



Diplomarbeit

Laufzeitersetzung offener Workflownetze

Nannette Liske

15. Juli 2008

Gutachter:

- Prof. Dr. Karsten Wolf
Institut für Informatik, Universität Rostock
- Prof. Dr. Wolfgang Reisig
Institut für Informatik, Humboldt-Universität zu Berlin

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit genannten Quellen benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches gekennzeichnet.

Ich bin damit einverstanden, dass ein Exemplar dieser Arbeit in der Bibliothek des Instituts für Informatik der Humboldt-Universität zu Berlin ausgestellt wird.

Berlin, den 15. Juli 2008.

Nannette Liske

Inhaltsverzeichnis

Abbildungsverzeichnis	7
1 Einleitung	9
1.1 Motivation	9
1.2 Ziele der Arbeit	10
1.3 Verwandte Themen	11
1.4 Gliederung	12
2 Grundlagen	13
2.1 Modelle	13
2.1.1 Petrinetze	14
2.1.2 Offene Workflownetze	15
2.1.3 Serviceautomaten	17
2.2 Bedienungsanleitung	21
3 neue Definitionen	25
3.1 Wissensfunktionen	25
3.1.1 Erreichbarkeitsfunktion	25
3.1.2 Fortführbarkeitsfunktion	26
3.2 Laufzeitersetzung	31
3.2.1 Eingabeplätze neu belegen	33
4 Instantane Laufzeitersetzungen	35
4.1 Algorithmus zur Bestimmung instantaner Laufzeitersetzungen	37
4.1.1 Notwendigkeit der Überprüfung auf Fortführbarkeit	40
4.2 Einschränkungen der Netze	41
4.2.1 Leere Transitionen im inneren Netz	41
4.2.2 Überdeckte Endmarkierungen	42
4.2.3 Überdeckte Markierungen	43
4.3 Korrektheit des Algorithmus INSTANTANELE	45
5 Nebenläufige Laufzeitersetzungen	47
5.1 Algorithmus zum Erzeugen nebenläufiger Laufzeitersetzungen	48
5.2 Korrektheit	53
5.2.1 Leerer Vorbereich	53
5.2.2 Nicht überführbare Markierungen	55

Inhaltsverzeichnis

5.2.3	Korrektheit des Algorithmus ZERTEILUNG	57
5.3	Heuristik	58
5.3.1	differenzierte Deadlockprüfung	59
5.3.2	Verkleinerung der Menge B	64
5.4	Größenreduktion durch nebenläufige Laufzeitersetzungen	66
6	Fazit	69
	Literaturverzeichnis	71

Abbildungsverzeichnis

2.1	oWFNs zweier Getränkeautomaten	17
2.2	Partnerautomat für das Netz N_1 aus Abb. 2.1	19
2.3	Komposition von N_1 aus Abb. 2.1 und R aus Abb. 2.2	20
2.4	Bedienungsanleitung für das Netz N_1 aus Abb. 2.1	22
2.5	Teilmenge von Strategien (\sqsubseteq -Relation)	23
3.1	oWFN mit Erreichbarkeitsfunktion	26
3.2	Berechnung der Fortführbarkeitsfunktion	27
3.3	umgekehrtes Netz und umgekehrter Automat	28
3.4	oWFN mit Fortführbarkeitsfunktion	29
3.5	Fortführbarkeitsfunktion für nicht bedienbares oWFN	29
3.6	oWFN mit Fortführbarkeitsfunktion bei unbeschränkter Umkehrung	30
3.7	Laufzeitersetzung offener Workflownetze gemäß <i>OG</i> aus Abb. 2.5	33
3.8	Laufzeitersetzung, die keinen Eingabeplatz belegt	34
4.1	instantane Laufzeitersetzung gemäß <i>OG</i> von Abb. 2.4	36
4.2	Berechnung der instantanen Laufzeitersetzung	38
4.3	Wissensfunktionen K_1 und K_2 für die Netze aus Abb. 4.1	38
4.4	Ausschnitt der Wissensfunktionen für Abb. 4.3	39
4.5	vollständige instantane Laufzeitersetzung	39
4.6	Notwendigkeit der Überprüfung in Zeile 9	40
4.7	leere Transitionen im inneren Netz	41
4.8	überdeckte Endmarkierung	43
4.9	überdeckte Markierung	44
5.1	Berechnung nebenläufiger Laufzeitersetzungen	49
5.2	Zerteilung von Laufzeitersetzungstransitionen	49
5.3	zwei offene Workflownetze	50
5.4	instantane Laufzeitersetzung zu Abb. 5.3	51
5.5	berechnete nebenläufige Laufzeitersetzung	52
5.6	zerteilte Laufzeitersetzungstransitionen zur Tab. 5.4	52
5.7	anders zerteilte Laufzeitersetzungstransitionen zur Tab. 5.4	52
5.8	Test auf leeren Vorbereich	53
5.9	Test auf nicht-überführbare Markierungen	56
5.10	Beispielnetze für Heuristiken	59
5.11	Serviceautomat mit Wissensfunktion für die Netze aus Abb. 5.10	60

Abbildungsverzeichnis

5.12 nicht überführbare Markierungen	60
5.13 modifizierter Ausschnitt aus ZERTEILUNGSSCHRITT	61
5.14 Problembeispiel für Heuristiken	63
5.15 modifizierter Ausschnitt aus ZERTEILUNG	65
5.16 exponentiell viele instantane Laufzeitersetzungstransitionen	67

1 Einleitung

Geschäftsprozesse sind ein standardisiertes Verfahren, um organisatorische Vorgänge zu beschreiben und zu realisieren. Dabei besteht ein Geschäftsprozess aus einer Menge von Aktivitäten, denen eine Ausführungsvorschrift in beliebiger Form zugrundeliegt. Umfasst ein Geschäftsprozess mehrere, untereinander kommunizierende Teile, welche räumlich oder organisatorisch unabhängig voneinander verlaufen, so spricht man auch vom *verteilten Geschäftsprozess*. Die Teile eines solchen Prozesses werden *Services* genannt.

In der Geschäftswelt spielt die *Ersetzbarkeit von Services* durchaus eine große Rolle. Firmenstrukturen unterliegen ständigen Veränderungen, Übernahmen ganzer Geschäftszweige durch andere Firmen sind keine Seltenheit. Mitunter möchte auch eine Firma einen Kundenstamm, oder einen bestimmten Teil davon, in einen neuen (rentableren) Geschäftsprozess übernehmen.

1.1 Motivation

In verschiedenen Arbeiten ist die Ersetzbarkeit von Services bereits untersucht worden (siehe Verwandte Themen). Dabei wurde allerdings oft das Augenmerk nur auf die *Austauschbarkeit*, also die *Ersetzung zur Entwurfszeit*, gerichtet. Das bedeutet, es wurde untersucht, inwiefern ein bestimmter Service anstelle eines anderen verwendet werden kann. In manchen Fällen ist die Ersetzung zur Entwurfszeit jedoch nur bedingt sinnvoll, weil es sich um sehr langlebige oder kritische Vorgänge handelt. Ebenso ist das Szenario denkbar, dass nicht genug Ressourcen vorhanden sind, um alle laufenden Instanzen eines alten Service bis zu Ende zu führen und gleichzeitig alle neuen Interaktionen mit dem neuen Service zu beginnen.

Beispiele für langwierige Geschäftsprozesse bieten häufig jene Abläufe, welche in Behörden und Verwaltungen vorkommen. Bei diesen verteilten Geschäftsprozessen können sich mitunter aufgrund veränderter Gesetzeslagen die Abarbeitungsvorschriften ändern, zum Beispiel bei Lebensversicherungen. In solch einem Fall sollen die laufenden Verträge natürlich nicht gekündigt werden. Vielmehr möchte man das neue Gesetz soweit wie möglich auf die laufenden Services anwenden. Eine erste theoretische Arbeit auf die-

sem Gebiet stammt von Ryu et al. [RCS⁺08]. Die Autoren haben die Veränderungen eines laufenden Service betrachtet und ihre Auswirkungen hinsichtlich seiner Bedienung untersucht.

1.2 Ziele der Arbeit

Die vorliegende Arbeit führt ein neues Konzept zur Ersetzung von Services zur Laufzeit ein. Das gesuchte Szenario erfüllt folgende Bedingungen:

1. Ein Service wird zur Laufzeit in einen anderen Service überführt.
2. Die bislang geleistete Arbeit soll bewahrt bleiben, d.h. ein Abbruch der Kommunikation ist ausgeschlossen.
3. Der Austausch geschieht unbemerkt vom Kommunikationspartner (Kunden).

Oft enthalten Geschäftsprozesse mehrere, zeitlich unabhängige Arbeitsabläufe. Diese nebenläufigen Aspekte sollen sich auch in der *Laufzeitersetzung* der Services widerspiegeln, weshalb die Ziele noch um folgende Punkte ergänzt werden:

4. Sofern möglich, soll die Laufzeitersetzung der Services nebenläufig, in mehreren Schritten, durchgeführt werden.
5. Die Laufzeitersetzung soll sowohl für alle möglichen Kommunikationspartner als auch für einen beliebig ausgewählten Teil bestimmt werden können.

Als Modell zur Beschreibung der Geschäftsprozesse werden *offene Workflownetze* verwendet. Das zu berücksichtigende Verhalten der Partner, die *Strategien*, sind in Form von *Bedienungsanleitungen* bzw. *annotierten Automaten* gegeben. Die Berechnung der sogenannten *Laufzeitersetzungen* erfolgt in zwei Stufen:

- Zunächst wird ein Algorithmus präsentiert und in seiner Korrektheit bewiesen, welcher zu zwei gegebenen offenen Workflownetzen sowie einer als annotierter Automat gegebenen Strategiemenge die *vollständige instantane Laufzeitersetzung* findet.
- Darauf aufbauend kann ein Algorithmus angegeben werden, der *Nebenläufigkeiten* der Workflownetze berücksichtigt. Dazu werden instantane Laufzeitersetzungen derart modifiziert, dass nebenläufige Arbeitsabläufe auch nebenläufig vom einen Netz in das andere überführt werden.

1.3 Verwandte Themen

In der Literatur finden sich viele verschiedene Äquivalenzbegriffe für Geschäftsprozesse bzw. Services. Bereits 1971 definiert Milner die *Simulation* von Prozessen [Mil71]. Es handelt sich dabei um eine Relation zwischen den Zuständen zweier Prozesse (Zustandsautomaten). In [Mil89] erweitert der Autor das Konzept noch um die *Bisimulation* zweier Prozesse.

Neue Äquivalenzbegriffe, welche sich nicht auf die Vorgänge innerhalb der Prozesse beziehen sondern auf die *Bedienung* (Interaktion mit dem Partner), tauchen im Wesentlichen erst nach der Jahrtausendwende in der Literatur auf. Beispielsweise beschreibt Martens in [Mar03] die *Bedienbarkeitsäquivalenz* von Services. Dabei gilt, dass ein Service gegen einen anderen Service zur Entwurfszeit ausgetauscht werden kann, wenn alle *Strategien* des einen auch Strategien des anderen Service sind. Ist die Menge der Strategien beider Services identisch, so heißen sie bedienbarkeitsäquivalent. Auf Grundlage der Bedienbarkeitsäquivalenz beschäftigt sich Richter mit der Frage, ob ein Service durch einen anderen ausgetauscht werden kann, wenn dabei nur eine bestimmte Strategie erhalten bleiben soll [Ric02].

Van Glabbeek gibt in [Gla01] eine Übersicht weiterer gebräuchlicher Äquivalenzbegriffe. Auf Basis dieser Begriffe wurden in [HDvdA⁺05] verschiedene Äquivalenzbegriffe Workflownetze untersucht.

In [SMB08] definieren die Autoren die Austauschbarkeit von Services anhand von *Bedienungsanleitungen* (Operating Guidelines). Ihre Herangehensweise erlaubt die Beschränkung auf eine beliebige Teilmenge von Strategien. Das bedeutet, dass die Services für eine beliebig ausgewählte Klasse von Partnern austauschbar sind. Mithilfe des Software-Tool Fiona [LMSW06] lässt sich diese Form der Austauschbarkeit, und damit auch die *Bedienbarkeitsäquivalenz* [Mar03], effizient bestimmen.

Die bisher genannten Äquivalenzbegriffe für Services bzw. Workflownetze beziehen sich auf die *Austauschbarkeit*, also die Ersetzung zur Entwurfszeit. Seit kurzer Zeit ist auch die *Ersetzung zur Laufzeit* Gegenstand theoretischer Arbeiten. So untersuchen in [RCS⁺08] die Autoren für *Geschäftsprotokolle* (business protocols) – einem zustandsbasierten Modell für Geschäftsprozesse – die Ersetzbarkeit von Zuständen zur Laufzeit. Dabei entscheiden sie, ob ein Kommunikationspartner von einem Zustand p eines solchen Protokolls dasselbe Kommunikationsmuster verwenden kann wie von einem Zustand q eines anderen Protokolls. Lassen sich alle Zustände eines Geschäftsprotokolls P_1 ersetzen durch Zustände von P_2 , dann heißt P_1 *gesamt zur Laufzeit ersetzbar* durch P_2 . Die Autoren betrachten bereits die Beschränkung auf bestimmte Kommunikationspartner und geben auch einen Algorithmus zur Bestimmung der Ersetzungsregeln an.

Im Gegensatz zu zustandsbasierten Geschäftsprotokollen mit synchroner Kommunikation möchten wir auch Prozesse mit asynchroner Kommunikation zur Laufzeit ersetzen können. Als erste Arbeit beschreibt [Lis07] ein solches Szenario – die *Laufzeitersetzung von Serviceautomaten*. Serviceautomaten sind Zustandsautomaten mit asynchronem Nachrichtenkanal. Die Kommunikationspartner sind als Bedienungsanleitung dargestellt. Als Ergebnis der Arbeit kann folgendes Problem algorithmisch gelöst werden: Gegeben seien zwei Serviceautomaten P_1 und P_2 sowie eine Bedienungsanleitung OG . Es soll die *maximale Laufzeitersetzung* von P_1 nach P_2 berechnet werden, wobei als Interaktionspartner nur solche gemäß OG in Frage kommen.

Die vorliegende Diplomarbeit greift das Konzept der Laufzeitersetzung von Serviceautomaten auf und überträgt es auf *offene Workflownetze* [MRS05, LMW07]. Dieses Modell ist in der Lage, nebenläufige Arbeitsabläufe, welche in Geschäftsprozessen häufig vorhanden sind, explizit darzustellen. Die asynchrone Kommunikation wird in offenen Workflownetzen beibehalten.

1.4 Gliederung

Die weitere Arbeit ist folgendermaßen aufgebaut: Das Kapitel *Grundlagen* führt zunächst die zugrundeliegenden Modelle zur Darstellung von Geschäftsprozessen ein. Darauf aufbauend folgen die Definitionen von Wissensfunktionen und Laufzeitersetzung im Abschnitt *neue Definitionen*. Die Wissensfunktionen dienen der Berechnung von Laufzeitersetzungen. Eine einfache Form von Laufzeitersetzungen, die instantanen Laufzeitersetzungen, werden im vierten Kapitel untersucht. Neben der Definition und einigen Beispielen enthält dieser Abschnitt auch einen Algorithmus zur Bestimmung der *vollständigen instantanen Laufzeitersetzung* sowie den Beweis seiner Korrektheit.

Es folgt im Kapitel *Nebenläufige Laufzeitersetzungen* die Erweiterung auf beliebige Laufzeitersetzungen. Die Ergebnisse eines vergleichsweise einfachen Algorithmus werden anhand von erläuternden Beispielen diskutiert. Anschließend präsentieren wir eine heuristische Modifikation des Algorithmus, welche deutlich mehr Nebenläufigkeiten finden kann. Die Korrektheit der Algorithmen wird jeweils formal bewiesen.

Im *Fazit* findet sich eine inhaltliche Zusammenfassung der gewonnenen Erkenntnisse. Daran knüpft ein Ausblick, welcher offene Fragen nennt und Möglichkeiten zur Weiterentwicklung der in dieser Arbeit vorgestellten Konzepte ausführt.

2 Grundlagen

2.1 Modelle

Typischerweise wird ein Service von anderen Services eingebunden oder bindet selbst weitere Services ein. Das Paradigma des *Service-Oriented Computing* (SOC) beschreibt die Interaktion von Services [Pap01]. Eine häufige Form der Implementation von Services stellen *Web Services* [GGKS02] dar. Verschiedene große Firmen, darunter IBM und Microsoft, haben 2002 einen gemeinsamen, XML-basierten Standard zur Spezifikation von Web Services, die *Business Process Execution Language* (BPEL) [AAA⁺06], festgelegt. Es gibt verschiedene Möglichkeiten, den von einem BPEL-Programm beschriebenen Service, mathematisch zu modellieren. Einen guten Überblick bietet [BK06]. *Offene Workflownetze* (oWFN) [MRS05, LMW07] basieren auf den von van der Aalst vorgeschlagenen *Workflownetzen* [Aal98]. Sie stellen eine spezielle Form von Petrinetzen [Rei82] dar und eignen sich besonders gut für theoretische Analyseverfahren [LMSW06]. Ein offenes Workflownetz beschreibt sowohl das interne Verhalten eines Service als auch sein Interface, also die Schnittstelle zur Kommunikation mit anderen Services. Durch die asynchrone Arbeitsweise von Petrinetzen lassen sich die in Geschäftsprozessen auftretenden Nebenläufigkeiten hervorragend darstellen. Mithilfe von *BPEL2oWFN* können BPEL-Programme zu Analysezwecken in offene Workflownetze konvertiert werden [HSS05, Loh08].

