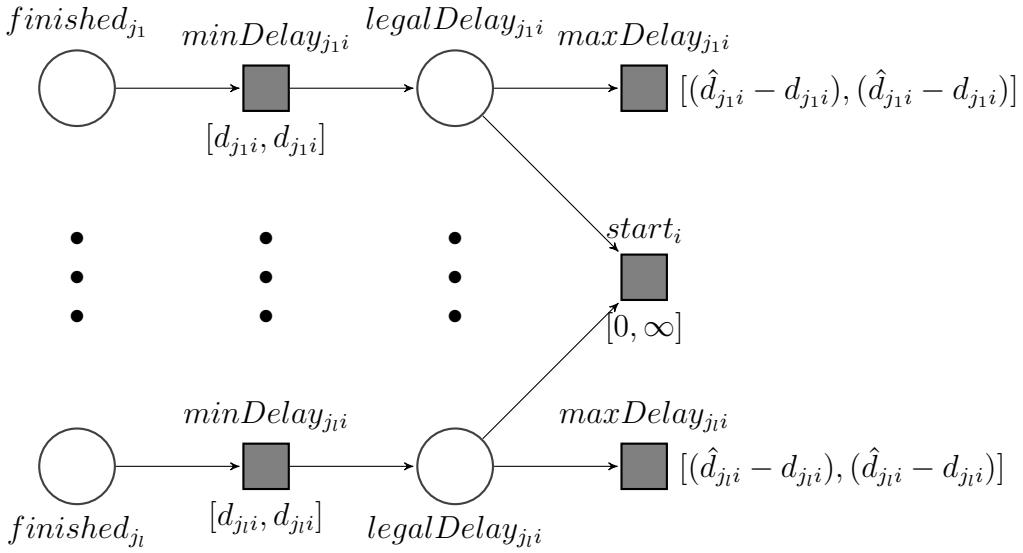
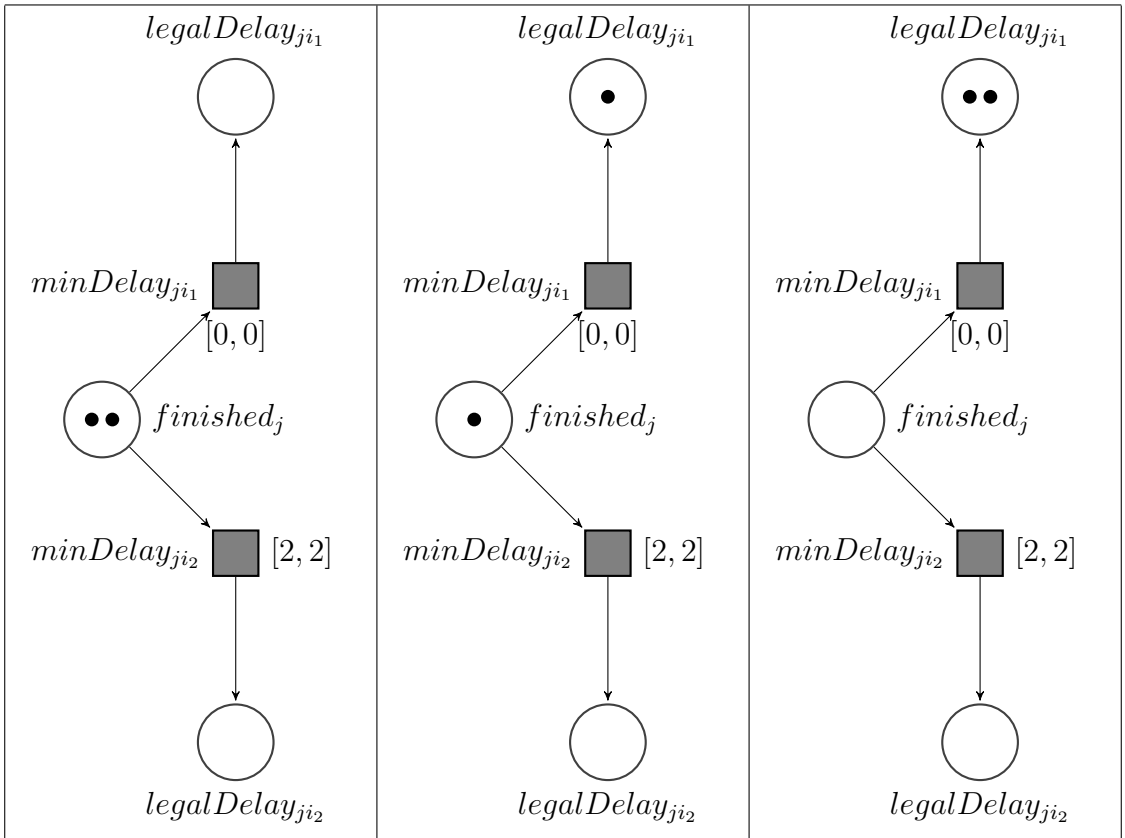


Abbildung 28: Modellierung der Vorgängerrelation einer Aktion i unter Berücksichtigung der Verzögerungen.

Auf diese Weise wird die Anforderung, die in Abbildung 27 dargestellt wird, zwar prinzipiell erfüllt, es treten allerdings technische Probleme auf, die im Folgenden behandelt werden.

Nehmen wir an, dass eine Aktion j genau zwei Nachfolgeaktionen i_1 und i_2 hat, wobei die minimalen Verzögerungen gegeben seien durch $d_{ji_1} = 0$ bzw. $d_{ji_2} = 2$, dann ist laut bisheriger Konstruktion die Aktion j abgeschlossen, sobald sich in $finished_j$ genau zwei Token befinden. Wenn das der Fall ist, ergibt sich eine Situation, wie sie in Abbildung 29 zu sehen ist. Die beiden Transitionen $minDelay_{ji_1}$ und $minDelay_{ji_2}$ sind – in TPN_1 und TPN_2 aus der Abbildung – jeweils aktiviert, wegen der Zeitintervalle muss allerdings $minDelay_{ji_1}$ jeweils vor $minDelay_{ji_2}$ schalten. Daraus folgt, dass in einer Konstruktion nach den bisherigen Überlegungen (siehe auch Abbildung 28) Situationen existieren, in denen die $start_i$-Transition einer Nachfolgeaktion i von j nie bereit ist zu schalten, weil die Transition $minDelay_{ji}$ nie schalten kann.



TPN_1 : Die Transitionen $minDelay_{ji_1}$ und $minDelay_{ji_2}$ sind beide aktiviert. Auf Grund der zugehörigen Zeitintervalle muss allerdings zunächst $minDelay_{ji_1}$ schalten	TPN_2 (TPN_1 nach dem Schalten von $minDelay_{ji_1}$) : Die Transitionen $minDelay_{ji_1}$ und $minDelay_{ji_2}$ sind beide aktiviert. Auf Grund der zugehörigen Zeitintervalle muss allerdings zunächst $minDelay_{ji_1}$ schalten	TPN_3 (TPN_2 nach dem Schalten von $minDelay_{ji_1}$) : Keine der beiden Transitionen ist aktiviert.

Abbildung 29: Illustriert ein Problem der Konstruktion nach Abbildung 28

Dieses Problem lässt sich beheben, indem für jede Aktion j und jede Aktion $i \in Post_j$ eine Stelle $notStartedDelay_{ji}$ hinzugefügt wird, die Vorstelle von $minDelay_{ji}$ ist und initial ein Token enthält (siehe auch Abbildung 30). Dadurch ergibt sich, dass die $minDelay_{ji}$ Transitionen jeweils nur genau einmal schalten können, wodurch oben beschriebene Situation nicht mehr auftreten kann.