Bedienbare Services sind solche, für die es einen Partner-Service gibt, mit dem er sinnvoll interagieren kann [Mar03, Sch05]. Fehler im Design von Services werden somit durch die Untersuchung seiner Bedienbarkeit erkannt. Ein Analysewerkzeug für die Bedienbarkeit offener Workflownetze ist mit *Fiona* gegeben [LMSW06]. Darüberhinaus lässt sich mit *Fiona* die *Bedienungsanleitung* [MS05, LMW07] eines Service konstruieren. Die Bedienungsanleitung charakterisiert alle sinnvoll interagierenden Partner in einer kompakten Darstellung. Weitere Analysen des zugrundeliegenden Petrinetzes lassen sich beispielsweise mit dem *Low Level Analyzer* (LoLA) [Sch00] durchführen .

2.1.1 Petrinetze

Definition 2.1 (Petrinetz)

Ein Petrinetz N ist ein Tupel $N = [P, T, F, W, m_0]$ aus den endlichen, disjunkten Mengen P (Plätze) und T (Transitionen), der Relation $F \subseteq (T \times P) \cup (P \times T)$ (Bögen), der Abbildung $W : F \rightarrow \mathbb{N}$ (Bogenvielfachheit) sowie einer initialen Markierung m_0 .

Für nicht-vorhandene Bögen, also Paare $[x, y] \notin F$, vereinbaren wir, dass der Ausdruck $W([x, y])$ dem Wert 0 entspricht.

Wir sagen, in einem Petrinetz gibt es von einem Element x (Platz oder Transition) einen Pfad zu einem Element y , falls es weitere Elemente $x_0 \dots x_n \in P \cup T$ gibt, sodass $x = x_0$, $y = x_n$ und $[x_i, x_{i+1}] \in F$.

Definition 2.2 (Markierung)

Als $\text{bags}(X)$ bezeichnen wir die Menge aller Multimengen über X , das sind alle Abbildungen $X \rightarrow \mathbb{N}$. Eine Markierung m eines Petrinetzes ist eine Multimenge von Plätzen des Netzes, also $m \in \text{bags}(P)$. Wir schreiben für Markierungen auch kurz $m = [p_0, \dots, p_n]$, wobei jeder Platz entsprechend häufig der Anzahl seines Vorkommens in der Multimenge auftritt. Die Markierung $[]$ heißt leere Markierung.

Jede Markierung $m : P \rightarrow \mathbb{N}$ entspricht kanonisch einer Markierung m_{\supseteq} auf einer größeren Platzmenge $P' \supseteq P$ mit $m_{\supseteq}(p) = \begin{cases} m(p), & p \in P \\ 0, & \text{sonst} \end{cases}$. Zur Vereinfachung der Schreibweise ist im folgenden bei größeren Platzmengen einfach von m die Rede.

Grundlegende Rechenoperationen und Vergleiche lassen sich auf Markierungen übertragen, indem sie platzweise angewendet werden. In diesem Sinne sind die Summe bzw. die Differenz von Markierungen zu verstehen als Summe bzw. Differenz von Multimengen:

$$\left. \begin{aligned} (m_1 + m_2)(p) &:= m_1(p) + m_2(p) \\ (m_1 - m_2)(p) &:= m_1(p) - m_2(p) \\ m_1 \leq m_2 &\Leftrightarrow m_1(p) \leq m_2(p) \end{aligned} \right\} \forall p \in P$$

Die grafische Darstellung von Petrinetzen erfolgt üblicherweise durch Kreise (für Plätze), Rechtecke (für Transitionen) und Pfeile (für Bögen). Die Bogenvielfachheiten werden als Zahl an den entsprechenden Pfeilen markiert. Eine Ausnahme bilden dabei einfache Bögen – hier wird der Übersichtlichkeit die 1 weggelassen. Bögen mit Vielfachheit 0 werden als nicht vorhanden betrachtet und auch nicht eingezeichnet. Anfangsmarkierung der abgebildeten Netze ist – wenn nicht anders erwähnt – $[p_0]$ bzw. $[v_0]$.

Definition 2.3 (Vor- und Nachbereich)

Der Vorbereich einer Transition t ist die Markierung $\bullet t$ mit $\bullet t(p) = W([p, t])$. Analog ist der Nachbereich definiert als Markierung $t\bullet$ mit $t\bullet(p) = W([t, p])$. Eine Transition mit leerem Vor- und Nachbereich heißt leere Transition.

Definition 2.4 (Schalten von Transitionen)

Eine Transition t ist aktiviert in einer Markierung m , wenn gilt: $m \geq \bullet t$. Befindet sich das Petrinetz in der Markierung m , so kann eine aktivierte Transition t zu einer Folgemarkierung $m' = m - \bullet t + t\bullet$ schalten ($m \xrightarrow{t} m'$).

Definition 2.5 (Erreichbarkeit)

Die Erreichbarkeit von Markierungen ist induktiv definiert. Dabei heißt im Netz N eine Markierung m' von m aus erreichbar, wenn

- $m = m'$ oder
- es eine Transition t mit $m \xrightarrow{t} m''$ gibt und m' von m'' aus erreichbar ist.

Im Petrinetz N sei $R_N(m)$ die Menge aller von m aus erreichbaren Markierungen. Dann enthält $R_N(m_0)$ alle erreichbaren Markierungen des Netzes N .

Definition 2.6 (Beschränktheit)

Ein Petrinetz heißt beschränkt, wenn die Menge aller erreichbaren Markierungen endlich ist. k -beschränkt heißt das Netz, falls bei allen erreichbaren Markierungen kein Platz mehr als k Marken besitzt.

2.1.2 Offene Workflownetze

Workflownetze sind spezielle Petrinetze zur Beschreibung von Geschäftsprozessen. Bei offenen Workflownetzen existiert zusätzlich ein *Interface*. Dabei handelt es sich um Ein- und Ausgabeplätze, welche zur Kommunikation mehrerer offener Workflownetze untereinander dienen. Dadurch lassen sich auch verteilte Geschäftsprozesse theoretisch beschreiben.

Im übertragenen Sinne ist das Interface als Nachrichtenkanal zu verstehen, wobei vom Inhalt der Nachrichten und der Art der Übertragung (Brief, E-Mail, ...) abstrahiert wird auf ihre bloße Existenz. Die Richtung einer Nachricht ist darin kodiert, ob sie auf einem Eingabe- oder einem Ausgabeplatz liegt. Für die Systemanalyse ist diese Modellebene vollkommen ausreichend.

Definition 2.7 (Offenes Workflownetz)

Ein offenes Workflownetz (*oWFN*) $N = [P, T, F, W, m_0, \Omega_N]$ ist eine Erweiterung des Petrinetzes $[P, T, F, W, m_0]$, wobei

- P die disjunkte Vereinigung der Eingabepätze P_I , Ausgabepätze P_O und inneren Plätze P_N darstellt,
- $F \cap (P_O \times T) = \emptyset$ und $F \cap (T \times P_I) = \emptyset$,
- die Anfangsmarkierung m_0 sowie alle Endmarkierungen nur innere Plätze beinhalten und
- in keiner Endmarkierungen $m_f \in \Omega_N$ eine Transition aktiviert ist.

Als *Interface* des Netzes bezeichnen wir die Nachrichtenkanäle, die durch P_I und P_O repräsentiert sind. Die Bögen F können von Eingabepätzen nur ausgehend und bei Ausgabepätzen nur eingehend sein. Übertragen in die reale Welt entspricht dies beispielsweise beim klassischen Briefverkehr dem Umstand, dass ein verschickter Brief vom Absender nicht zurückgeholt werden kann. Der Postweg ist dabei für den Absender ein Ausgabekanal und für den Empfänger ein Eingabekanal. In den Abbildungen sind die Netze mit einem gestrichelten Rahmen versehen, welcher das Interface symbolisiert. Die Eingabepätze befinden sich stets links und die Ausgabepätze rechts auf diesem Rahmen.

In Abbildung 2.1 sind zwei offene Workflownetze eines fiktiven Getränkeautomaten dargestellt. Über Knöpfe am Automaten kann sich der Kunde beispielsweise zwischen Kaffee (Eingabe K) und Tee (Eingabe T) entscheiden. Im Startzustand wartet das dargestellte Netz auf eine dieser beiden Eingaben. Anschließend wirft der Kunde Geld ein (Eingabe €) und der Automat entscheidet selbständig, ob er nun das Getränk ausgibt (Ausgabe B_K bzw. B_T) oder mehr Geld verlangt (Ausgabe G). Nach der Ausgabe des Getränks endet die Interaktion. Der Kunde hat aber auch die Möglichkeit, während der iterativen Geldeingabe vorzeitig abubrechen (Eingabe A). In diesem Fall gibt der Automat die erhaltenen Münzen zurück (Ausgabe W) und die Interaktion endet ebenfalls.

Jede Markierung eines offenen Workflownetzes N fassen wir als Summe der Markierungen der Eingabe-, Ausgabe- und inneren Plätze auf ($m = m^I + m^O + m^N$). Die Kurzschreibweise m^N bezeichnet dabei die Markierung der inneren Plätze von N , entsprechend stellen m^I und m^O die Markierungen der Eingabe- bzw. Ausgabepätze dar.

Definition 2.8 (Inneres Netz)

Das innere Netz von N ist das Netz ohne die Interfaceplätze sowie ohne deren adjazente Bögen. Die Anfangs- und Endmarkierungen bleiben erhalten.

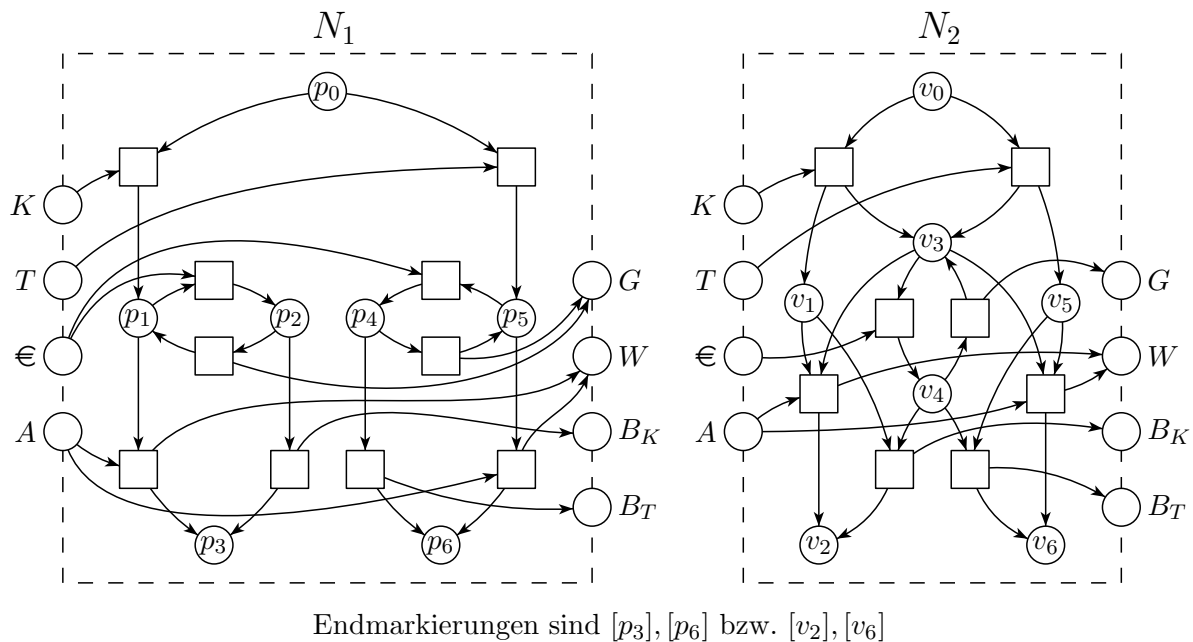


Abbildung 2.1: oWFNs zweier Getränkeautomaten

In dieser Arbeit betrachten wir nur solche offenen Workflownetze, deren inneres Netz beschränkt ist. Desweiteren schließen wir leere Transitionen im inneren Netz aus. Die Begründung für diese Einschränkung folgt im Abschnitt 4.2.

Definition 2.9 (Überdeckung)

Eine Markierung m eines oWFN überdeckt eine andere Markierung m' , falls für alle inneren Plätze $p \in P_N$ gilt: $m(p) \geq m'(p)$. Gibt es sogar einen Platz mit $m(p) > m'(p)$ so sprechen wir von echter Überdeckung.

Man beachte, dass es nur von den inneren Plätzen eines Netzes abhängt, ob eine Markierung eine andere überdeckt. In Abbildung 2.1 würde $[p_1, p_4]$ die Markierung $[p_1, \text{€}]$ echt überdecken. Auch $[p_1]$ überdeckt $[p_1, \text{€}]$, jedoch nicht echt.

2.1.3 Serviceautomaten

Nachdem wir als Modell für die Geschäftsprozesse offene Workflownetze betrachtet haben, wollen wir nun ein Modell für das Verhalten der Partner des Geschäftsprozesses anschauen. Im Gegensatz zu den offenen Workflownetzen, welche das *interne Verhalten* der Geschäftsprozesse widerspiegeln, interessieren wir uns bei den Partnern lediglich für die *Kommunikation nach außen*.

Als einfaches Modell, welches internes Verhalten vernachlässigt, bieten sich Serviceautomaten an. Eine Einführung in dieses Modell bieten beispielsweise [MS05]. Bei Serviceautomaten handelt es sich um Zustandsautomaten mit asynchronen Nachrichtenkanälen als Interface. In [LMW07] ist eine Möglichkeit dargestellt, einen entsprechenden Serviceautomaten aus einem oWFN zu konstruieren. Dabei kann der erzeugte Automat als das kommunikative Verhalten des ursprünglichen Netzes betrachtet werden.

Definition 2.10 (Serviceautomat)

Ein Serviceautomat $R = [Q, C_I, C_O, \delta, q_0, \Omega_R]$ mit Nachrichtenkanälen C ist ein endlicher Automat mit

- disjunkten Eingabekanälen C_I und Ausgabekanälen C_O , mit $C_I \cup C_O = C$,
- einer endlichen Zustandsmenge Q ,
- einer Zustandsüberföhrungsfunktion $\delta \subseteq Q \times (C_I \cup C_O \cup \{\tau\}) \times Q$ und
- Endzuständen $\Omega \subseteq Q$.

Von den Endzuständen aus werden keine Nachrichten versendet, d.h. wenn $q \in \Omega$ und $(q, x, q') \in \delta$, dann ist $x \in C_I$.

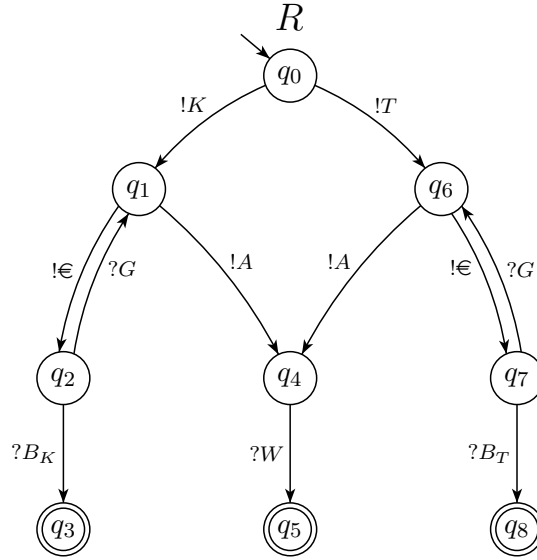
Wir verwenden die klassische Darstellung endlicher Zustandsautomaten, d.h. Zustände sind Kreise, Überföhrungen Pfeile, Anfangs- und Endzustände sind mit eingehendem Pfeil bzw. doppeltem Rand markiert. Zur grafischen Darstellung der Kommunikation des Serviceautomaten wird an den Kanten bei *eingehenden Nachrichten* $x \in C_I$ ein Fragezeichen vorangestellt, während *ausgehende Nachrichten* $x \in C_O$ mit einem Ausrufezeichen gekennzeichnet sind.

Der Serviceautomat in Abbildung 2.2 repräsentiert einen zu dem oben vorgestellten Getränkeautomaten passenden Kunden. Dieser Kunde wählt entweder Kaffee (K) oder Tee (T) und wirft solange Münzen ein (€) bis das Getränk bezahlt ist. Anschließend nimmt er das Getränk entgegen (B_K). Wie man sieht, kann er aber auch den Bezahlvorgang abbrechen und sein Wechselgeld annehmen (W).