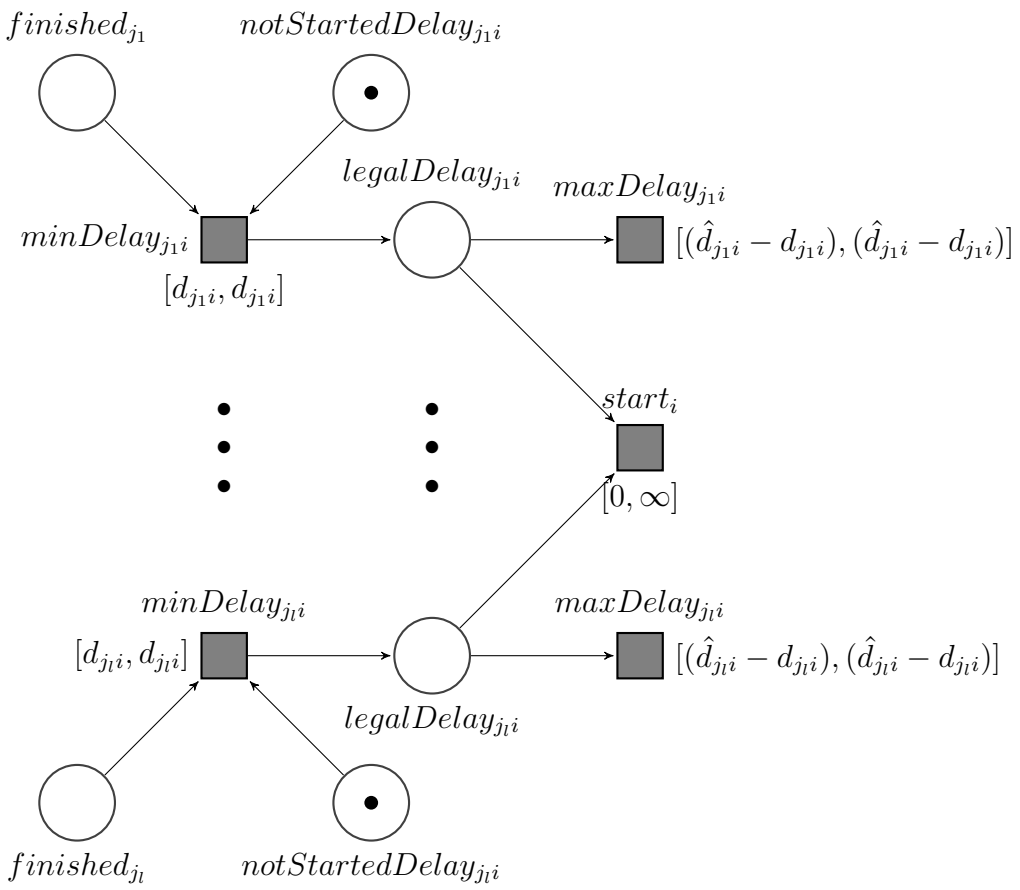


Abbildung 30: 1. Verbesserung der Modellierung der Vorgängerrelation einer Aktion i unter Berücksichtigung der Verzögerungen.

Doch auch mit der Konstruktion, die in Abbildung 30 dargestellt wird, ergeben sich ungewollte Effekte, die aus der Definition des Schaltens von zeitbehafteten Petri-Netzen resultieren (siehe auch Definition 28). Diese besagt, dass nach dem Schalten einer Transition t eines zeitbehafteten Petri-Netzes von Zustand $z = (m, h)$ in Zustand $z' = (m', h')$, mit m, m' als p-Markierungen und h, h' als t-Markierungen des Petri-Netzes, gilt, dass $h'(t') = 0$ für alle Transitionen t', die m- und m'-aktiviert sind und gemeinsame Vorstellen mit t haben. Diese Tatsache kann zu Problemen mit der bisher beschriebenen Modellierung der Vorgängerrelation führen.

Nehmen wir an, eine Aktion j hat genau zwei Nachfolger i_1, i_2, wobei die minimale Verzögerung gegeben sei durch $d_{ji_1} = 1$ und $d_{ji_2} = 2$. Sei außerdem die Aktion j beendet, sodass gerade zwei Token in Stelle $finished_j$ gelegt wurden. Laut aktueller Konstruktion ergibt sich dann eine Situation, wie sie in TPN_1 aus Abbildung 31 zu sehen ist. Im Zustand, der in TPN_1 dargestellt ist, muss zunächst eine Zeiteinheit vergehen, bis $minDelay_{ji_1}$ schalten kann. Daraus ergibt sich die Situation, die in TPN_2 dargestellt ist. Nach dem Schalten von $minDelay_{ji_1}$ (siehe TPN_3), wird der Wert der t-Markierung, die sich durch das Schalten ergibt, für die Transition $minDelay_{ji_2}$ auf 0 gesetzt, weil die beiden Transitionen gemeinsame Vorstellen haben.

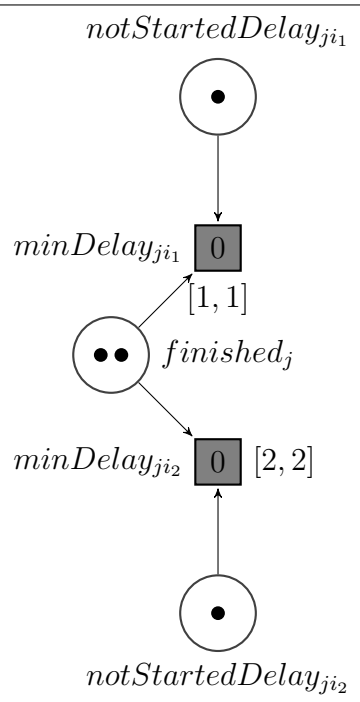
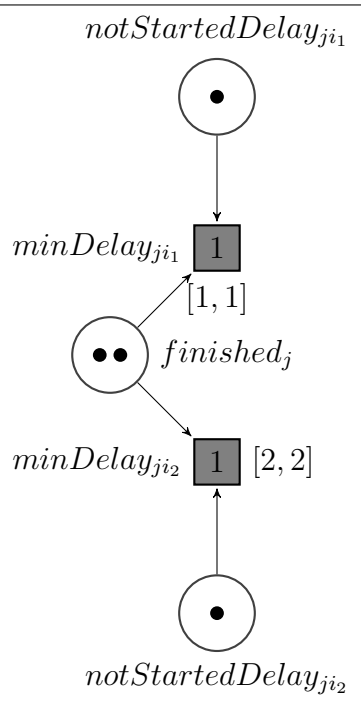
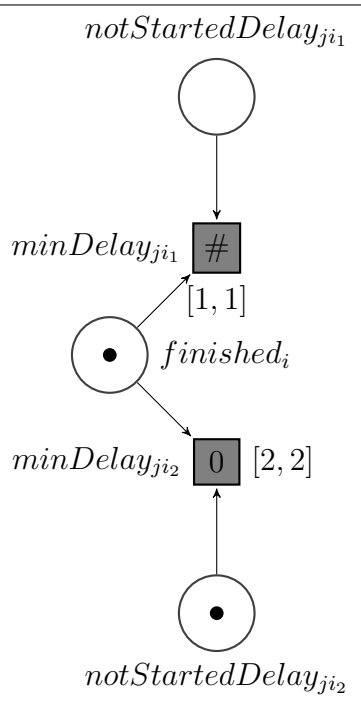
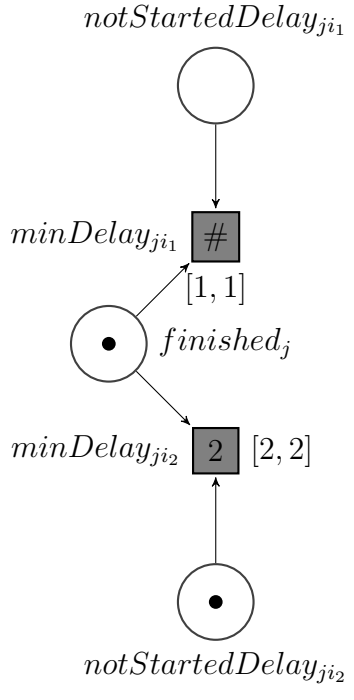
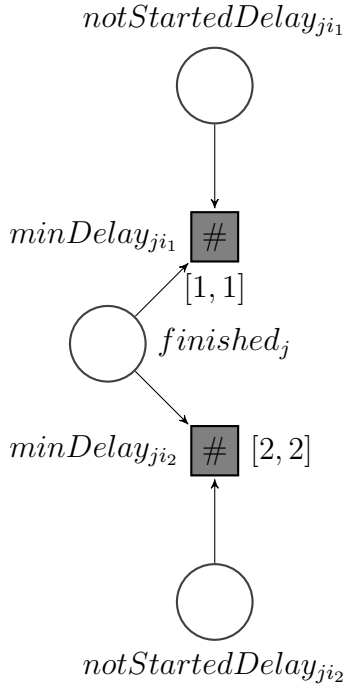
		
TPN_1 : Ausschnitt des konstruierten zeitbehafteten Petri-Netzes, nachdem Aktion j „beendet“ wurde.	TPN_2: TPN_1 nachdem eine Zeiteinheit vergangen ist.	TPN_3 : TPN_2 nachdem Transition $minDelay_{ji_1}$ geschaltet hat.
		
TPN_4 : TPN_3 nachdem zwei Zeiteinheiten vergangen sind.	TPN_5 : TPN_4 nach dem Schalten von Transition $minDelay_{ji_2}$.	

Abbildung 31: Illustriert ein Problem der Konstruktion nach Abbildung 30

Bevor $minDelay_{ji_2}$ schalten kann, müssen also nochmal 2 Zeiteinheiten vergehen (siehe TPN_4). D.h. zwischen erstmaliger Aktivierung und dem Schalten von Transition $minDelay_{ji_2}$ vergehen in dieser Situation mindestens 3 Zeiteinheiten, obwohl die minimale Verzögerung zwischen j und i_2 den Wert 2 hat. Dieser Effekt ist unerwünscht, weil er zum Beispiel im Fall $d_{ji} = \hat{d}_{ji}$ dazu führen kann, dass in einem Schedule, der aus einem durchführbaren Lauf, der nach m_{goal} führt, berechnet wurde, die maximale Verzögerung nicht eingehalten wird.

Dieses Problem lässt sich beheben, indem dafür gesorgt wird, dass die Transitionen $minDelay_{ji}$ einer Aktion j und ihrer Nachfolger i keine gemeinsamen Vorstellen mehr haben. Um dies zu erreichen, werden die Kanten von $finished_j$ zu den Transitionen $minDelay_{ji}$ entfernt. Stattdessen wird jeweils eine neue Transition $startDelay_{ji}$ eingefügt, die eine eingehende Kante von $finished_j$ sowie eine ausgehende Kante in eine ebenfalls neue Stelle $startedDelay_{ji}$ hat. Von $startedDelay_{ji}$ nach $minDelay_{ji}$ wird ebenfalls eine Kante gezogen. Dabei ist zu beachten, dass vor dem Schalten der Transitionen $startDelay_{ji}$ keine weiteren Zeiteinheiten vergehen dürfen, damit zwischen der Aktivierung von $startDelay_{ji}$ und dem Schalten von $minDelay_{ji}$ jeweils genau d_{ji} Zeiteinheiten vergehen. Aus diesem Grund wird das Zeitintervall der Transitionen jeweils auf $[0, 0]$ gesetzt. Abbildung 32 illustriert dies.

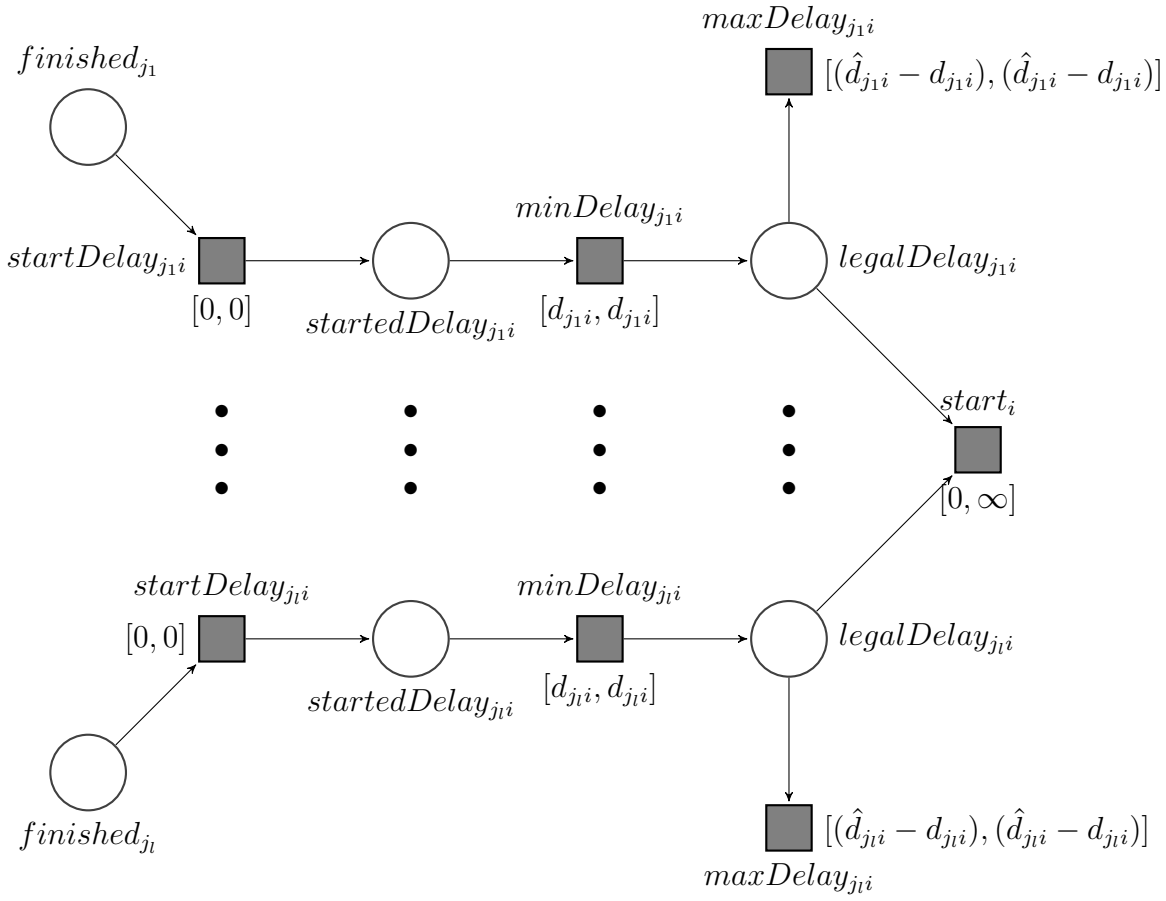


Abbildung 32: 2.Verbesserung der Vorgängerrelation einer Aktion i unter Berücksichtigung der Verzögerungen.

Jetzt haben die Transitionen $minDelay_{ji}$ zwar keine gemeinsamen Vorstellen mehr, dafür aber die Transitionen $startDelay_{ji}$. Da das Zeitintervall der Transitionen jeweils auf $[0, 0]$ gesetzt wird, stellt dies kein Problem dar, denn ein Zurücksetzen des h-Wertes einer aktivierten Transition verändert den Wert der Transition nicht, weil dieser vorher ohnehin 0 gewesen sein muss.