Definition 2.11 (Partnerautomat)

Seien $N = [P, T, F, W, m_0, \Omega_N]$ ein oWFN und $R = [Q, C_I, C_O, \delta, q_0, \Omega_R]$ ein Serviceautomat. Wenn $P_I = C_O$ und $P_O = C_I$, dann heißt R Partnerautomat für N .

Dass die Interfaces zusammenpassen, garantiert noch nicht eine sinnvolle Interaktion eines Partners mit einem bestimmten Geschäftsprozess. Man kann sich leicht einen Kunden vorstellen, der zwar die Bedienelemente des Getränkeautomaten kennt, diese jedoch

Abbildung 2.2: Partnerautomat für das Netz N_1 aus Abb. 2.1

in falscher Reihenfolge benutzt. Daher führen wir ein Kriterium ein, das zum Ausdruck bringt, ob die Interaktion eines Partnerautomat mit einem bestimmten Geschäftsprozess sinnvoll verläuft oder nicht.

Definition 2.12 (Komposition)

Sei $N = [P, T, F, W, m_0, \Omega_N]$ ein oWFN und $R = [Q, C_I, C_O, \delta_R, q_0, \Omega_R]$ ein Partnerautomat für N . Dann heißt der Automat $N \oplus R = [Q_{N \oplus R}, \delta_{N \oplus R}, q_{0_{N \oplus R}}, \Omega_{N \oplus R}]$ mit

- Zuständen $Q_{N \oplus R}$ (Paare aus einer Markierung von N und einem Zustand von R),
- einer Zustandüberföhrungsfunktion $\delta_{N \oplus R} \subseteq Q_{N \oplus R} \times Q_{N \oplus R}$,
- einem Startzustand $q_{0_{N \oplus R}} = (m_0, q_0)$ und
- Endzuständen $\Omega_{N \oplus R} = \{(m_f, q_f) \mid m_f \in \Omega_N \text{ und } q_f \in \Omega_R\}$

die Komposition aus N und R . Die Zustände und Überföhrungsfunktion sind induktiv definiert: $q_{0_{N \oplus R}} \in Q_{N \oplus R}$. Für $(m, q) \in Q_{N \oplus R}$ existiert – in den folgenden Fällen – auch $(m', q') \in Q_{N \oplus R}$ und $(m, q) \rightarrow (m', q') \in \delta_{N \oplus R}$:

- $q = q'$ und es gibt eine Transition $t \in T$ mit $m \xrightarrow{t} m'$,
- es gibt einen Eingabepplatz $p_I \in P_I$ mit $(q, p_I, q') \in \delta_R$ und $m' = m + [p_I]$,
- es gibt einen Ausgabepplatz $p_O \in P_O$ mit $(q, p_O, q') \in \delta_R$ und $m' = m - [p_O]$,
- $(q, \tau, q') \in \delta_R$ und $m' = m$.

Wir nennen Zustände $q \in Q_{N \oplus R}$ *erreichbar*, wenn es eine Zustandsfolge q_0, q_1, \dots, q_n gibt mit $q_0 = q_{0_{N \oplus R}}$ und $q_n = q$, wobei für je zwei aufeinanderfolgende Zustände eine Kante in der Komposition existiert, $(q_i, q_{i+1}) \in \delta_{N \oplus R}$.

Definition 2.13 (Deadlockfrei)

Die Komposition $N \oplus R$ aus dem oWFN $N = [P, T, F, W, m_0, \Omega_N]$ und dem Serviceautomaten $R = [Q, C_I, C_O, \delta, q_0, \Omega_R]$ heißt *deadlockfrei*, falls jeder erreichbare Zustand der Komposition, der kein Endzustand ist, mindestens einen Nachfolger besitzt.

Definition 2.14 (Beschränkte Kommunikation)

Ein oWFN und ein Serviceautomat kommunizieren k -beschränkt, falls in der Komposition auf keinem Nachrichtenkanal jemals mehr als k Nachrichten liegen. Ist der konkrete Wert dieser Schranke uninteressant, so spricht man auch einfach von beschränkter Kommunikation.

Definition 2.15 (Strategie)

Ein Partnerautomat R heißt k -Strategie für ein oWFN N , wenn die Komposition $N \oplus R$ deadlockfrei ist sowie k -beschränkt kommuniziert. Die Menge aller k -Strategien für N sei $\text{Strat}_k(N)$. Ihre Vereinigung $\text{Strat}(N) := \bigcup_{k \in \mathbb{N}} \text{Strat}_k(N)$ enthält dann alle Strategien für N .

Man sieht leicht, dass die Komposition aus Abbildung 2.3 deadlockfrei ist. Die beiden Partner N_1 und R kommunizieren zudem 1-beschränkt. Somit ist R eine 1-Strategie für das Netz N_1 .

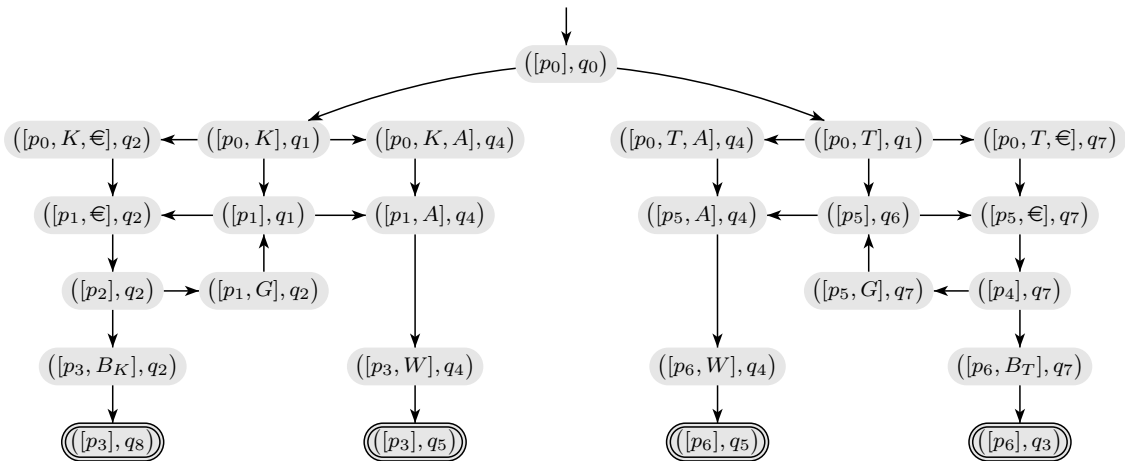


Abbildung 2.3: Komposition von N_1 aus Abb. 2.1 und R aus Abb. 2.2

In dieser Arbeit werden nur Netze und Serviceautomaten betrachtet, die miteinander beschränkt kommunizieren.

2.2 Bedienungsanleitung

Eine kompakte Darstellung aller Strategien zu einem gegebenen Netz N bietet die sogenannte *Bedienungsanleitung für N* . In [MS05] wurde dieses Konzept erstmals eingeführt und in [LMW07] erweitert. In [SMB08] wurde zusätzlich eine \sqsubseteq -Relation für Bedienungsanleitungen eingeführt. Die nachfolgenden Definitionen sind angelehnt an [SMB08].

Ein Automat heißt *deterministisch*, wenn es keine τ -Transitionen gibt und für jeden Zustand und jedes Label höchstens eine ausgehende Kante mit diesem Label existiert.

Definition 2.16 (Annotierter Automat)

Ein annotierter Automat $R^\phi = [Q, C_I, C_O, \delta, q_0, \Omega, \phi]$ besteht aus einem deterministischen Automaten $R = [Q, C_I, C_O, \delta, q_0, \Omega_R]$ und einer Annotationsfunktion $\phi : Q \rightarrow \mathcal{BF}$. Dabei ist \mathcal{BF} die Menge der booleschen Funktionen über $C_I \cup C_O \cup \{\tau, final\}$.

Definition 2.17 (Assignment)

Ein Assignment eines Serviceautomaten R weist wie folgt jedem Zustand q von R eine boolesche Belegung $\beta_R(q)$ mit den Werten *true/false* für die Kantenbeschriftungen der von q ausgehenden Kanten zu:

$$\beta_R(q)(x) := \begin{cases} true, & x \neq final \text{ und } \exists q' : (q, x, q') \in \delta \\ true, & x = final \text{ und } q \in \Omega \\ false, & \text{sonst} \end{cases}$$

Definition 2.18 (Simulationsrelation)

Seien $S = [Q_S, C_I, C_O, \delta_S, q_{0_S}, \Omega_S]$ und $R = [Q_R, C_I, C_O, \delta_R, q_{0_R}, \Omega_R]$ zwei Serviceautomaten. Eine Simulationsrelation ϱ zwischen S und R ist wie folgt induktiv definiert:

- $(q_{0_S}, q_{0_R}) \in \varrho$.
- Wenn $(q_S, q_R) \in \varrho$ und $x \in C_I \cup C_O$ sowie $(q_S, x, q'_S) \in \delta_S$, dann ist $(q_R, x, q'_R) \in \delta_R$ und $(q'_S, q'_R) \in \varrho$.
- Wenn $(q_S, q_R) \in \varrho$ und $(q_S, \tau, q'_S) \in \delta_S$, dann ist $(q'_S, q_R) \in \varrho$.

Die Beziehung von einem Serviceautomaten zu einem annotierten Automaten wird über ein *Matching* hergestellt.

Definition 2.19 (Matching)

Sei $S = [Q_S, C_I, C_O, \delta_S, q_{0_S}, \Omega_S]$ ein Serviceautomat, $R^\phi = [Q_R, C_I, C_O, \delta_R, q_{0_R}, \Omega_R, \phi]$ ein annotierter Automat und ϱ eine Simulationsrelation zwischen S und R . Dann matcht S mit R^ϕ , falls für alle $(q_S, q_R) \in \varrho$ gilt: $\beta_S(q_S) \models \phi(q_R)$. Eine Belegung β erfüllt dabei die Annotation ϕ (kurz $\beta \models \phi$), wenn die durch ϕ gegebene Formel mit der Belegung β eine wahre Aussage darstellt.

$Match(R^\phi)$ ist die Menge aller Automaten, die mit R^ϕ matchen.

Definition 2.20 (Bedienungsanleitung)

Sei N ein oWFN und $OG = R^\phi$ ein annotierter Automat. OG heißt genau dann k -Bedienungsanleitung (engl. *Operating Guideline*) zu N , wenn $Match(OG) = Strat_k(N)$. Wir schreiben in dem Fall auch $OG = OG_N$.

Bedienungsanleitungen bieten die Möglichkeit, alle Strategien eines Netzes kompakt darzustellen. Die Berechnung einer solchen Bedienungsanleitung ist in [LMW07] beschrieben. Zur Anschauung zeigt die Abbildung 2.4 eine 1-Bedienungsanleitung für das Netz N_1 aus Abbildung 2.1.

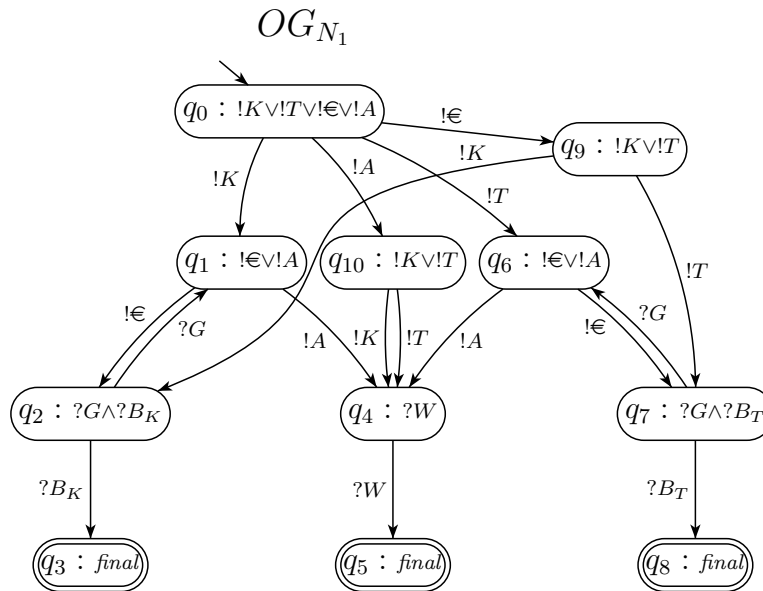


Abbildung 2.4: Bedienungsanleitung für das Netz N_1 aus Abb. 2.1

Wie eingangs gefordert, wollen wir Laufzeitersetzungen auch für Teilmengen aller Kunden angeben können. Dies bedeutet, wir benötigen eine Möglichkeit, Mengen von Strategien, die als annotierte Automaten gegeben sind, zueinander in Beziehung zu setzen.

Definition 2.21 (\sqsubseteq -Relation für annotierte Automaten)

Seien $S^\psi = [Q_S, C_I, C_O, \delta_S, q_{0_S}, \Omega_S, \psi]$ und $R^\phi = [Q_R, C_I, C_O, \delta_R, q_{0_R}, \Omega_R, \phi]$ zwei annotierte Automaten. Sei ϱ eine Simulationsrelation zwischen S und R . Dann gelte die Beziehung $S^\psi \sqsubseteq R^\phi$, falls für alle $(q_S, q_R) \in \varrho$ gilt: $\phi(q_R) \implies \psi(q_S)$ ist eine Tautologie.

Man kann sich die \sqsubseteq -Relation annotierter Automaten vorstellen als eine Art Mengenvergleich für die repräsentierten Strategien. Werden nämlich alle Strategien, die S^ψ repräsentiert, auch von R^ϕ repräsentiert, so ist $S^\psi \sqsubseteq R^\phi$.

In Abbildung 2.5 sieht man zwei annotierte Automaten OG' (a) und OG (b), welche bestimmte Strategien von Kunden beschreiben. Die Kunden gemäß OG wählen zuerst ihr Getränk (Tee T oder Kaffee K). Anschließend bezahlen sie (€) solange der Automat weiteres Geld verlangt (G), oder sie brechen den Vorgang ab und lassen sich ihr eingeworfenes Geld zurückgeben. Die Kunden gemäß OG' wählen hingegen als Getränk immer nur Kaffee. Ansonsten verhalten sie sich wie die Kunden mit der freien Getränkewahl. Es gilt die Beziehung $OG' \sqsubseteq OG \sqsubseteq OG_{N_1}$. Wie wir sehen werden, gibt es umso mehr Möglichkeiten der Laufzeitersetzung von einem Ausgangsnetz in ein ZN, je weniger Strategien dabei bewahrt werden sollen.

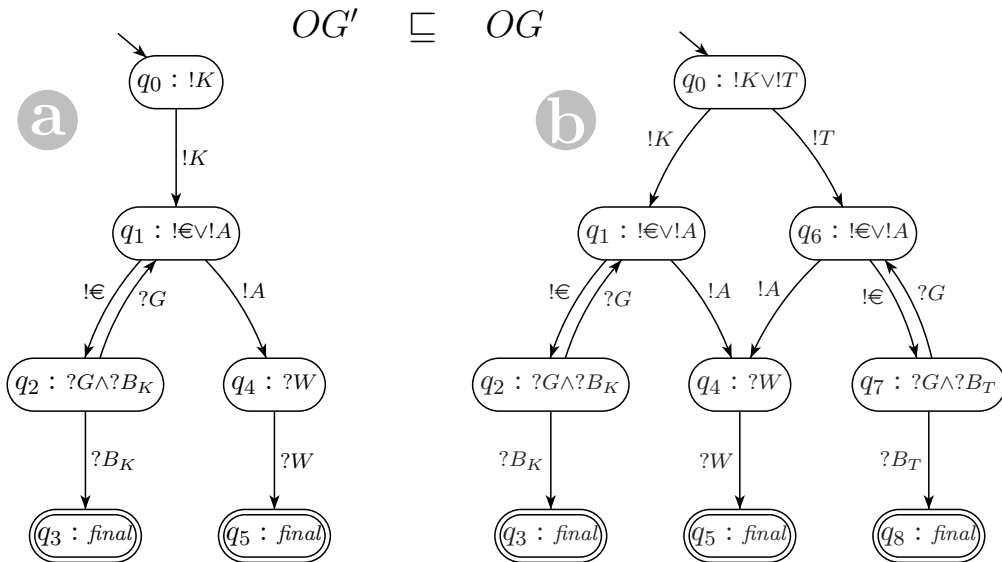


Abbildung 2.5: Teilmenge von Strategien (\sqsubseteq -Relation)

3 neue Definitionen

3.1 Wissensfunktionen

Zur Berechnung der Laufzeitersetzung führen wir zwei Hilfsfunktionen, die Erreichbarkeitsfunktion und die Fortführbarkeitsfunktion, ein. Diese enthalten Informationen über erreichbare bzw. fortführbare Markierungen. Betrachten wir einen Partnerautomaten R , so wollen wir wissen, welche Markierungen eines Netzes N bei der Interaktion mit R erreicht werden können und von welchen Markierungen von N die Interaktion problemlos fortgesetzt werden kann.