Betrachtet man noch einmal die Situation aus Abbildung 29, dann stellt man fest, dass diese Situation mit der Konstruktion aus Abbildung 32 nicht mehr vorkommen kann. Denn sei j eine Aktion, dann gibt es für jede Nachfolgeaktion i von j eine Transition $startDelay_{ji}$, die $finished_j$ als Vorstelle hat. Diese Transitionen haben jeweils ein Zeitintervall von $[0, 0]$, weshalb keine der Transitionen $startDelay_{ji}$ zwingend vor einer anderen geschaltet werden muss. Daraus resultiert, dass die $notStartedDelay_{ji}$ Vorstelle, die in Abbildung 30 eingeführt wurde, nicht mehr benötigt wird.

Der Pseudocode 6 veranschaulicht die Konstruktion für ein gegebenes Tupel $(j, i) \in V$ mit $i, j \in [n]$. Dabei ist zu beachten, dass der Pseudocode für jedes Tupel $t \in V$ ausgeführt werden muss.

Algorithm 6: ModelPredecessorRelationAndDelays(PredecessorTuple (j, i))

Input: Ein Tupel $(j, i) \in V$ mit $i, j \in [n]$
Data: Das RCPSP X, das bisher konstruierte zeitbehaftete Petri-Netz Z
Result: Das zeitbehaftete Petri-Netz Z erweitert um die Modellierung der Vorgängerrelation (j, i) sowie der Verzögerungen zwischen Aktion j und Aktion i

begin

$P := P \cup \{legalDelay_{ji}, startedDelay_{ji}\}$
$T := T \cup \{minDelay_{ji}, maxDelay_{ji}, startDelay_{ji}\}$
$F := F \cup \{(finished_j, startDelay_{ji})\}$
$\cup\{(startDelay_{ji}, startedDelay_{ji}), (startedDelay_{ji}, minDelay_{ji})\}$
$\cup\{(minDelay_{ji}, legalDelay_{ji}), (legalDelay_{ji}, maxDelay_{ji}), (legalDelay_{ji}, start_i)\}$

$$W(f) := \begin{cases} 1 & \text{, falls } f = (finished_j, startDelay_{ji}) \\ 1 & \text{, falls } f = (startDelay_{ji}, startedDelay_{ji}) \\ 1 & \text{, falls } f = (startedDelay_{ji}, minDelay_{ji}) \\ 1 & \text{, falls } f = (minDelay_{ji}, legalDelay_{ji}) \\ 1 & \text{, falls } f = (legalDelay_{ji}, maxDelay_{ji}) \\ 1 & \text{, falls } f = (legalDelay_{ji}, start_i) \\ W(f) & \text{, sonst} \end{cases}$$

$$I(t) := \begin{cases} (d_{ji}, d_{ji}) & \text{, falls } t = minDelay_{ji} \\ (\hat{d}_{ji} - d_{ji}, \hat{d}_{ji} - d_{ji}) & \text{, falls } t = maxDelay_{ji} \\ (0, 0) & \text{, falls } t = startDelay_{ji} \\ I(t) & \text{, sonst} \end{cases}$$

$$m_0(p) := \begin{cases} 0 & \text{, falls } p = legalDelay_{ji} \\ 0 & \text{, falls } p = startedDelay_{ji} \\ m_0(p) & \text{, sonst} \end{cases}$$

Für jede Nachfolgeaktion i einer Aktion j wird also ein Token aus $finished_j$ entnommen. Dabei ist zu beachten, dass die Aktion j mehrere Nachfolgeaktionen haben kann. Jede der

$startDelay_{ji}$-Transitionen dieser Nachfolgeaktionen muss ein Token aus $finished_j$ entnehmen, damit die jeweilige Nachfolgeaktion durchgeführt werden kann. Sei $Post_j = \{b \in [n] | (j,b) \in V\}$ die Menge der Nachfolgeaktionen einer Aktion j, dann müssen beim Ausführen dieser mindestens $\#Post_j$ Token in die Stelle $finished_j$ gelegt werden. D.h. für eine Aktion j und die Modi $m \in [M_j]$ muss jede Transition $modus_{jm}$ mindestens $\#Post_j$ Token in die Stelle $finished_j$ legen. Dies stellt den Grund dar, weshalb in Abschnitt 4.3.2 entsprechende Kanten mit $max\{\#Post_j, 1\}$ gewichtet wurden. Sollte eine Aktion keine Nachfolgeaktionen haben, werden die entsprechenden Kanten mit 1 gewichtet, denn, wie in der Einleitung dieses Kapitels erwähnt wurde, soll in der Markierung m_{goal} jede $finished_j$ Stelle, deren Aktion j keine Nachfolgeaktionen hat, ein Token enthalten.

4.3.5 Modellierung der zu erreichenden p-Markierung

Mit Hilfe der eingeführten Konstruktion ist es nun möglich, ein zeitbehaftetes Petri-Netz aus dem gegebenen RCPSP zu erhalten. Im Kontext der Ziele der Konstruktion wurde erläutert, dass auch eine Markierung m_{goal} konstruiert werden soll, die genau dann erreichbar sein soll, wenn das gegebene RCPSP eine Lösung hat. Dabei soll aus einem durchführbaren Lauf, der zu m_{goal} führt, ein Schedule berechenbar sein.

Damit dies mit der eingeführten Methode möglich ist, darf Markierung m_{goal} nur erreichbar sein, wenn die $start_i$- und $modus_{im}$- Transitionen für jede Aktion $i \in [n]$ und jeweils einen Modus $m \in [M_i]$ im durchführbaren Lauf vorkommen. Um dies zu erreichen, wird definiert, dass in Markierung m_{goal} die Stellen $finished_i$ der Aktionen $i \in [n]$ genau dann ein Token enthalten sollen, wenn sie keine Nachfolgeaktionen haben, also $Post_i = \emptyset$. Alle anderen Stellen sollen keine Token enthalten.

Wenn dies der Fall ist, müssen genannte Transitionen für alle Aktionen im durchführbaren Lauf enthalten sein. Denn die $finished$-Stellen der Aktionen ohne Nachfolgeaktionen würden keine Token enthalten, wenn die Transitionen nicht für die entsprechenden Aktionen geschaltet hätten. Damit diese Transitionen schalten konnten, mussten alle $finished_j$ Stellen der Vorgängeraktionen $j \in Pre_i$ Token enthalten haben, die aber entzogen wurden, was wiederum nur möglich ist, wenn die genannten Transitionen für entsprechende Aktionen geschaltet haben. Diese Kausalität setzt sich fort bis zu den Aktionen, die keine Vorgänger haben.

Pseudocode 7 veranschaulicht die Konstruktion von m_{goal}.

Algorithm 7: ModelMGoal()

Data: Das RCPSP X, das vollständige konstruierte zeitbehaftete Petri-Netz Z
Output: Die Markierung m_{goal}
begin

$$m_{goal}(p) := \begin{cases} 1 & \text{, falls } p = finished_i \text{ für eine Aktion } i \in [n] \text{ mit } Post_i = \emptyset \\ 0 & \text{, sonst} \end{cases}$$

Sobald alle $finished$-Stellen, deren Aktionen keine Nachfolger haben, ein Token erhalten haben, lässt sich ein Schedule nach eingeführter Methode berechnen. Die Tokenzahl der übrigen Stellen spielt in dem Kontext im Prinzip keine Rolle.