3.1.1 Erreichbarkeitsfunktion

Die Erreichbarkeitsfunktion ist angelehnt an das *Knowledge*, welches in [LMW07] bei der Berechnung einer Bedienungsanleitung verwendet wird. Wir modifizieren die Definition dahingehend, dass nicht zwei Serviceautomaten miteinander kommunizieren sondern ein Serviceautomat mit einem offenen Workflownetz.

Definition 3.1 (Erreichbarkeitsfunktion)

Seien $N = [P, T, F, W, m_0, \Omega_N]$ ein oWFN, $R = [Q, C_I, C_O, \delta, q_0, \Omega_R]$ ein Partnerautomat für N und q ein Zustand von R . Sei weiterhin $Q_{N \oplus R}$ die Menge der erreichbaren Zustände der Komposition $N \oplus R$. Dann ist die Erreichbarkeitsfunktion K_1 definiert über den Zuständen des Automaten als $K_1(q) := \{m \mid (m, q) \in Q_{N \oplus R}\}$.

Die Erreichbarkeitsfunktion gibt für jeden Zustand q des Serviceautomaten an, in welchen Markierungen sich das oWFN befinden kann, wenn bei korrekter Kommunikation der beteiligten Partner der Automat in diesen Zustand gelangt ist. Die Bestimmung der Erreichbarkeitsfunktion ist trivial, auch für nichtdeterministische Automaten: Man kann die Informationen direkt aus der Komposition des Netzes mit dem Serviceautomaten ablesen. Eine effiziente Implementation ist in dem Software-Tool Fiona gegeben.

3 neue Definitionen

Im folgenden betrachten wir den Algorithmus $\text{ERREICHBARKEIT}(N_1, R)$ als gegeben. Dieser erhält als Eingabe ein oWFN N_1 sowie einen Serviceautomaten R und liefert als Rückgabe die Erreichbarkeitsfunktion.

Ein Beispiel für eine Erreichbarkeitsfunktion zeigt Abbildung 3.1. Der beteiligte Automat R ist hier und den folgenden Beispielen jener aus Abbildung 2.5(b). Der Übersicht halber schreiben wir die Funktionswerte in Tabellenform an die Zustände des Automaten. Man überzeugt sich leicht von der Korrektheit im Vergleich mit der Komposition aus Abbildung 2.3.

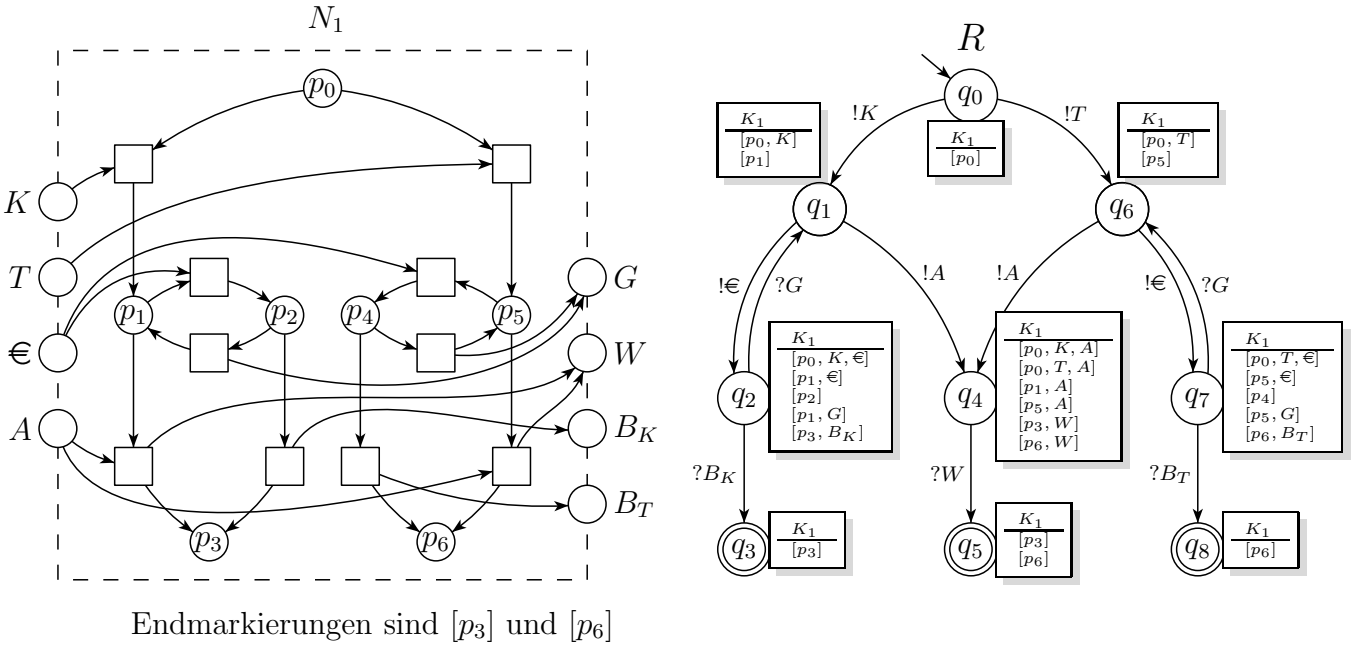


Abbildung 3.1: oWFN mit Erreichbarkeitsfunktion

3.1.2 Fortführbarkeitsfunktion

Die Fortführbarkeitsfunktion stellt die Umkehrung der Erreichbarkeitsfunktion dar. Dabei werden nicht erreichbare sondern *fortführbare* Markierungen betrachtet. Uns interessiert für jeden Zustand des Partnerautomaten, in welchen Markierungen sich das Workflownetz befinden kann, damit die beiden ab diesem Zustand deadlockfrei interagieren.

Definition 3.2 (fortführbar in q)

Seien $N = [P, T, F, W, m_0, \Omega_N]$ ein oWFN und $R = [Q, C_I, C_O, \delta, q_0, \Omega_R]$ ein Partnerautomat für N . Eine Markierung m von N heißt *fortführbar im Zustand* $q \in Q$, wenn die Komposition $N \oplus R$ mit Startzustand (m, q) deadlockfrei ist und die beiden Partner beschränkt kommunizieren.

Definition 3.3 (Fortführbarkeitsfunktion)

Seien $N = [P, T, F, W, m_0, \Omega_N]$ ein oWFN, $R = [Q, C_I, C_O, \delta, q_0, \Omega_R]$ ein Partnerautomat für N und q ein Zustand von R . Dann ist die Fortführbarkeitsfunktion K_2 definiert über den Zuständen des Automaten als $K_2(q) := \{m \mid m \text{ ist fortführbar in } q\}$.

Für die Berechnung der Fortführbarkeitsfunktion nutzen wir folgenden Ansatz: Wir lassen sowohl den Automaten R als auch das Netz N ab den Endzuständen rückwärts laufen und bestimmen dabei die *rückwärts erreichbaren Markierungen*, analog zur vorher betrachteten Erreichbarkeitsfunktion. Dafür benötigen wir Umkehrungen von Serviceautomaten und offenen Workflownetzen.

Definition 3.4 (Umkehrung)

Bei der Umkehrung $\text{reverse}(N)$ eines oWFN N werden die Bögen umgedreht und Ein- und Ausgabekanäle vertauscht. Endmarkierung ist die alte Startmarkierung, während ein neu hinzugefügter Platz v'_0 anfangsmarkiert ist und Transitionen zu allen bisherigen Endmarkierungen erhält.

Bei der Umkehrung $\text{reverse}(R)$ eines Automaten R werden alle Kanten umgedreht und die Ein- und Ausgabekanäle vertauscht. Endzustand ist der alte Startzustand, während der neu hinzugefügte Startzustand q'_0 τ -Kanten zu allen bisherigen Endzuständen erhält.

Es folgt der Algorithmus $\text{FORTFUEHRBARKEIT}(N_2, R)$. Als Eingabe erhält der Algorithmus ein oWFN sowie einen Serviceautomaten und berechnet die Fortführbarkeitsfunktion. Der Berechnung liegt folgende Idee zugrunde: Zunächst werden die rückwärts erreichbaren Markierungen bestimmt und anschließend diejenigen wieder gelöscht, die zu einem Deadlock in der Komposition führen.

```

FORTFUEHRBARKEIT( $N_2, R$ )
(1)   $K_2 := \text{ERREICHBARKEIT}(\text{reverse}(N_2), \text{reverse}(R))$ 
(2)  do
(3)    foreach  $q \in Q$ 
(4)      while  $\exists m \in K_2(q)$  mit
(4a)         $(\exists q' : (q, x, q') \in \delta \text{ mit } K_2(q') \not\neq \begin{cases} m + [x], & x \in C_I \\ m - [x], & x \in C_O \\ m, & x = \tau \end{cases}$ 
(4b)        oder  $R_{N_2}(m) \not\subseteq K_2(q)$ ) do
(5)           $K_2(q) := K_2(q) \setminus \{m\}$ 
(6)  while  $K_2$  ändert sich
(7)  return  $K_2$ 

```

Algorithmus 3.2: Berechnung der Fortführbarkeitsfunktion

Wir verdeutlichen die Arbeitsweise des Algorithmus anhand des Beispiels von Abbildung 3.4. Die Umkehrungen des gegebenen Netzes N_2 und des Automaten R sind in Abbildung 3.3 dargestellt. Desweiteren lässt sich an dem umgekehrten Automaten die Erreichbarkeitsfunktion ablesen. Anhand der Zwischenergebnisse des Algorithmus erläutern wir die Berechnung von $K_2(q_1)$.

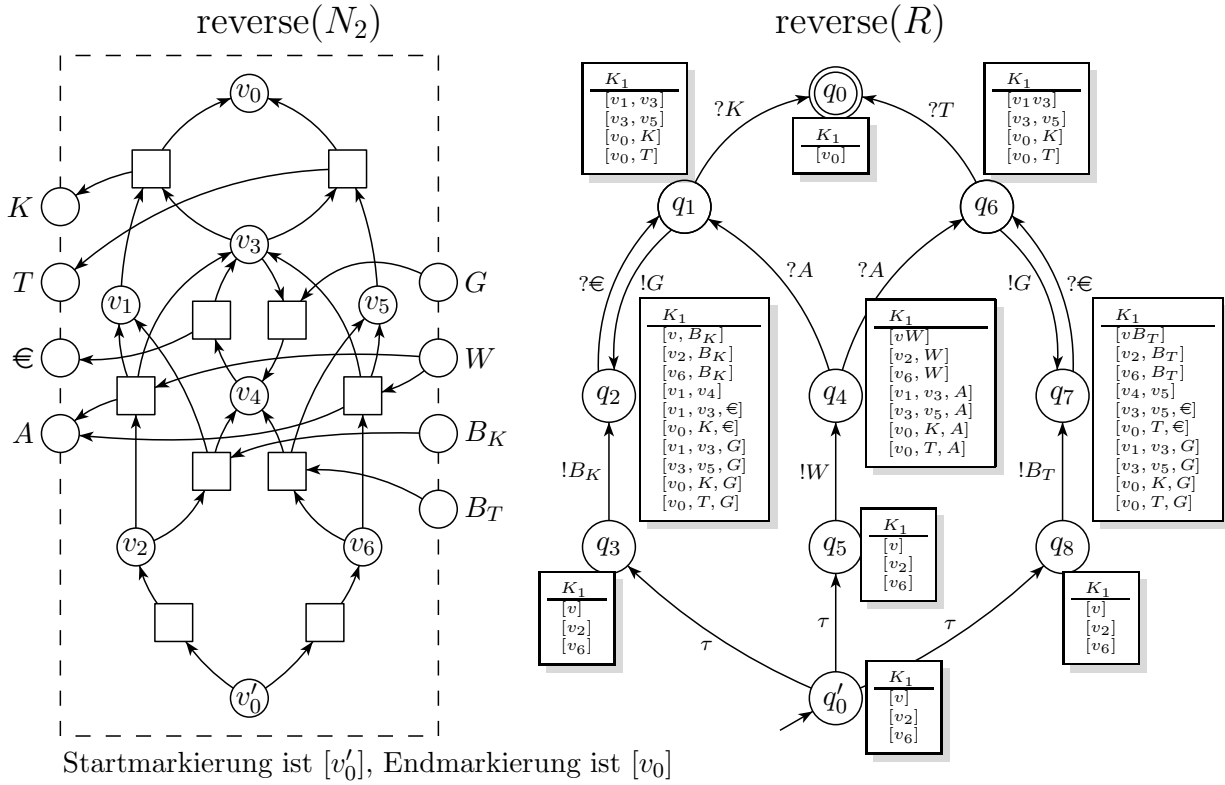
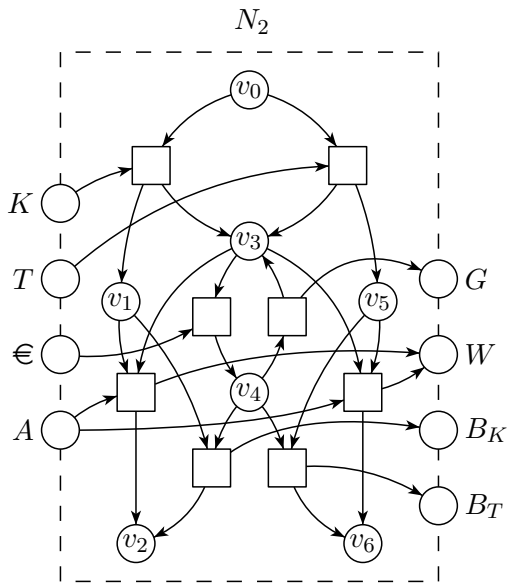


Abbildung 3.3: umgekehrtes Netz und umgekehrter Automat

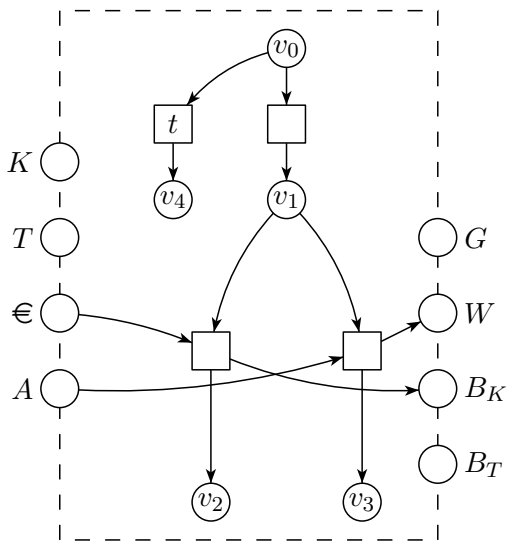
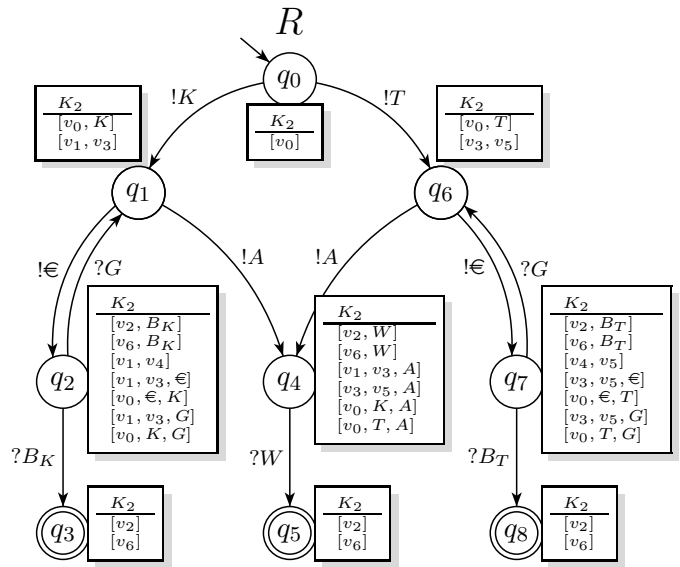
Gemäß Zeile 1 des Algorithmus enthält $K_2(q_1)$ zunächst $\{[v_1, v_3], [v_3, v_5], [v_0, K], [v_0, T]\}$ (vgl. Abb. 3.3). In den Zeilen 2 ff. werden diejenigen Markierungen gelöscht, die in einem Zustand von R nicht fortführbar sind. Aus $K_2(q_1)$ wird $[v_0, T]$ durch die Bedingung in Zeile 4a gelöscht. Schließlich enthält $K_2(q_2)$ nicht die Markierung $[v_0 \in T]$ und ist somit nicht fortführbar. Die Markierung $[v_3 v_5]$ wird aus dem gleichen Grund aus $K_2(q_1)$ entfernt. Das Ergebnis des Algorithmus ist in Abbildung 3.4 zu sehen.

Ein weiteres Beispiel verdeutlicht die Notwendigkeit der Bedingung in Zeile 4b. In Abbildung 3.5 sieht man ein Workflownetz, für das der Automat R keine Strategie ist. Das Netz besitzt in der Tat überhaupt keine Strategien. Der Zustand q_1 des Automaten wird zunächst mit den Markierungen $[v_0]$ und $[v_1]$ versehen (Zeile 1 des Algorithmus). Allerdings kann von der Markierung $[v_0]$ im Netz die Transition t schalten zu $[v_4]$, womit sich das Netz in einem Deadlock befindet. Somit wird in Zeile 4b diese Markierung wieder entfernt. Das Ergebnis der Berechnung ist in Abbildung 3.5 gegeben.



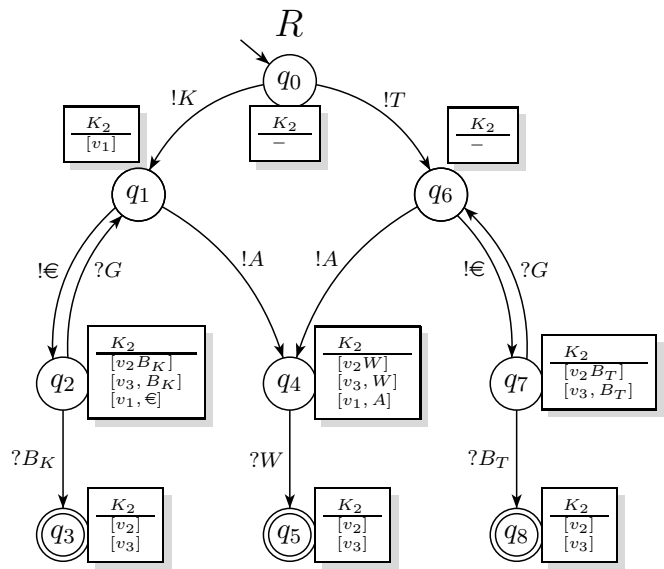
Endmarkierungen sind $[v_2]$ und $[v_6]$

Abbildung 3.4: oWFN mit Fortführbarkeitsfunktion



Endmarkierungen sind $[v_2]$ und $[v_3]$

Abbildung 3.5: Fortführbarkeitsfunktion für nicht bedienbares oWFN



3 neue Definitionen

Trotz der eingangs geforderten k -Beschränktheit der beteiligten Workflownetze kann es passieren, dass das umgekehrte Netz unbeschränkt ist. Abbildung 3.6 zeigt ein solches Beispiel. Bei der Komposition des umgekehrten Netzes mit dem umgekehrten Automaten ist der Platz v_4 wegen der Transition t unbeschränkt.

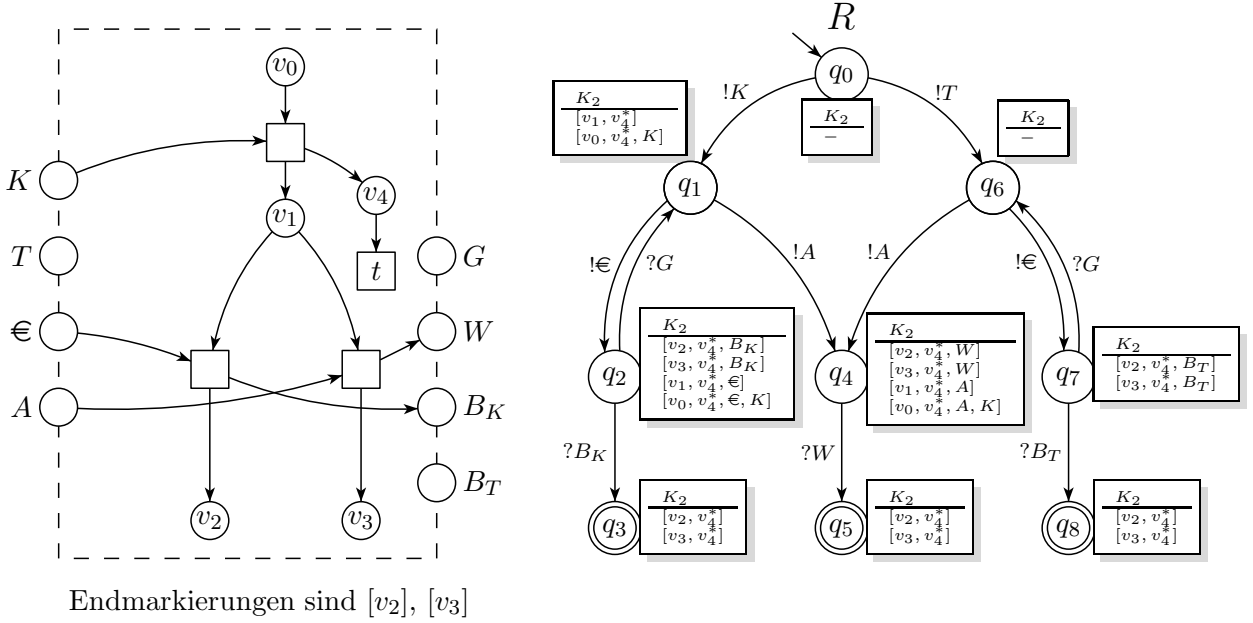


Abbildung 3.6: oWFN mit Fortführbarkeitsfunktion bei unbeschränkter Umkehrung

In diesem Fall erzeugen wir die Komposition aus umgekehrtem Netz mit umgekehrtem Automaten nicht vollständig, sondern beschränken uns auf solche Zustände, bei denen die Plätze mit höchstens k Marken belegt sind.

Mit dem folgenden Lemma beweisen wir die Korrektheit des vorgestellten Algorithmus und schaffen damit die Grundlage für den folgenden Abschnitt *Laufzeitersetzung*.

Lemma 3.5 *Der Algorithmus FORTFUEHRBARKEIT(N_2, R) berechnet die Fortführbarkeitsfunktion K_2 für das oWFN N_2 und den Serviceautomaten R korrekt.*

Beweis:

Offensichtlich umfassen die rückwärts erreichbaren Markierungen (Zeile 1 des Algorithmus) mindestens alle deadlockfreien Markierungen.

Wir führen nun die Annahme zum Widerspruch, für einen Zustand q entspräche die vom Algorithmus berechnete Menge $K_2(q)$ nicht der Fortführbarkeitsfunktion. Dann gäbe es eine Markierung $m \in K_2(q)$, die nicht fortführbar in q ist. Mindestens einer der folgenden drei Punkte träfe dann zu:

- (i) Der Zustand (q, m) in der Komposition $R \oplus N_2$ hat keinen Nachfolger und ist nicht Endzustand.
- (ii) R kann in einen Zustand q' wechseln, aber die entsprechend modifizierte Markierung des Netzes ist nicht fortführbar in q' .
- (iii) N_2 kann schalten in eine Markierung m' , aber m' ist nicht fortführbar in q .

Punkt (i) ist unmöglich, da die Komposition als umgekehrte Komposition von den Endzuständen aus erstellt wurde und es somit Nachfolger von (q, m) geben muss. Punkt (ii) wird durch die Überprüfung in Zeile 4a des Algorithmus ausgeschlossen. Zeile 4b garantiert, dass Punkt (iii) nicht auftritt.

Die in den Punkten (i)–(iii) beschriebenen Zustände hätten in der Komposition stets einen Deadlock zur Folge. Umgekehrt würde, wenn m in q nicht fortführbar wäre, einer der drei Punkte auf (q, m) zutreffen. Daher werden durch den Algorithmus genau alle nicht fortführbaren Markierungen von den rückwärts erreichbaren Markierungen entfernt. Übrig bleiben genau die fortführbaren Markierungen. \square

3.2 Laufzeitersetzung

Im Folgenden seien als zugrundeliegende Netze stets zwei offene Workflownetze $N_1 = [P_1 \cup P_I \cup P_O, T_1, F_1, W_1, m_{0_1}, \Omega_1]$ und $N_2 = [P_2 \cup P_I \cup P_O, T_2, F_2, W_2, m_{0_2}, \Omega_2]$ mit demselben Interface gegeben. Die Idee der Laufzeitersetzung ist, mithilfe zusätzlicher Transitionen Marken vom Ausgangs- in das Zielnetz zu verschieben. Die Interaktion beginnt dann mit der Startmarkierung von N_1 und endet bei einer Endmarkierung von N_2 .

Definition 3.6 (Transitionsmenge zweier Netze)

Als Transitionsmenge von N_1 nach N_2 bezeichnen wir ein Tupel $\mathcal{T} = (T_{\mathcal{T}}, F_{\mathcal{T}}, W_{\mathcal{T}})$ aus Transitionen, Bögen und Bogenvielfachheiten, wenn für alle $t \in T_{\mathcal{T}}$ gilt: $\bullet t \in \text{bags}(P_1)$ und $t\bullet \in \text{bags}(P_2 \cup P_I)$.

Die Transitionen in einer Transitionsmenge enthalten im Vorbereich nur innere Plätze des Ausgangsnetzes. Im Nachbereich sind innere Plätze des Zielnetzes oder auch Eingabepätze erlaubt. Es liegt nahe, die beteiligten Netze mit Hilfer einer solchen Transitionsmenge zu einem größeren offenen Workflownetz zu verbinden.

Definition 3.7 (verbundenes Netz)

Sei $\mathcal{T} = (T_{\mathcal{T}}, F_{\mathcal{T}}, W_{\mathcal{T}})$ eine Transitionsmenge von N_1 nach N_2 . Das oWFN

$$N_1 \oplus \mathcal{T} \oplus N_2 := (P_1 \cup P_2 \cup P_I \cup P_O, T_1 \cup T_{\mathcal{T}} \cup T_2, F_1 \cup F_{\mathcal{T}} \cup F_2, W_1 \cup W_{\mathcal{T}} \cup W_2, m_{0_1}, \Omega_2)$$

heißt das verbundene Netz aus N_1 , N_2 und \mathcal{T} .

N_1 heißt in diesem Zusammenhang Ausgangsnetz, N_2 heißt Zielnetz.

Das Interface des verbundenen Netzes ist hier die Verschmelzung der (identischen) Interfaces beider Workflownetze N_1 und N_2 . Dadurch können die Transitionen der beiden ursprünglichen Netze gemeinsam die Ein- und Ausgabepplätze verwenden. Wie wir sehen werden, ist dieses Verhalten bei der Laufzeitersetzung explizit erwünscht.

Man beachte, dass die Anfangsmarkierung des verbundenen Netztes genau die Anfangsmarkierung des Ausgangsnetzes ist, während die Endmarkierungen des verbundenen Netzes denen des Zielnetzes entsprechen. Die Marken auf den Plätzen sollen dabei zur Laufzeit von N_1 nach N_2 verschoben werden.

Definition 3.8 (Laufzeitersetzung)

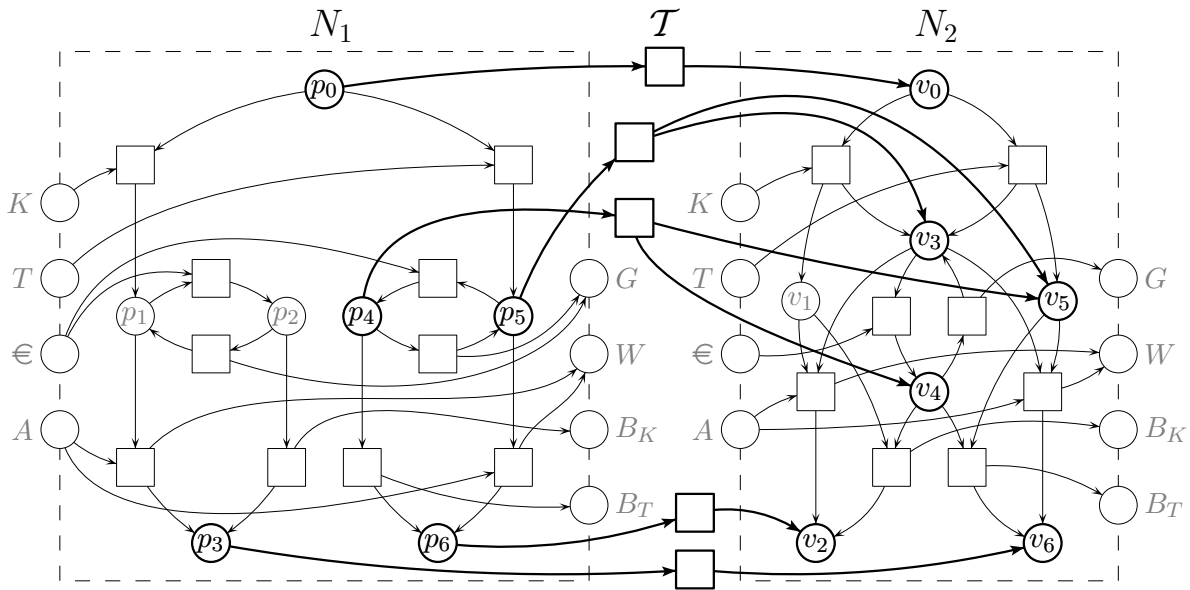
Gegeben sei eine Bedienungsanleitung OG mit $OG \sqsubseteq OG_{N_1}$. Dann heißt eine Transitionsmenge \mathcal{T} von N_1 nach N_2 Laufzeitersetzung gemäß OG , falls gilt:

$$\begin{aligned} \text{Jeder Serviceautomat } R \in \text{Match}(OG) \text{ ist eine Strategie für } N_1 \oplus \mathcal{T} \oplus N_2, \\ \text{also } \text{Match}(OG) \subseteq \text{Strat}(N_1 \oplus \mathcal{T} \oplus N_2). \end{aligned}$$

Die Transitionen $t \in T_{\mathcal{T}}$ nennen wir *Laufzeitersetzungstransitionen* und schreiben kurz $\bullet t \rightarrow \square \rightarrow t \bullet$. Zur weiteren Vereinfachung der Schreibweise bezeichne im folgenden eine Laufzeitersetzungstransition t nicht nur eine Transition des verbundenen Workflownetzes sondern zugleich die mit t verbundenen Bögen und Bogenvielfachheiten. Die Aussage $t \in \mathcal{T}$ bedeutet dann, dass die Transition t zusammen mit den entsprechenden Bögen in der Laufzeitersetzung enthalten ist.

Abbildung 3.7 zeigt eine Laufzeitersetzung für die Netze zweier Getränkeautomaten. Die beiden Netze teilen zwar die Interfaceplätze, zur besseren Übersichtlichkeit sind diese jedoch doppelt – einmal für jedes Netz – dargestellt. Die gestrichelten Rahmen markieren optisch die ursprünglichen Workflownetze. Als Konvention liegen Eingabepplätze stets links auf dem Rahmen, Ausgabepplätze rechts.

In dem konkreten Beispiel besitzen die beiden Netze die gleiche Bedienungsanleitung. Daher ist auch die Laufzeitersetzungstransition $[p_0] \rightarrow \square \rightarrow [v_0]$ enthalten – die Netze könnten also gleich zu Beginn der Interaktion ersetzt werden.



Startmarkierung des verbundenen Netzes ist $[p_0]$, Endmarkierungen sind $[v_2]$ und $[v_6]$

Abbildung 3.7: Laufzeitersetzung offener Workflownetze gemäß *OG* aus Abb. 2.5

In unserem Beispiel sind auch Laufzeitersetzungstransitionen enthalten, die eine Endmarkierung des Ausgangsnetzes in eine Endmarkierung des Zielnetzes überführt. Solche Transitionen können immer als Laufzeitersetzungstransitionen verwendet werden.

Definition 3.9 (triviale Laufzeitersetzung)

Die Transitionsmenge $\{m_f \rightarrow \square \rightarrow m'_f \mid m_f \in \Omega_1 \text{ und } m'_f \in \Omega_2\}$, welche ausschließlich Überführungen von Endmarkierungen enthält, heißt triviale Laufzeitersetzung.

Die Bezeichnung der trivialen Laufzeitersetzung als *Laufzeitersetzung* impliziert, dass die Bedingungen nach Definition 3.8 erfüllt sind. Tatsächlich können wir dies an dieser Stelle noch nicht beweisen. Erst im Abschnitt 4.2 gehen wir erneut auf die triviale Laufzeitersetzung ein und rechtfertigen somit im Nachhinein die Namensgebung.

3.2.1 Eingabeplätze neu belegen

Wie aus der Definition der Transitionsmenge ersichtlich, dürfen die Laufzeitersetzungstransitionen auch Marken auf die Eingabeplätze P_I legen. Im übertragenen Sinne hieße solch ein Verhalten, dass die Laufzeitersetzung dem Zielservice gegenüber bestimmte Nachrichten des Partners simuliert. Der Zielservice würde die Nachrichten intern verarbeiten, als wären sie vom Partner (bzw. Kunden) persönlich übermittelt worden.

3 neue Definitionen

Abbildung 3.8 verdeutlicht, was passieren kann, wenn man Eingabepunkte im Nachbarbereich von Laufzeitersetzungstransitionen verbietet. Man sieht leicht, dass es deutlich weniger Laufzeitersetzungstransitionen gibt, wenn der Eingabepunkt ϵ nicht belegt werden darf.