Um m_{goal} allerdings formal in Hinblick auf das Erreichbarkeits-Problem verwenden zu können, muss sie für alle Stellen angeben, wie viele Token diese haben sollen. Dies stellt insofern ein Problem dar, dass die Ressourcen-Stellen noch Token enthalten können, wenn alle Aktionen abgeschlossen sind und die genaue Zahl dieser nicht bekannt ist. Deshalb wird die Konstruktion um Transitionen erweitert, die es erlauben, Token aus den Stellen zu entziehen, die Ressourcenvorkommen repräsentieren. Für jede Ressource $k \in [e]$ bzw. $q \in [v]$ wird also eine Transition $eRemove_k$ bzw. $nRemove_q$ hinzugefügt, die jeweils die Vorstelle $eRessource_k$ bzw. $nRessource_q$ hat und aus dieser beim Schalten genau ein Token entnimmt. Mit Hilfe der neuen Transitionen können also jederzeit alle Token aus den Ressourcen-Stellen entzogen werden, sodass spezifiziert werden kann, dass diese in Markierung m_{goal} keine Token enthalten. Der Pseudocode 8 bzw. 9 veranschaulicht dies für eine erneuerbare Ressource $k \in [e]$ bzw. nicht erneuerbare Ressource $q \in [v]$.

Algorithm 8: ModelRemovalRenewableRessource(RenewableRessource k)

Input: Eine erneuerbare Ressource $k \in [e]$
Data: Das RCPSP X, das bisher konstruierte zeitbehaftete Petri-Netz Z
Result: Das zeitbehaftete Petri-Netz Z erweitert um das Entfernen der erneuerbaren Ressource k
begin

$$T := T \cup \{eRemove_k\};$$
$$F := F \cup \{(eRessource_k, eRemove_k)\}$$
$$W(f) := \begin{cases} 1 & \text{, falls } f = (eRessource_k, eRemove_k) \\ W(f) & \text{, sonst} \end{cases}$$
$$I(t) := \begin{cases} (0, \infty) & \text{, falls } t = eRemove_k \\ I(t) & \text{, sonst} \end{cases}$$

Algorithm 9: ModelRemovalNonRenewableRessource(NonRenewableRessource k)

Input: Eine nicht erneuerbare Ressource $q \in [v]$
Data: Eine nicht erneuerbare Ressource q, das RCPSP X, das bisher konstruierte zeitbehaftete Petri-Netz Z
Result: Das zeitbehaftete Petri-Netz Z erweitert um das Entfernen der nicht erneuerbaren Ressource q
begin

$$T := T \cup \{nRemove_q\};$$
$$F := F \cup \{(nRessource_q, nRemove_q)\}$$
$$W(f) := \begin{cases} 1 & \text{, falls } f = (nRessource_q, nRemove_q) \\ W(f) & \text{, sonst} \end{cases}$$
$$I(t) := \begin{cases} (0, \infty) & \text{, falls } t = nRemove_q \\ I(t) & \text{, sonst} \end{cases}$$

Dabei ist zu beachten, dass der Pseudocode in der vollständigen Konstruruktion für jede

erneuerbare Ressource $k \in [e]$ bzw. nicht erneuerbare Ressource $q \in [v]$ ausgeführt werden muss.

Mit der Konstruktion der Entfernung der Ressourcen sowie der zu erreichenden Markierung kann im Folgenden die vollständige Konstruktion formuliert werden.

4.3.6 Vollständige Konstruktion

In diesem Abschnitt werden schließlich die einzelnen Teil der Konstruktion zusammengefügt. Dabei lassen sich folgende Abhängigkeiten zwischen den einzelnen Teilen identifizieren, die die Reihenfolge der Ausführung beeinflussen:

- Bevor der Ressourcenverbrauch der Aktionen modelliert werden kann, müssen die Aktionen und Ressourcen bereits modelliert sein.
- Für die Modellierung der Vorgängerrelation und Verzögerungen, müssen die Aktionen bereits modelliert worden sein.
- Die Modellierung der Ressourcen und des Entfernens der Ressourcen muss jeweils für jede erneuerbare bzw. nicht erneuerbare Ressource durchgeführt werden. Dies kann jeweils in einer einzelnen Schleife geschehen.
- Die Konstruktion der Aktionen sowie der Ressourcennutzung der Aktionen muss jeweils pro Aktion durchgeführt werden, deshalb kann dies in einer einzelnen Schleife geschehen.
- m_{goal} kann erst modelliert werden, wenn das zeitbehaftete Petri-Netz komplett konstruiert wurde.

Algorithm 10: transformation(RCPSP X)

Input: Ein RCPSP X
Result: Ein zeitbehaftetes Petri-Netz Z und eine Markierung m_{goal}
begin
  $Z := (P, T, F, W, m_0, I)$
  $initialization()$
  for $k \in [e]$ **do**
    $ModelRenewableRessource(k)$
    $ModelRemovalRenewableRessource(k)$
  for $q \in [v]$ **do**
    $ModelNonRenewableRessource(q)$
    $ModelRemovalNonRenewableRessource(q)$
  for $i \in [n]$ **do**
    $ModelAction(i)$
    $ModelRessourceUsage(i)$
  for $t = (j, i) \in V$ **do**
    $ModelPredecessorRelationAndDelays((j, i))$
  $m_{goal} := modelMGoal()$

Aus diesen Überlegungen ergibt sich Pseudocode 10, der die komplette Konstruktion beschreibt. Die vollständige Konstruktion wird im Folgenden anhand eines Beispiels veranschaulicht.

4.3.7 Beispiel

Gegeben sei das RCPSP $X = (D, [n], [e], [v], R^e, R^v, M, p, r^e, r^v, V, d, \hat{d})$, das mit Hilfe von Tabellen und Vorgängerrelation-Graph in Abbildung 33 beschrieben ist. Zusätzlich sei der Zeithorizont durch $D = 15$ gegeben.

Modus	Bearbeitungszeiten und Ressourcenbedarfe				
1	Aktionen		2	3	4
	Bearbeitungszeit		3	2	5
	Verbrauch Ressource	1	20	0	0
		1 (nicht erneuerbar)	0	40	80
2	Aktionen			3	4
	Bearbeitungszeit			3	10
	Verbrauch Ressource	1		0	0
		1 (nicht erneuerbar)		20	60

Ressourcen	1	1 (nicht erneuerbar)
Vorkommen	30	80

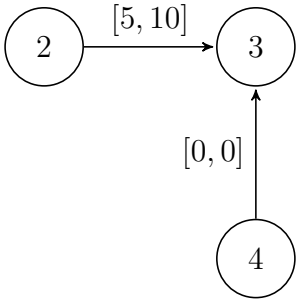


Abbildung 33: Beispiel für ein RCPSP mit Multiple Modes, Verzögerungen und nicht erneuerbaren Ressourcen.

Führt man die Konstruktion für das RCPSP aus, ergibt sich das zeitbehaftete Petri-Netz $Z = (P, T, F, W, m_0, I)$, das in Abbildung 34 dargestellt wird.

Neben dem zeitbehafteten Petri-Netz liefert die Konstruktion auch die p-Markierung m_{goal}. Betrachtet man den Vorgängerrelation-Graph, dann fällt auf, dass lediglich Aktion 3 keine Nachfolgeaktionen hat. Laut Konstruktion ist $m_{goal} : P \to \mathbb{N}$ also definiert durch:

$$m_{goal}(p) = \begin{cases} 1 & , \text{ falls } p = finished_3 \\ 0 & , \text{ sonst} \end{cases}$$