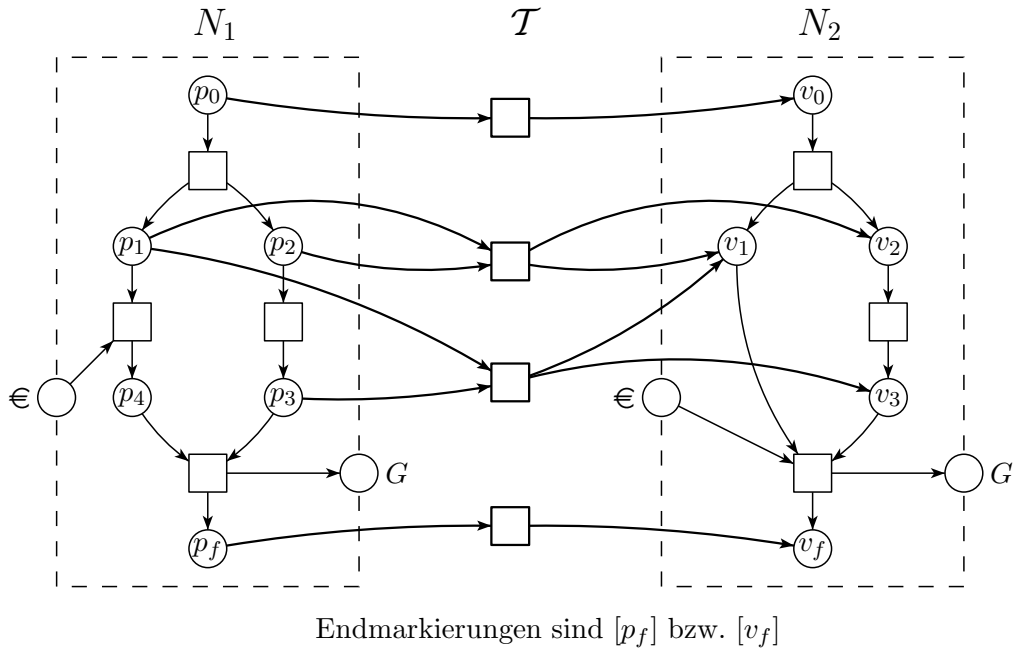


Abbildung 3.8: Laufzeitersetzung, die keinen Eingabepunkt belegt

Bei diesem Netz können die Markierungen $[p_2, p_4]$ und $[p_3, p_4]$ nur überführt werden, indem auf den Platz ϵ eine Marke gelegt wird. Ansonsten ist keine Überführung dieser beiden Markierungen möglich. Befindet sich das Netz N_1 in einer solchen Markierung, dann könnte die nächstmögliche Überführung der Marken in das Zielnetz erst in der Endmarkierung stattfinden.

4 Instantane Laufzeitersetzungen

Bei der Laufzeitersetzung von Serviceautomaten [Lis07] werden Zustände des Ursprungsautomaten überführt in Zustände des Zielautomaten. Ein Serviceautomat befindet sich stets in genau einem Zustand. Somit ist der Ausgangsautomat nach dem Schalten der Laufzeitersetzungstransition inaktiv. Die Überführung wurde durch sogenannte τ -Transitionen repräsentiert, also solche Transitionen, die keine Nachricht vom Nachrichtenkanal nehmen und keine neue hineingeben.

Analog zu dem Vorgehen bei Serviceautomaten untersuchen wir in diesem Kapitel zunächst die sogenannten *instantanen Laufzeitersetzungen* zweier offener Workflownetze. Bei dieser Art von Laufzeitersetzung wird die Eigenschaft übernommen, das Ursprungsnetz durch eine einzige Laufzeitersetzungstransition komplett zu leeren. Das bedeutet, dass alle Marken des Ausgangsnetzes gleichzeitig konsumiert und neue Marken im Zielnetz produziert werden. Damit übertragen wir grundlegend die Idee der Überführung von Serviceautomaten auf offene Workflownetze.

Zu Beginn der Interaktion ist das Ausgangsnetz aktiviert, nach dem Schalten der Laufzeitersetzungstransition soll es deaktiviert sein. Umgekehrt soll das Zielnetz solange deaktiviert sein, bis eine der Laufzeitersetzungstransitionen geschaltet hat. Wie wir später sehen werden, ist die Deaktivierung des Ausgangsnetzes wegen der Identität der Interfaceplätze nicht offensichtlich und wird erst durch strukturelle Bedingungen an die beteiligten Netze sichergestellt.

Ein Ziel dieser Arbeit ist, die Laufzeitersetzung speziell für eine vorgegebene Menge von Partnerstrategien zu erzeugen. Diese Strategien seien im folgenden stets durch den annotierten Automaten $OG = R^\phi$ gegeben. Wie schon in der Definition von Laufzeitersetzungen gefordert, handelt es sich dabei um Strategien für das Ausgangsnetz N_1 , d.h. $OG \sqsubseteq OG_{N_1}$.

Im Abschnitt 3.1 haben wir bereits die Wissensfunktionen für offene Workflownetze kennengelernt. In den folgenden Betrachtungen setzen wir die Erreichbarkeitsfunktion K_1 , die den Zuständen von R die erreichbaren Markierungen des Ausgangsnetzes N_1 zuordnet, sowie die Fortführbarkeitsfunktion K_2 , für die entsprechend fortführbaren Markierungen des Zielnetzes N_2 , als gegeben voraus.

Definition 4.1 (instantane Laufzeitersetzung)

Sei $\mathcal{T} = (T_{\mathcal{T}}, F_{\mathcal{T}}, W_{\mathcal{T}})$ eine Laufzeitersetzung von $N_1 = [P_I \cup P_1 \cup P_O, T_1, F_1, m_{0_1}, \Omega_1]$ nach $N_2 = [P_I \cup P_2 \cup P_O, T_2, F_2, m_{0_2}, \Omega_2]$ gemäß OG . \mathcal{T} heißt instantan, wenn nach Schalten einer beliebigen Transition $t \in \mathcal{T}$ gilt: $m(p) = 0, \forall p \in P_1$.

In diesem Kapitel meinen wir mit \mathcal{T} – wenn nicht anders erwähnt – eine Laufzeitersetzung vom Netz N_1 nach N_2 gemäß der Bedienungsanleitung $OG = R^\phi$, wie in der Definition 4.1 beschrieben. Zur einfachen Formulierung der nachstehenden Aussagen und Beweise sprechen wir lediglich von einer *erreichbaren Markierung* m_1 bzw. einer *fortführbaren Markierung* m_2 , falls es einen Zustand $q \in Q$ gibt, sodass mit $m_1 \in K_1(q)$ bzw. $m_2 \in K_2(q)$.

Instantane Laufzeitersetzungen gibt es in jedem Fall, unabhängig davon, welche Workflownetze ersetzt und welche Partnerstrategien der Laufzeitersetzung zugrunde gelegt werden sollen. Schließlich ist jede triviale Laufzeitersetzung, offensichtlich auch instantan. Interessanter ist natürlich die Frage, ob es auch nicht-triviale instantane Laufzeitersetzungen gibt.

Ein Beispiel für eine nicht-triviale instantane Laufzeitersetzung zeigt Abbildung 4.1. Dieser Laufzeitersetzung liegt als Strategiemenge OG die Bedienungsanleitung von Abbildung 2.4 zugrunde. Offensichtlich gibt es gemäß OG neben den dargestellten noch weitere Laufzeitersetzungstransitionen – beispielsweise $[p_3] \rightarrow \square \rightarrow [v_2]$.

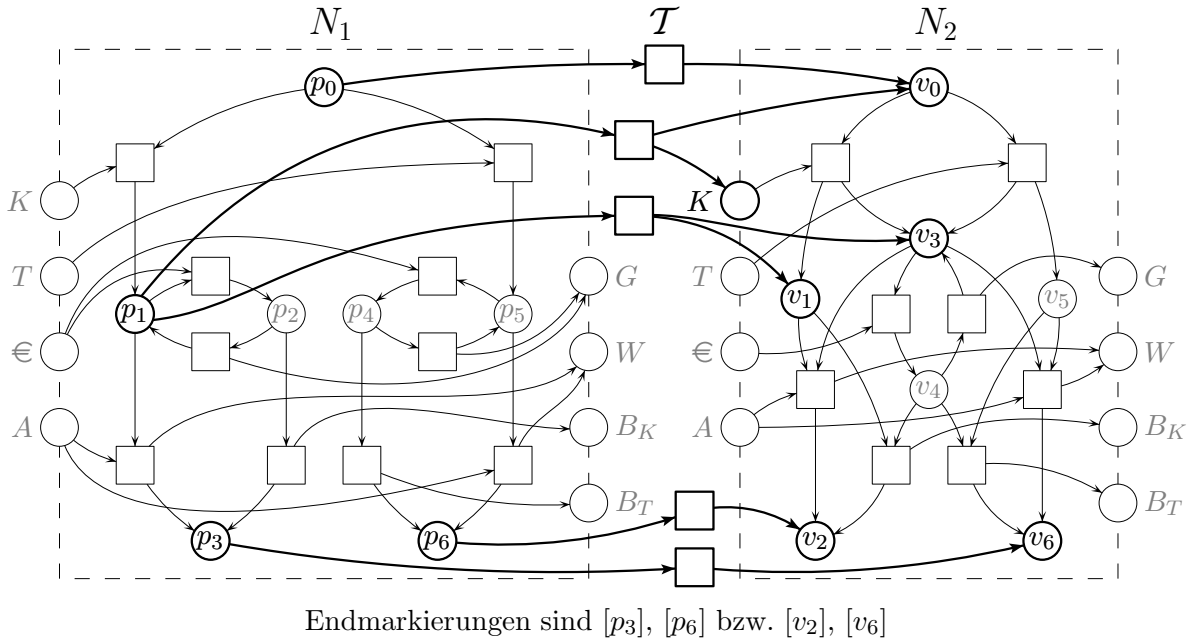


Abbildung 4.1: instantane Laufzeitersetzung gemäß OG von Abb. 2.4

Für Laufzeitersetzungen von Serviceautomaten wurde das Kriterium der Maximalität definiert [Lis07]. Wir wollen dieses Kriterium nun ähnlich für instantane Laufzeitersetzungen offener Workflownetze formulieren.

Definition 4.2 (vollständige instantane Laufzeitersetzung)

Eine instantane Laufzeitersetzung \mathcal{T} heißt vollständig, falls sie alle instantanen Laufzeitersetzungstransitionen enthält. Als instantane Laufzeitersetzungstransition bezeichnen wir dabei all jene Transitionen, deren Vorbereich eine erreichbare Markierung des Ausgangsnetzes darstellt und die in einer beliebigen instantanen Laufzeitersetzung enthalten sind.

Gemäß dieser Definition ist die vollständige instantane Laufzeitersetzung genau die Vereinigung aller instantanen Laufzeitersetzungen. Man überzeugt sich leicht davon, dass diese eindeutig ist. Die in Abbildung 4.1 dargestellte instantane Laufzeitersetzung ist, wie wir gesehen haben, nicht vollständig. Unser Ziel ist daher, einen Algorithmus anzugeben, der stets die vollständige instantane Laufzeitersetzung berechnet.

4.1 Algorithmus zur Bestimmung instantaner Laufzeitersetzungen

Die im Abschnitt 3.1 kennengelernten Wissensfunktionen bilden die Grundlage des folgenden Algorithmus zur Bestimmung instantaner Laufzeitersetzungen. Gegeben seien zwei Workflownetze N_1 und N_2 , welche keine leeren Transitionen im inneren Netz besitzen, sowie ein annotierter Automat $OG = R^\phi \sqsubseteq OG_{N_1}$ zur Beschreibung aller Partnerstrategien.

Es folgt der Algorithmus $\text{INSTANTANELE}(N_1, N_2, R)$. Er bestimmt eine Menge instantaner Laufzeitersetzungstransitionen von N_1 nach N_2 . Der Algorithmus sucht dazu zu jeder erreichbaren Markierung des Ausgangsnetzes passende fortführende Markierungen des Zielnetzes. Sein Vorgehen dabei lässt sich wie folgt zusammenfassen: Der Algorithmus bestimmt zuerst die Erreichbarkeitsfunktion für das Netz N_1 und die Fortführbarkeitsfunktion für das Netz N_2 . Dann beginnt er, die triviale Laufzeitersetzung nach und nach zur vollständigen Laufzeitersetzung zu erweitern. Für alle erreichbaren Markierungen des Ausgangsnetzes werden die potentiell fortführenden Markierungen des Zielnetzes untersucht (Zeile 5–6). Dabei fallen bestimmte Paarungen von Ausgangs- und Zielmarkierung, die bei der Laufzeitersetzung zu Deadlocks führten, aus der Betrachtung heraus (Zeile 7–10). Für die anderen wird jeweils eine instantane Laufzeitersetzungstransition erzeugt (Zeile 11–12).

4 Instantane Laufzeitersetzungen

```

INSTANTANELE( $N_1, N_2, R$ )
(1)  $K_1 := \text{ERREICHBARKEIT}(N_1, R)$ 
(2)  $K_2 := \text{FORTFUEHRBARKEIT}(N_2, R)$ 
(3)  $\mathcal{T} := \{m_f \rightarrow \square \rightarrow m'_f \mid m_f \in \Omega_1 \text{ und } m'_f \in \Omega_2\}$ 
(4) foreach  $q \in Q$ 
(5)   foreach Markierung  $m_1 \in K_1(q)$ ,
      die von keiner erreichbaren Markierung echt überdeckt wird
(6)      $M := \{m_2^N + (m_2^I - m_1^I) \mid$ 
            $m_2 \in K_2(q), m_1^O = m_2^O \text{ und } m_1^I \leq m_2^I\}$ 
(7)   foreach  $q' \in Q$ 
(8)     foreach  $m_2 \in M$ 
(9)       if  $\exists m$  mit  $(m_1^N + m) \in K_1(q')$  und  $(m_2 + m) \notin K_2(q')$ 
(10)         $M := M \setminus \{m_2\}$ 
(11)   foreach  $m_2 \in M$ 
(12)      $\mathcal{T} := \mathcal{T} \cup \{m_1^N \rightarrow \square \rightarrow m_2\}$ 
(13) return  $\mathcal{T}$ 

```

Algorithmus 4.2: Berechnung der instantanen Laufzeitersetzung

Im Detail erläutern wir das Verhalten des Algorithmus beispielhaft anhand der Wissensfunktionen von Abbildung 4.3. Nach Initialisierung der Transitionsmenge \mathcal{T} mit der trivialen Laufzeitersetzung iteriert der Algorithmus in Zeilen 4 und 5 über alle erreichbaren Markierungen des ersten Netzes. Sagen wir, er wählt zuerst den Zustand q_0 , dann findet er die initiale Markierung $m_1 = [p_0]$ (Zeile 5). Als Menge M potentieller Zielmarkierungen werden aus der Fortführbarkeitsfunktion $K_2(q_0)$ anschließend diejenigen Markierungen ausgewählt, die dieselben Ausgabeplätze wie m_1 und mindestens alle Eingabepätze dieser Markierung haben (Zeile 6). Diese Bedingung erfüllt hier nur $[v_0]$.

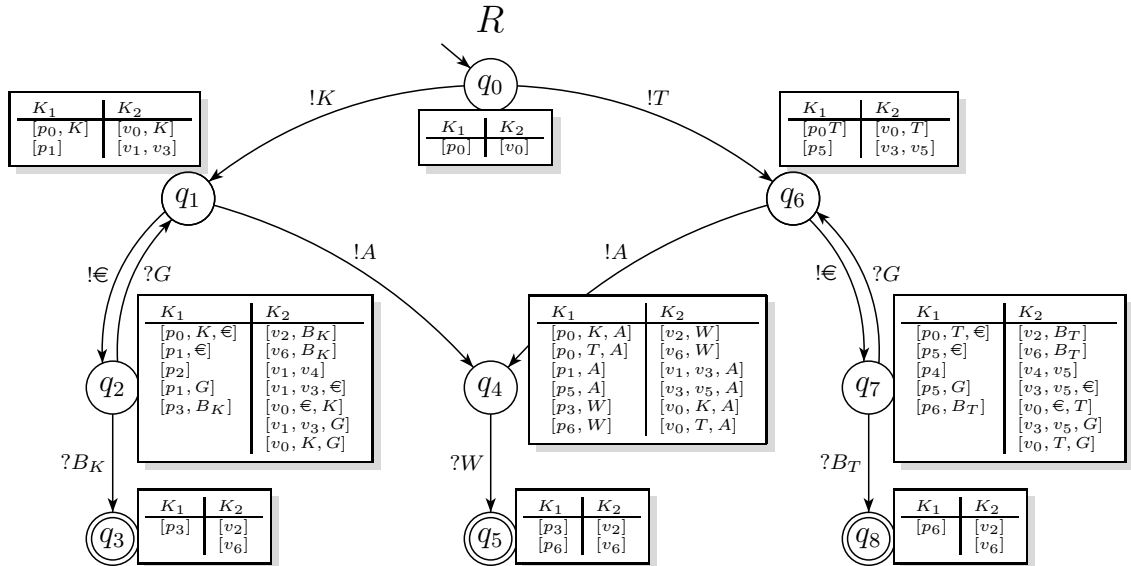


Abbildung 4.3: Wissensfunktionen K_1 und K_2 für die Netze aus Abb. 4.1

4.1 Algorithmus zur Bestimmung instantaner Laufzeitersetzungen

Der Algorithmus überprüft weiter alle Zustände, bei denen $m_1^N = [p_0]$ in K_1 enthalten ist. Dies sind die Zustände q_1, q_2, q_4, q_5 und q_7 . Zum besseren Verständnis zeigt die Tabelle 4.4 die Wissensfunktionen für diese Zustände mit Unterscheidung zwischen inneren und Interface-Plätzen auf.