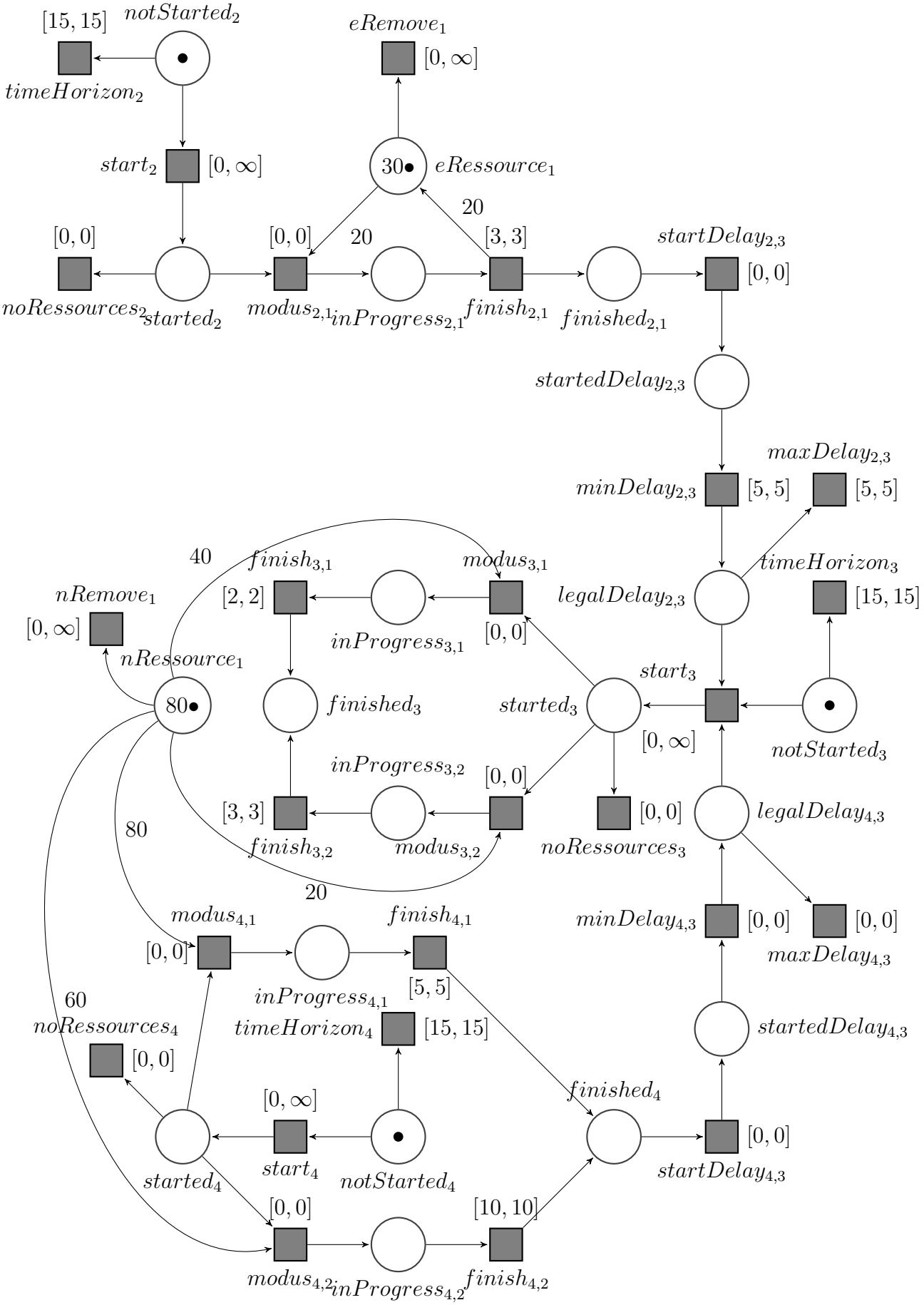


Abbildung 34: Konstruiertes zeitbehaftetes Petri-Netz zum Beispiel, das in Abbildung 33 beschrieben wird.

Um einige Aspekte des konstruierten zeitbehafteten Petri-Netzes näher zu veranschaulichen, betrachten wir nun einige durchführbare Läufe in Z, die vom initialen Zustand z_0 ausgehen. Aus diesen Läufen werden entsprechend der Methode, die in Abbildung 17 skizziert ist, Schedules berechnet. Dabei wird auffallen, dass sich aus den Läufen, die nicht zu m_{goal} führen und sich auch nicht so erweitern lassen, dass sie m_{goal} erreichen, nur nicht zulässige Schedules berechnen lassen, während sich aus anderen Läufen zulässige Schedules berechnen lassen. Tabelle 4 zeigt einen ersten durchführbaren Lauf in Z.

Durchführbarer Lauf		Kommentar	
Abstrakt	Konkret		
τ_0	0		
t_1	$start_2$		
τ_1	0		
t_2	$modus_{2,1}$	$m_2 = 1$	
τ_2	3		
t_3	$finish_{2,1}$		
τ_3	0		
t_4	$startDelay_{2,3}$		
τ_4	0		
t_5	$start_4$		
τ_5	0		
t_6	$modus_{4,1}$	$m_4 = 1$	Entnimmt 80 Token aus Stelle $nRessource_1$
τ_6	5		
t_7	$finish_{4,1}$		
τ_7	0		
t_8	$minDelay_{2,3}$		
τ_8	0		
t_9	$startDelay_{4,3}$		
τ_9	0		
t_{10}	$minDelay_{4,3}$		
τ_{10}	0		
t_{11}	$start_3$		
τ_{11}	0		$modus_{3,1}$ und $modus_{3,2}$ können wegen fehlender Token in $nRessource_1$ nicht schalten.
t_{12}	$noRessources_3$		Deshalb muss $noRessources_3$ schalten. Danach kann m_{goal} nicht mehr erreicht werden.

Tabelle 4: Beispiel für einen durchführbaren Lauf im Petri-Netz aus Abbildung 34, der nicht zu m_{goal} führt.

In diesem Lauf schaltet unter anderem die Transition $modus_{4,1}$, die 80 Token aus Stelle $nRessource_1$ entnimmt. Laut Methode zum Berechnen eines Schedules aus einem Lauf in

Z wird in diesem Schedule Aktion 4 in Modus 1 gestartet. Betrachtet man die Ressourcenbedarfe im Beispiel, dann fällt auf, dass in dem Schedule Aktion 3 nicht mehr ausgeführt werden kann. Für Aktion 3 ergeben sich zwei Fälle:

1. Aktion 3 wird in Modus 1 ausgeführt, dann gilt $\sum_{i=1}^{3} r^v_{i,1,m_i} = 0+40+80 = 120 > 80 = R^v_1$. Damit wird das Vorkommen der nicht erneuerbaren Ressource 1 überschritten. Der Schedule ist also nicht zulässig.
2. Aktion 3 wird in Modus 2 ausgeführt, dann gilt $\sum_{i=1}^{3} r^v_{i,1,m_i} = 0+20+60 = 100 > 80 = R^v_1$. Damit wird das Vorkommen der nicht erneuerbaren Ressource 1 überschritten. Der Schedule ist also nicht zulässig.

In beiden Fällen ergibt sich kein zulässiger Schedule. Folgerichtig können die Transitionen $modus_{3,1}$ und $modus_{3,2}$ in besagtem Lauf nicht schalten, weshalb Transition $noRessources_3$ schalten muss und m_{goal} nicht erreicht wird und auch nicht mehr erreicht werden kann.

Tabelle 5 zeigt einen weiteren durchführbaren Lauf in Z.

Durchführbarer Lauf		Kommentar
Abstrakt	Konkret	
τ_0	0	$\left.\vphantom{\tau_0}\right\} S_2 = \sum_{w=0}^{1-1} \tau_w = 0$
t_1	$start_2$	$\left.\vphantom{\tau_0}\right\} S_4 = \sum_{w=0}^{2-1} \tau_w = 0$
τ_1	0	
t_2	$start_4$	
τ_2	0	
t_3	$modus_{4,1}$	$m_4 = 1$
τ_3	0	
t_4	$modus_{2,1}$	$m_2 = 1$
τ_4	3	
t_5	$finish_{2,1}$	
τ_5	0	
t_6	$startDelay_{2,3}$	
τ_6	2	
t_7	$finish_{4,1}$	
τ_7	0	
t_8	$startDelay_{4,3}$	
τ_8	0	
t_9	$minDelay_{4,3}$	
τ_9	0	
t_{10}	$maxDelay_{4,3}$	
τ_{10}	3	
t_{11}	$minDelay_{2,3}$	
τ_{11}	5	
t_{12}	$maxDelay_{2,3}$	

Tabelle 5: Beispiel für einen durchführbaren Lauf im zeitbehafteten Petri-Netz aus Abbildung 34, der nicht zu m_{goal} führt.