Zustand	Markierung in $K_1()$	Markierungen in $K_2()$
q_1	$[p_0] + [K]$	$[v_0] + [K]$
q_2	$[p_0] + [K, \text{€}]$	$[v_0] + [K, \text{€}]$
q_4	$[p_0] + [K, A]$ $[p_0] + [T, A]$	$[v_0] + [K, A]$ $[v_0] + [T, A]$
q_6	$[p_0] + [T]$	$[v_0] + [T]$
q_7	$[p_0] + [T, \text{€}]$	$[v_0] + [T, \text{€}]$

Tabelle 4.4: Ausschnitt der Wissensfunktionen für Abb. 4.3

Diejenigen dieser Zustände, bei welchen $[p_0]$ als erreichbare Markierung des Ausgangsnetzes vorkommt, enthalten als fortführbare Markierung im Zielnetz $[v_0]$. Die Überprüfung in Zeile 9 des Algorithmus lässt diese Markierung zu, sodass wir die neue instantane Laufzeitersetzungstransition $[p_0] \rightarrow \square \rightarrow [v_0]$ erhalten.

Als nächster Zustand sei in Zeile 4 q_1 ausgewählt, die nächste betrachtete Markierung des Ausgangsnetzes ist dann $m_1 = [p_1]$. Die potentiellen Zielmarkierungen sind hierfür $M = \{[v_0, K], [v_1, v_3]\}$. Weil diese in Zeile 9 nicht verworfen werden, fügt der Algorithmus die zwei instantanen Laufzeitersetzungstransitionen $[p_1] \rightarrow \square \rightarrow [v_1, v_3]$ und $[p_1] \rightarrow \square \rightarrow [v_0, K]$ hinzu. Er fährt so fort, bis alle erreichbaren Markierungen des Ausgangsnetzes überprüft worden sind.

Die vollständige instantane Laufzeitersetzung ist in Tabelle 4.5 dargestellt. Dabei sind die Transitionen optisch getrennt in solche, die nur innere Plätze des Zielnetzes im Nachbereich enthalten (links), und solche, die zusätzlich Eingabepplätze belegen (rechts).

$[p_0] \rightarrow \square \rightarrow [v_0]$	
$[p_1] \rightarrow \square \rightarrow [v_1, v_3]$	$[p_1] \rightarrow \square \rightarrow [v_0, K]$
$[p_2] \rightarrow \square \rightarrow [v_1, v_4]$	$[p_1] \rightarrow \square \rightarrow [v_1, v_4, \text{€}]$
$[p_3] \rightarrow \square \rightarrow [v_2]$	$[p_2] \rightarrow \square \rightarrow [v_1, v_3, \text{€}]$
$[p_3] \rightarrow \square \rightarrow [v_6]$	$[p_2] \rightarrow \square \rightarrow [v_0, K, \text{€}]$
$[p_4] \rightarrow \square \rightarrow [v_4, v_6]$	$[p_5] \rightarrow \square \rightarrow [v_0, T]$
$[p_5] \rightarrow \square \rightarrow [v_3, v_5]$	$[p_6] \rightarrow \square \rightarrow [v_3, v_6, \text{€}]$
$[p_6] \rightarrow \square \rightarrow [v_2]$	$[p_6] \rightarrow \square \rightarrow [v_0, T, \text{€}]$
$[p_6] \rightarrow \square \rightarrow [v_6]$	

Tabelle 4.5: vollständige instantane Laufzeitersetzung

Ein weiteres Beispiel soll zeigen, wozu die Überprüfung in Zeile 9 des Algorithmus benötigt wird. Bei den Netzen zuvor wurde keine Transition durch diese Überprüfung verworfen.

4.1.1 Notwendigkeit der Überprüfung auf Fortführbarkeit

In Abbildung 4.6 sind zwei Netze gegeben, bei denen ohne die Überprüfung in Zeile 9 ein Deadlock in der Komposition aus dem verbundenen Netz und dem Partner entstehen würde. Damit wäre die Transitionsmenge keine Laufzeitersetzung.

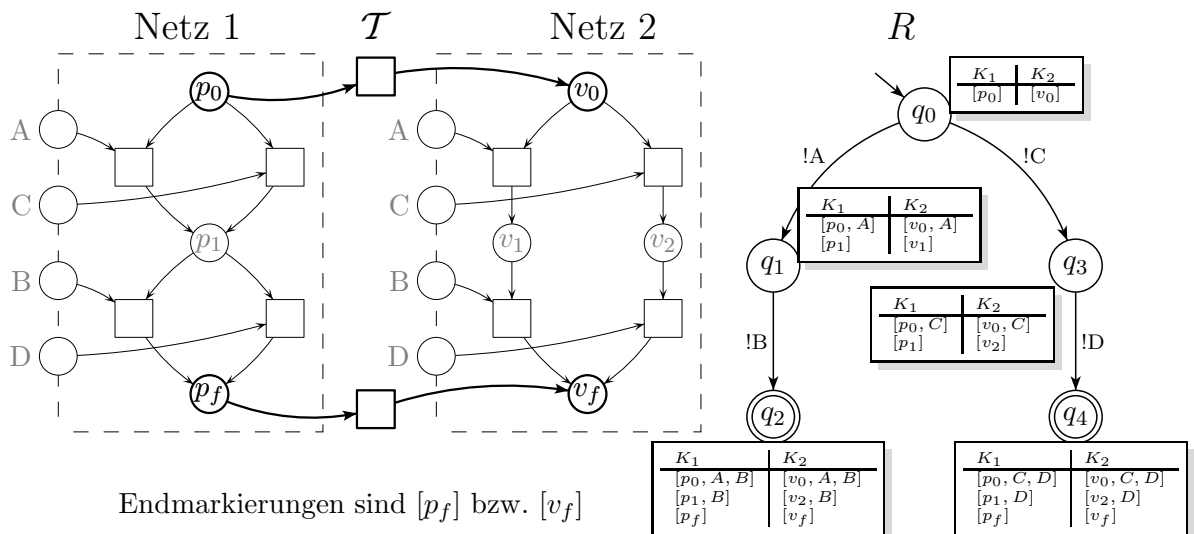


Abbildung 4.6: Notwendigkeit der Überprüfung in Zeile 9

Folgende Konstellation ist in diesem Beispiel gegeben: Die Markierung $[p_1]$ des Ausgangsnetzes ist in $K_1(q_1)$ sowie $K_1(q_3)$ enthalten. Die Markierung $[v_1]$ des Zielnetzes ist analog in $K_2(q_1)$ jedoch nicht in $K_2(q_3)$ enthalten. Bei diesem Fall würde die Bedingung in Zeile 9 des Algorithmus zutreffen und die Transition $[p_1] \rightarrow \square \rightarrow [v_1]$ als Laufzeitersetzungstransition verwerfen. Gäbe es diese Überprüfung nicht, und damit die Transition $[p_1] \rightarrow \square \rightarrow [v_1]$, so gelangt man nach dem Schalten in die Markierung $[v_1]$. Wegen $[p_1] \in K_1(q_3)$ kann diese Transition mitunter dann schalten, wenn sich der Partner gerade im Zustand q_3 befindet. Jedoch ist die Markierung $[v_1]$ nicht fortführbar in q_3 ($[v_1] \notin K_2(q_3)$). Aus diesem Grund werden solche Transitionen bei der Erzeugung der Laufzeitersetzung verworfen.

Ganz allgemein muss gewährleistet sein, dass wenn eine Laufzeitersetzungstransition $m_1 \rightarrow \square \rightarrow m_2$ schaltet, die Markierung m_2 in *allen* Zuständen fortführbar ist, bei denen m_1 erreichbar ist. Anderenfalls ergibt sich, wie wir gesehen haben, in der Komposition aus dem verbundenen Netz mit dem Partner ein Deadlock.

Proposition 4.3 Wird für eine Ausgangsmarkierung m_1 die potentielle Zielmarkierung m_2 in Zeile 9 des Algorithmus verworfen, so hat für ein $R \in \text{Match}(OG)$ die Komposition mit dem verbundenen Netz mit der Überführung $m_1 \rightarrow \square \rightarrow m_2$ einen Deadlock.

4.2 Einschränkungen der Netze

4.2.1 Leere Transitionen im inneren Netz

Abbildung 4.7 zeigt das oWFN N'_2 eines Kaffeeautomaten, der nach Einwurf von € einen Kaffee ausgibt – unabhängig von der Wahl des Kunden. Auf den Abbruchwunsch (A) des Partners reagiert der Automat ebenfalls. Das Netz enthält zwei Transitionen, t_K und t_T , die im inneren Netz leer sind.

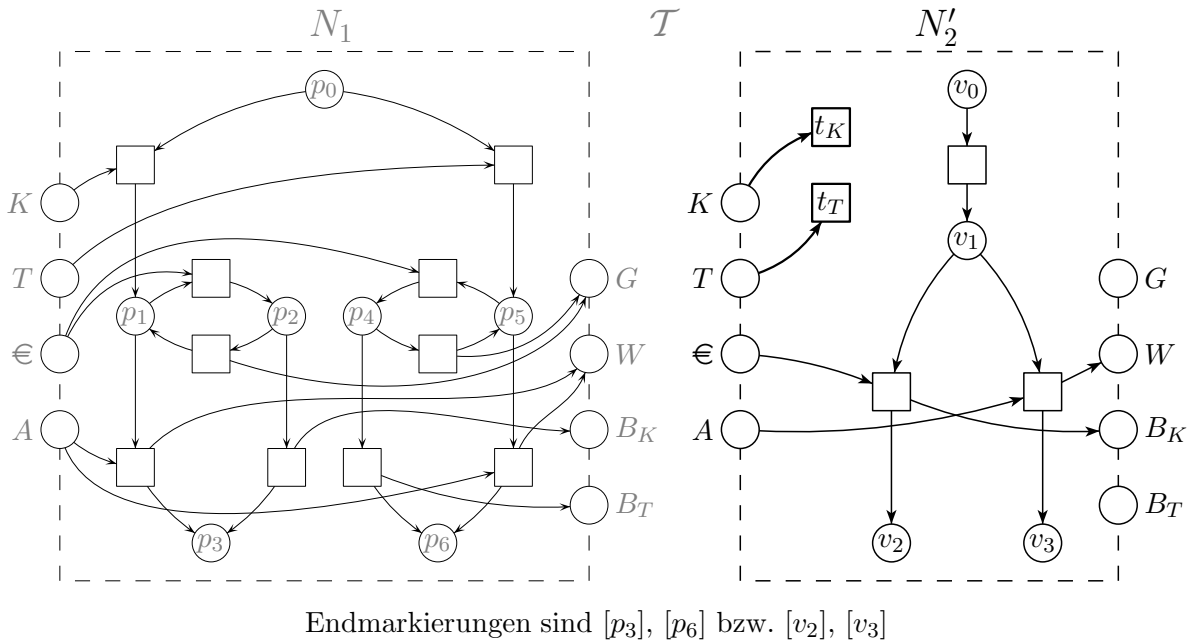


Abbildung 4.7: leere Transitionen im inneren Netz

Intuitiv sollte das Netz N_1 durch N'_2 zur Laufzeit ersetzt werden können. Weil die Interface-Plätze beim verbundenen Netz verschmolzen werden, ergibt sich tatsächlich aber bereits bei der trivialen Laufzeitersetzung folgendes Problem: Wählt nämlich ein Partner Tee aus, so wird der Platz T markiert. Die Transition t_T des zweiten Netzes ist anschließend aktiviert und kann diese Marke vom Eingabepplatz wieder entfernen. Damit ist aber keine Transition mehr aktiviert, weder im Ausgangs- noch im Zielnetz, obwohl kein Endzustand erreicht wurde. Die Komposition enthält also einen Deadlock.

4 Instantane Laufzeitersetzungen

Wir möchten stattdessen, dass das Zielnetz tatsächlich erst dann aktiviert ist, wenn eine Laufzeitersetzungstransition geschaltet hat. Ebenso soll das Ausgangsnetz deaktiviert sein, sobald eine Laufzeitersetzungstransition geschaltet hat. Also schließen wir leere Transitionen im inneren Netz von vornherein explizit aus.

Durch diese Bedingung ist sichergestellt, dass das Ausgangsnetz nach dem Schalten einer instantanen Laufzeitersetzungstransition auch tatsächlich deaktiviert ist. Ebenso ist aufgrund dieser strukturellen Einschränkung das Zielnetz auch erst dann aktiviert, wenn eine Laufzeitersetzungstransition geschaltet hat.

Lemma 4.4 *Sei N ein oWFN mit beschränktem inneren Netz. Sind alle inneren Plätze von N leer und gibt es im inneren Netz von N keine leeren Transitionen, dann sind alle Transitionen in N deaktiviert.*

Beweis:

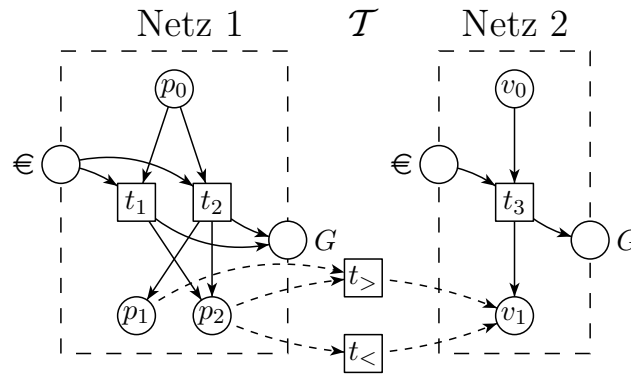
Jede Transition in N besitzt mindestens einen inneren Platz im Vorbereich. Wäre dem nicht so, dann gäbe es eine Transition, welche nur Eingabepätze im Vorbereich besitzt. Diese Transition hat dann mindestens einen inneren Platz im Nachbereich, weil es nach Voraussetzung keine leeren Transitionen im inneren Netz gibt. Dann wäre das innere Netz von N aber nicht beschränkt, Widerspruch zur Annahme. Also besitzt jede Transition einen inneren Platz im Vorbereich, woraus die Aussage folgt. \square

4.2.2 Überdeckte Endmarkierungen

Intuitiv erwarten wir von jeder Laufzeitersetzung, dass zumindest die Endmarkierungen des ersten Netzes in eine Endmarkierung des zweiten Netzes überführt werden können. Schließlich sind die Endmarkierungen des zweiten Netzes gleichzeitig die Endmarkierungen des bei der Laufzeitersetzung verbundenen Netzes.

Wir werden nun zeigen, dass Endmarkierungen, die von beliebigen Markierungen echt überdeckt werden, problematisch bei der Erzeugung von Laufzeitersetzungen sind. In Abbildung 4.8 sind dazu zwei Netze dargestellt, für die es keine instantane Laufzeitersetzung gibt.

Das erste Netz befindet sich nach Schalten der Transition t_1 in der Endmarkierung $[p_2]$, während t_2 in die Endmarkierung $[p_1, p_2]$ schaltet. Offenbar überdeckt die zweite Endmarkierung echt die Endmarkierung $[p_2]$. Das zweite Netz besitzt nur eine Endmarkierung, $[v_1]$. Betrachten wir die möglichen Transitions Mengen von N_1 nach N_2 , so sehen wir, dass es in der Komposition des verbundenen Netzes mit einem Partner stets zu einem Deadlock kommt.



Endmarkierungen sind $[p_2]$ und $[p_1p_2]$ bzw. $[v_1]$

Abbildung 4.8: überdeckte Endmarkierung

- Lassen wir als Überführung die Transition $t_< = [p_2] \rightarrow \square \rightarrow [v_1]$ zu, so bleibt nach dem Schalten von t_2 und $t_<$ die Markierung $[p_1]$ übrig.
- Nehmen wir stattdessen $t_<$ nicht zur Laufzeitersetzung hinzu, dafür aber die Transition $t_> = [p_1, p_2] \rightarrow \square \rightarrow [v_1]$, so erhalten wir nach Schalten von t_1 die Markierung $[p_2]$, welche nicht in das zweite Netz überführt wird.
- Offensichtlich erhalten wir ebenfalls einen Deadlock in der Komposition, wenn gar keine der genannten Laufzeitersetzungstransitionen verwendet wird.