In diesem Lauf werden die Transitionen $start_2$, $start_4$, $modus_{4,1}$ und $modus_{2,1}$ geschaltet, woraus sich die Werte S_2, S_4, m_4, m_2 des Schedules berechnen lassen. Auch dieser Schedule kann für keine Belegung von S_3 und m_3 zulässig werden. Betrachte dazu, die gegebenen Verzögerungen $\hat{d}_{2,3} = 10$ und $d_{4,3} = 0$. Für einen zulässigen Schedule müsste gelten:

1.

$$\begin{aligned} & S_2 + p_{2,m_2} + \hat{d}_{2,3} \leq S_3 \\ \Rightarrow\ & S_2 + p_{2,1} + \hat{d}_{2,3} \leq S_3 \\ \Rightarrow\ & 0 + 3 + 10 = 13 \leq S_3 \end{aligned}$$

2.

$$\begin{aligned} & S_4 + p_{4,m_4} + d_{4,3} \geq S_3 \\ \Rightarrow & S_4 + p_{4,1} + d_{4,3} \\ \Rightarrow & 0 + 5 + 0 = 5 \geq S_3 \end{aligned}$$

Damit müsste gelten, $13 \leq S_3 \leq 5$, was einen Widerspruch darstellt. Konsequenterweise kann ein zulässiger Lauf, der die angegebene Schaltfolge aus Tabelle 5 verlängert, niemals $start_3$ enthalten, denn $maxDelay_{4,3}$ und $maxDelay_{2,3}$ haben bereits geschaltet, sodass $start_3$ nicht mehr aktiviert werden kann. Dementsprechend erhält $finished_3$ nie ein Token und m_{goal} wird nicht erreicht.

Tabelle 6 zeigt einen weiteren durchführbaren Lauf in Z. Aus diesem durchführbaren Lauf lässt sich ein Schedule ablesen. Dieser Schedule ist zulässig, denn es gilt:

- $S_4 + p_{4,m_4} + d_{4,3} = S_4 + p_{4,2} + d_{4,3} = 0 + 10 + 0 = 10$
- $S_2 + p_{2,m_2} + d_{2,3} = S_2 + p_{2,1} + d_{2,3} = 0 + 3 + 5 = 8$
- $S_4 + p_{4,m_4} + \hat{d}_{4,3} = S_4 + p_{4,2} + \hat{d}_{4,3} = 0 + 10 + 0 = 10$
- $S_2 + p_{2,m_2} + \hat{d}_{2,3} = S_2 + p_{2,1} + \hat{d}_{2,3} = 0 + 3 + 10 = 13$
- $S_3 = 10$

Damit werden die Verzögerungen eingehalten, weil dann auch gilt:

- $S_4 + p_{4,m_4} + d_{4,3} = 10 \leq 10 = S_3$
- $S_4 + p_{4,m_4} + \hat{d}_{4,3} = 10 \geq 10 = S_3$
- $S_2 + p_{2,m_2} + d_{2,3} = 8 \leq 10 = S_3$
- $S_2 + p_{2,m_2} + \hat{d}_{2,3} = 13 \geq 10 = S_3$

Zusätzlich werden auch die Ressourcenvorkommen nicht überschritten. Betrachte dazu zunächst die erneuerbare Ressource 1. Diese wird nur von Aktion 2 verwendet. Weil diese nur 20 der insgesamt verfügbaren 30 Einheiten verwendet, wird das Vorkommen zu keinem Zeitpunkt überschritten. Die nicht erneuerbare Ressource 1 wird von Aktion 3 und 4 benötigt, wobei die Erstere im Modus 2 genau 20 und Letztere im Modus 2 genau 60 Einheiten benötigt. Damit werden insgesamt 80 Einheiten verwendet, wodurch das maximale Vorkommen von 80 nicht überschritten wird.

Es ist außerdem leicht zu sehen, dass alle Startpunkte innerhalb des Zeithorizonts liegen.

Durchführbarer Lauf		Kommentar		
Abstrakt	Konkret			
τ_0	0	$S_4 = \sum_{w=0}^{1-1} \tau_w = 0$	$S_2 = \sum_{w=0}^{2-1} \tau_w = 0$	$S_3 = \sum_{w=0}^{11-1} \tau_w = 10$
t_1	$start_4$			
τ_1	0			
t_2	$start_2$			
τ_2	0			
t_3	$modus_{2,1}$	$m_2 = 1$		
τ_3	0			
t_4	$modus_{4,2}$	$m_4 = 2$		
τ_4	3			
t_5	$finish_{2,1}$			
τ_5	0			
t_6	$startDelay_{2,3}$			
τ_6	5			
t_7	$minDelay_{2,3}$			
τ_7	2			
t_8	$finish_{4,2}$			
τ_8	0			
t_9	$startDelay_{4,3}$			
τ_9	0			
t_{10}	$minDelay_{4,3}$			
τ_{10}	0			
t_{11}	$start_3$			
τ_{11}	0			
t_{12}	$modus_{3,2}$	$m_3 = 2$		
τ_{12}	3			
t_{13}	$finish_{3,2}$			

Tabelle 6: Beispiel für einen durchführbaren Lauf im zeitbehafteten Petri-Netz aus Abbildung 34. Der durchführbare Lauf kann so erweitert werden, dass er zu m_{goal} führt

Der angegebene Lauf führt zu einer p-Markierung, in der nur die Stellen $finished_3$ und $eRessource_1$ Token enthalten. Der Lauf kann also mit Hilfe von Schaltungen der $eRemove_1$-

Transition so verlängert werden, dass er zu einer p-Markierung führt, in der nur noch die Stelle $finished_3$ Token enthält. Dies Markierung entspricht dann m_{goal}.

5 Fazit und Ausblick

Nachdem eine Einführung in Resource-constrained project scheduling problems mit Modi, erneuerbaren Ressourcen und Verzögerungen (RCPSP) sowie in zeitbehaftete Petri-Netze gegeben wurde, wurde in dieser Arbeit eine Konstruktion präsentiert, die es erlaubt, Erstere mit Letzteren zu modellieren. Dabei wurde das Erreichbarkeits-Problem der zeitbehafteten Petri-Netze genutzt, um eine „genau dann, wenn"-Beziehung zwischen beiden Modellen herzustellen, indem die angegebene Konstruktion aus jedem RCPSP X ein Tupel (Z,m_{goal}) konstruiert, wobei Z ein zeitbehaftetes Petri-Netz und m_{goal} eine p-Markierung von Z ist, sodass m_{goal} genau dann erreichbar in Z ist, wenn X eine Lösung hat. Zusätzlich wurde ein Verfahren eingeführt, das es ermöglicht, aus durchführbaren Läufen in Z, die zu m_{goal} führen, zulässige Lösungen für das ursprünglich gegebene RCPSP zu berechnen.

Während zwar andere Arbeiten existieren, die sich ebenfalls mit der Modellierung von Scheduling-Problemen mit Hilfe von Petri-Netzen beschäftigen, grenzt sich diese Arbeit von ihnen in der Wahl der zu modellierenden Probleme und der verwendeten Petri-Netz-Variante ab. So gibt beispielsweise [1] eine Modellierung mit Hilfe von timed Petri Nets anstelle von zeitbehafteten Petri-Netzen an, wobei ein anderes Scheduling-Problem zugrunde liegt, das zum Beispiel keine Verzögerungen und erneuerbare Ressourcen beinhaltet und allgemein eine andere Art der Ressourcen-Modellierung verwendet. Dadurch unterscheidet sich die Modellierung – abgesehen von kleineren Ähnlichkeiten – deutlich von der Konstruktion aus dieser Arbeit. Auch (der Artikel) [11] beschäftigt sich mit Scheduling-Problemen und Petri-Netzen, wobei Job Shop Scheduling und timed Petri Nets zugrunde liegen, wodurch ein deutlicher Unterschied zum Ansatz dieser Arbeit besteht. Zusätzlich existiert noch eine Reihe von Artikeln, die sich mit Modellierung und Scheduling von *Flexible Manufacturing Problems* beschäftigen. Ein Beispiel dafür ist der Artikel [10], der sich neben einem anderen betrachteten Problem auch durch die Wahl der Petri-Netz-Variante von dieser Arbeit unterscheidet. [18] fasst außerdem eine Anzahl an Arbeiten zusammen, die sich – im Gegensatz zu dieser Arbeit – mit Produktions-Scheduling befassen und dabei unterschiedlichste Petri-Netz-Varianten verwenden.

Für die weitere Arbeit an den Methoden, die in dieser Arbeit vorgestellt wurden, ergeben sich zwei Ansatzpunkte:

- Erweiterung der Konstruktion
- Nutzung der Konstruktion

Wir haben bereits gesehen, dass sich die Konstruktion, die in dieser Arbeit vorgestellt wurde, von vielen anderen Modellierungen im betrachteten Problem unterscheidet. Um diese Abgrenzung weiter auszubauen, könnte die Konstruktion in weiterführenden Arbeiten dahingehend erweitert werden, dass RCPSPs mit zusätzlichen Erweiterungen behandelt werden können. [9], [2] und [7] listen solche Erweiterungen für RCPSPs auf.

In Bezug auf den zweiten Ansatzpunkt wurde bereits in der Einleitung die Möglichkeit, die Konstruktion zur Analyse von Scheduling-Problemen zu verwenden, als Motivation für die Durchführung dieser Arbeit genannt. Später wurde ausgeführt, dass das Erreichbarkeits-Problem zur Ausführung einer Analyse genutzt werden kann. Eine Voraussetzung für die erfolgreiche Umsetzung ist es dabei, dass die Konstruktion tatsächlich korrekt ist, also dass die „genau dann, wenn“-Beziehung wie geplant vorliegt. In Kapitel 4.3 wurden zwar Argumente und Begründungen für die Korrektheit der Konstruktion geliefert und auch das in Abschnitt 4.3.7 betrachtete Beispiel bekräftigt diese, um allerdings eine formale Bestätigung der Korrektheit zu erhalten, könnte in weiterführenden Arbeiten ein Korrektheitsbeweis durchgeführt werden.

Um schließlich Analysemethoden nutzen zu können, muss die Konstruktion für gegebene RCPSPs durchgeführt werden. Betrachtet man das Beispiel aus Abschnitt 4.3.7, dann fällt auf, dass das konstruierte zeitbehaftete Petri-Netz bereits für sehr kleine Scheduling-Probleme einen großen Umfang annimmt, sodass eine Durchführung der Konstruktion „per Hand“ schnell nicht mehr in Frage kommt. Ein nächster Punkt für weitere Arbeiten ist also eine Implementierung, die eine Eingabe von RCPSPs ermöglicht, die Konstruktion durchführt und das Ergebnis in einem geeigneten Format ausgibt.

Ein letzter Ansatz zur Nutzung der Konstruktion ist es schließlich, die Analysemethoden für zeitbehafete Petri-Netze auf Beispielprobleme anzuwenden und die Ergebnisse mit Analysemethoden für Scheduling-Probleme zu vergleichen, um eine Einschätzung zu gewinnen, inwiefern sich die Nutzung der Konstruktion und Analysemethoden für Scheduling-Probleme lohnt. Dazu könnten Software-Tools, wie zum Beispiel der in [4] präsentierte Model-Checker „Romeo“, und andere Verfahren, wie zum Beispiel die in [16] vorgestellten, genutzt werden.

Literatur

[1] W. M. P. van der Aalst. „Petri net based scheduling“. In: *Operations-Research-Spektrum* 18.4 (1996), S. 219–229. DOI: 10.1007/BF01540160.

[2] Christian Artigues. „The Resource-Constrained Project Scheduling Problem“. In: *Resource-Constrained Project Scheduling*. ISTE, 2010, S. 19–35. DOI: 10.1002/9780470611227.ch1.

[3] F.D.J Bowden. „A brief survey and synthesis of the roles of time in petri nets“. In: *Mathematical and Computer Modelling* 31.10 (2000), S. 55–68. DOI: http://dx.doi.org/10.1016/S0895-7177(00)00072-8.

[4] Guillaume Gardey u. a. „Romeo: A Tool for Analyzing Time Petri Nets“. In: *Computer Aided Verification: 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005. Proceedings*. Hrsg. von Kousha Etessami und Sriram K. Rajamani. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, S. 418–423. DOI: 10.1007/11513988_41.

[5] Michael R. Garey und David S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. A series of books in the mathematical sciences. X, 338 S. ; 24 cm : Ill. New York, NY: Freeman, 1979.

[6] Claude Girault und Rüdiger Valk. *Petri nets for systems engineering: a guide to modeling, verification, and applications ; with 9 tables*. Berlin [u.a.]: Springer, 2003.

[7] Sönke Hartmann und Dirk Briskorn. „A survey of variants and extensions of the resource-constrained project scheduling problem“. In: *European Journal of Operational Research* 207.1 (2010), S. 1–14. DOI: http://dx.doi.org/10.1016/j.ejor.2009.11.005.

[8] Rajeev Motwani; Jeffrey D. Ullman; John E. Hopcroft. *Introduction to automata theory, languages, and computation*. 2. ed. Boston, Mass. [u.a.]: Addison-Wesley, 2001.

[9] Sigrid Knust und Peter Brucker. *Complex scheduling*. 2nd ed. GOR-publications. New York: Springer, 2012.

[10] Doo Yong Lee und F. DiCesare. „Scheduling flexible manufacturing systems using Petri nets and heuristic search“. In: *IEEE Transactions on Robotics and Automation* 10.2 (Apr. 1994), S. 123–132. DOI: 10.1109/70.282537.

[11] Santosh Krishnaji Sindhe Mullya Satish Anand. „Modeling and Simulation of Job Shop Scheduling Using Petri-Nets“. In: *International Journal of Engineering Research and Applications (IJERA)* 3.2 (März 2013), S. 492–496.

[12] T. Murata. „Petri nets: Properties, analysis and applications“. In: *IEEE Transactions on Power Systems blank page* 76.4 (1988), S. 541–580.

[13] Florian Jaehn; Erwin Pesch. *Ablaufplanung: Einführung in Scheduling*. Berlin [u.a.]: Springer Gabler, 2014.

[14] Carl Adam Petri. „Kommunikation mit Automaten“. ger. Diss. Universität Hamburg, 1962.

[15] Michael L. Pinedo. *Scheduling: theory, algorithms, and systems*. 4. ed. New York [u.a.]: Springer, 2012.

[16] Louchka Popova-Zeugmann. *Time and Petri Nets*. Springer-Verlag Berlin Heidelberg, 2013.

[17] Wolfgang Reisig. *Understanding Petri nets: modeling techniques, analysis methods, case studies*. Berlin [u.a.]: Springer, 2013.

[18] Gonca Tuncel und G. Mirac Bayhan. „Applications of Petri nets in production scheduling: a review“. In: *The International Journal of Advanced Manufacturing Technology* 34.7 (2007), S. 762–773. DOI: 10.1007/s00170-006-0640-1.