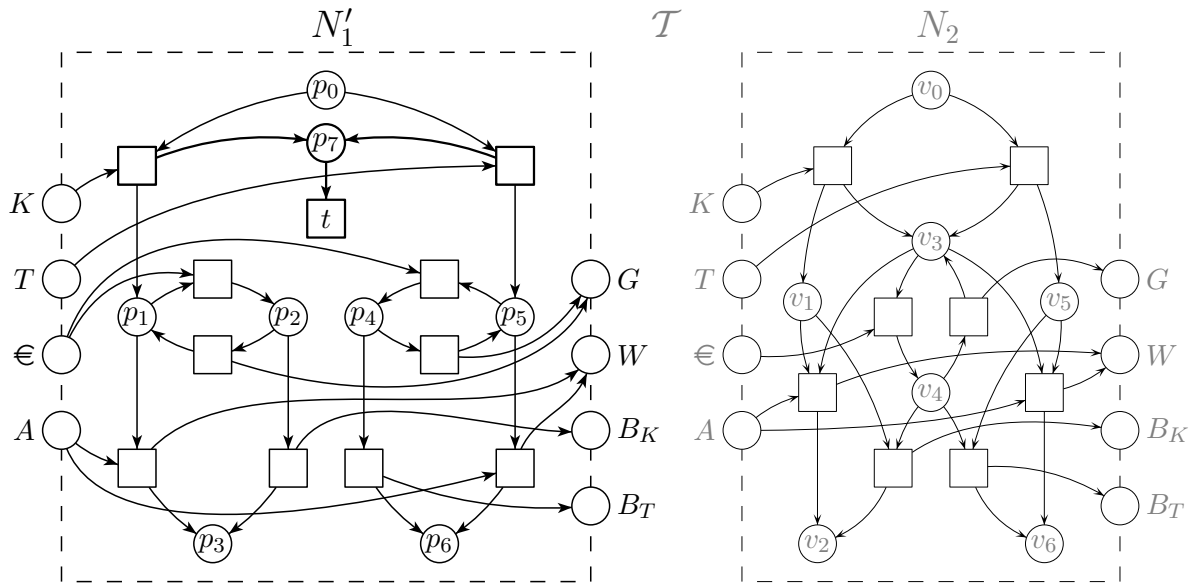
Bei Workflownetzen kann der Modellierer überdeckte Endmarkierungen auflösen, indem er eindeutig macht, welche Transition als letzte geschaltet hat. Für das oben genannte Beispiel können die Plätze p_{t_1} und p_{t_2} neu angelegt werden. Dann müsste die Transition t_1 im Nachbereich $[p_2, p_{t_1}]$ belegen und t_2 die Markierung $[p_1, p_2, p_{t_2}]$. Diese neuen Endmarkierungen werden nicht überdeckt.

Der Verbot echt überdeckter Endmarkierungen stellt also keine Einschränkung für die Ausdrucksmächtigkeit der gegebenen Netze dar. Daher gehen wir o.B.d.A. davon aus, dass im Ausgangsnetz keine Endmarkierung von einer erreichbaren Markierung überdeckt wird.

4.2.3 Überdeckte Markierungen

Überdeckte Markierungen, die keine Endmarkierungen sind, führen nicht automatisch zu den zuvor beschriebenen Problemen. Allerdings gibt es auch für überdeckte Markierungen ungewollte Effekte bei der Laufzeitersetzung.

Abbildung 4.9 zeigt eine leicht modifizierte Variante des Ausgangsnetzes für das Beispiel der zwei Getränkeautomaten. Der einzige Unterschied gegenüber N_1 in N'_1 ist, dass anfangs p_7 belegt und irgendwann geleert wird.



Endmarkierungen sind $[p_3], [p_6]$ bzw. $[v_2], [v_3]$

Abbildung 4.9: überdeckte Markierung

Die vollständige instantane Laufzeitersetzung von N'_1 als Ausgangsnetz zum Zielnetz N_2 sieht ähnlich zur Laufzeitersetzung von N_1 nach N_2 aus (vgl. Tabelle 4.5). Die Vorbereiche aller instantanen Laufzeitersetzungstransitionen enthalten – außer bei der Endmarkierung – zusätzlich den Platz p_7 . Die Konsequenz für das Verhalten des verbundenen Netzes ist, dass – sobald die Transition t den Platz p_7 geleert hat, keine Laufzeitersetzungstransition außer jener zur Überführung des Endzustandes schalten kann.

Die Existenz instantaner Laufzeitersetzungen ist unabhängig davon, ob es überdeckte erreichbare Markierungen im Ausgangsnetz gibt. Möchte man das zuvor beschriebene Verhalten jedoch vermeiden, so sollten die überdeckten Markierungen analog zum Vorgehen bei den überdeckten Endmarkierungen beim Entwurf des offenen Workflownetzes beseitigt werden.

4.3 Korrektheit des Algorithmus INSTANTANELE

In den vorherigen beiden Abschnitten haben wir den Algorithmus 4.2 zur Bestimmung instantaner Laufzeiteretzungen vorgestellt sowie strukturelle Bedingungen an die gegebenen offenen Workflownetze erläutert. In diesem Abschnitt beweisen wir formal die korrekte Funktionsweise des Algorithmus.

Satz 4.5 *Der Algorithmus INSTANTANELE(N_1, N_2, R) berechnet eine instantane Laufzeiteretzung von N_1 nach N_2 gemäß $OG = R^\phi$.*

Beweis:

Der Beweis orientiert sich an der Definition einer Laufzeiteretzung (s. Def. 3.8). Die ersten beiden Punkte zeigen, dass es sich bei \mathcal{T} um eine Transitionsmenge von N_1 nach N_2 handelt. Der dritte Punkt beweist das deadlockfreie Verhalten sowie die beschränkte Kommunikation.

1. z.z. für alle $t \in \mathcal{T}$ gilt: $\bullet t \in \text{bags}(P_1)$
Dies folgt direkt aus der Konstruktion der Transitionsmenge in Zeile 3 und Zeile 12 des Algorithmus.
2. z.z. für alle $t \in \mathcal{T}$ gilt: $t\bullet \in \text{bags}(P_2 \cup P_1)$
In Zeile 6 des Algorithmus werden die modifizierten Markierungen in der Menge M so gewählt, dass sie keinen Ausgabeplatz besitzen. Damit folgt die geforderte Bedingung ebenfalls aus Zeile 3 und Zeile 12.
3. z.z.: $N_1 \oplus \mathcal{T} \oplus N_2$ ist *deadlockfrei* für jedes $R' \in \text{Match}(OG)$
Nach Lemma 4.4 kann im Netz N_2 keine Transition schalten, solange keine Marken durch eine Laufzeiteretzungstransition von N_1 nach N_2 überführt wurden. Gemäß Zeile 3 des Algorithmus umfasst \mathcal{T} zumindest die trivialen Laufzeiteretzungstransitionen $[m_f] \rightarrow \square \rightarrow [m'_f]$, $m_f \in \Omega_1, m'_f \in \Omega_2$. Da nach Voraussetzung der Partnerautomat R' mit dem Ausgangsnetz deadlockfrei interagiert, wird auf jeden Fall irgendwann eine Laufzeiteretzungstransition schalten können (spätestens in der ursprünglichen Endmarkierung des Ausgangsnetzes).

Der Algorithmus wählt in Zeile 5 keine echt überdeckten Markierungen. Also sind nach dem einmaligen Schalten einer Laufzeiteretzungstransition t die inneren Plätze von N_1 komplett leer. Nach Lemma 4.4 kann nun in N_1 und auch in \mathcal{T} keine Transition mehr schalten.

Damit ist nach dem Schalten von t nur noch das zweite Netz, N_2 , aktiv. Nun ist die Fortführbarkeit des verbundenen Netzes mit einem Partnerautomat R' dadurch gewährleistet, dass die als Nachbereich erzeugten Markierungen m_2 der Wissens-

4 Instantane Laufzeitersetzungen

funktion K_2 für das zweite Netz entnommen sind. Diese enthält schließlich genau die im Netz N_2 fortführbaren Markierungen. Also ist die Komposition deadlockfrei und kommuniziert beschränkt.

Wie bereits oben gezeigt, entfernt jede Transition in \mathcal{T} alle Marken der inneren Plätze von N_1 . Damit ist bewiesen, dass die erzeugte Laufzeitersetzung auch eine instantane Laufzeitersetzung ist. \square

Wir haben bewiesen, dass die vom Algorithmus erzeugte Transitionsmenge tatsächlich eine instantane Laufzeitersetzung darstellt. Dies sagt aber nichts aus über den Nutzen der Laufzeitersetzung. Wie wir gesehen haben, ist bereits allein die Überführung der Endmarkierungen des ersten Netzes in die Endmarkierungen des zweiten Netzes eine instantane Laufzeitersetzung. Daher wollen wir noch die Vollständigkeit der vom Algorithmus gefundenen Lösung beweisen.

Satz 4.6 *Der Algorithmus $\text{INSTANTANELE}(N_1, N_2, R)$ berechnet die vollständige instantane Laufzeitersetzung von N_1 nach N_2 gemäß $OG = R^\phi$.*

Beweis:

Angenommen, \mathcal{T} wäre nicht vollständig. Dann gäbe es mindestens eine instantane Laufzeitersetzungstransition $t = m \rightarrow \square \rightarrow m'$, die nicht in \mathcal{T} enthalten ist. Nach der Definition Laufzeitersetzungen ist dieses m eine erreichbare, *innere* Markierung des ersten Netzes und steht somit in der Erreichbarkeitsfunktion mindestens eines Zustandes q .

Daher hätte der Algorithmus auch mindestens eine Markierung $m_1 = m_1^N + m_1^I + m_1^O$ angeschaut mit $m = m_1^N$. Zudem wäre die Markierung des zweiten Netzes, $m_2 = m' + m_1^I + m_1^O$, fortführbar in q und damit in $K_2(q)$ enthalten.

In Zeile 5 wird keine Markierung ausgewählt, die von einer anderen echt überdeckt wird. Anderenfalls wäre t nicht instantan (N_1 würde nicht komplett geleert). Die einzige andere Möglichkeit, warum der Algorithmus die Transition t verworfen hat, ist somit die Bedingung in Zeile 9. Nach Proposition 4.3 kann t keine Laufzeitersetzungstransition sein. Dies führt zum Widerspruch, also erzeugt der Algorithmus eine vollständige instantane Laufzeitersetzung. \square

Wir haben in diesem Kapitel die instantanen Laufzeitersetzungen kennengelernt und konnten einen Algorithmus angeben, der für zwei gegebene offene Workflownetze und eine vorgegebene Menge von Strategien als Partner die vollständige instantane Laufzeitersetzung erzeugt. Im nächsten Schritt sollen auch nicht-instantane Laufzeitersetzungen untersucht und erzeugt werden.

5 Nebenläufige Laufzeitersetzungen

Die bisher betrachteten instantanen Laufzeitersetzungen haben die Eigenschaft, dass alle Marken des Ausgangsnetzes auf einmal in das Zielnetz überführt werden. Petrinetze – und damit Workflownetze – arbeiten jedoch per Definition nebenläufig. Das bedeutet, die Reihenfolge, in der aktivierte Transitionen schalten, ist nicht festgelegt. Die Nebenläufigkeit – also dass verschiedene Aktionen unabhängig voneinander stattfinden – ist bei Geschäftsprozessen explizit erwünscht. In diesem Kapitel soll nun die Nebenläufigkeit auch für Laufzeitersetzungen genutzt werden. Dadurch werden voneinander unabhängige Teile des Ausgangsnetzes auch voneinander unabhängig in das Zielnetz überführt.

Laufzeitersetzungstransitionen sind Teil des verbundenen offenen Workflownetzes und können theoretisch ebenfalls unabhängig voneinander schalten. Erst die Beschränkung im vorherigen Kapitels auf *instantane* Laufzeitersetzungen erzwingt, dass die komplette Überführung des Ausgangs- in das Zielnetz in einem einzigen Schritt erfolgt. Unser folgender Ansatz ist, die in einer instantanen Laufzeitersetzung enthaltenen Nebenläufigkeiten zu finden und zu separieren. Das bedeutet, wir zerteilen instantane Laufzeitersetzungstransitionen in kleinere Transitionen, um so das Ausgangsnetz in mehreren Schritten nebenläufig in das Zielnetz überführen zu können.

Es gibt eine kanonische aber aufwändige Methode, nebenläufige Laufzeitersetzungen zu finden: Man teilt die Transitionen einer instantanen Laufzeitersetzung beliebig in kleinere Transitionen und prüft jedesmal durch eine vollständige Zustandsraumanalyse, ob die entstehende Komposition deadlockfrei bleibt. Dieser Brute-Force-Ansatz ist wegen der Größe der Komposition allerdings unpraktisch für reale Geschäftsprozesse.

Wir präsentieren im folgenden Abschnitt einen neuen Algorithmus, der Transitionen einer instantanen Laufzeitersetzung in kleinere Transitionen zerteilt. Dadurch entsteht eine neue, nebenläufige Laufzeitersetzung. Die einzelnen Arbeitsschritte des Algorithmus werden anhand verschiedener Beispiele detailliert erklärt. In den weiteren Abschnitten folgen ein formaler Korrektheitsbeweis sowie einige heuristische Modifikationen zur Verbesserung des Ergebnisses. Abschließend verdeutlichen wir den Vorteil von nebenläufigen Laufzeitersetzungen gegenüber instantanen Laufzeitersetzungen. Die zwei folgenden Definitionen erklären die für dieses Kapitel notwendigen Begriffe.

Definition 5.1 (Rechenoperationen auf Transitionen)

Als Summe mehrerer Transitionen t_1, \dots, t_n bezeichnen wir die Transition

$$\sum_i t_i := \left(\sum_i \bullet t_i \right) \rightarrow \square \rightarrow \left(\sum_i t_i \bullet \right).$$

Analog ist die Differenz zweier Transitionen t_1, t_2 definiert als

$$t_1 - t_2 := \left(\bullet t_1 - \bullet t_2 \right) \rightarrow \square \rightarrow \left(t_1 \bullet - t_2 \bullet \right).$$

Voraussetzung zur Bildung der Differenz-Transition ist natürlich, dass die Vor- bzw. Nachbereiche voneinander subtrahiert werden können, also $\bullet t_2 \leq \bullet t_1$ und $t_2 \bullet \leq t_1 \bullet$.

Der Schnitt mehrerer Transitionen t_1, \dots, t_n sei

$$\bigcap_i t_i := \left(\bigcap_i \bullet t_i \right) \rightarrow \square \rightarrow \left(\bigcap_i t_i \bullet \right).$$

Diese Schnitttransition überführt den Schnitt der Vorbereiche auf den Schnitt der Nachbereiche, wobei als Schnitt von Markierungen m_1, \dots, m_n das platzweise Minimum der Marken gemeint sei:

$$\left(\bigcap_i m_i \right) (p) := \min_i m_i(p).$$

Definition 5.2 (überführte, nicht überführbare Markierung)

Sei $\mathcal{T} = (T_{\mathcal{T}}, F_{\mathcal{T}}, W_{\mathcal{T}})$ eine instantane Laufzeitersetzung von N_1 nach N_2 gemäß R^ϕ . Eine Markierung heißt überführt in \mathcal{T} , wenn eine Transition $t \in \mathcal{T}$ mit $m^N = \bullet t$ existiert.

Eine Markierung heißt nicht überführbar, wenn sie in der vollständigen instantanen Laufzeitersetzung nicht überführt wird.

5.1 Algorithmus zum Erzeugen nebenläufiger Laufzeitersetzungen

Der Algorithmus ZERTEILUNG zum Finden nebenläufiger Laufzeitersetzungstransitionen basiert auf einer schrittweisen Aufspaltung der Transitionen einer instantanen Laufzeitersetzung. Anfangs sei eine instantane Laufzeitersetzung von N_1 nach N_2 gemäß OG gegeben, bei der die Vorbereiche der Laufzeitersetzungstransitionen paarweise verschieden sind. Eine solche Transitionsmenge kann aus der vollständigen instantanen Laufzeitersetzung gewonnen werden, indem Transitionen mit doppelt auftretenden Vorbereichen gelöscht werden.

5.1 Algorithmus zum Erzeugen nebenläufiger Laufzeitersetzungen

Der unten dargestellte Algorithmus ZERTEILUNG bestimmt die Menge der nicht überführten Markierungen (Zeile 1) sowie die Menge aller Teilmarkierungen, die in den überführten Markierungen enthalten sind (Zeile 2). Die Zerteilung der Laufzeitersetzungstransitionen findet in den Zeilen 3–8 statt. Dabei wird temporär mit einer Kopie der aktuellen Laufzeitersetzung gearbeitet, welche der Subalgorithmus ZERTEILUNGSSCHRITT verändert. Treten bei einem Zerteilungsschritt Konflikte auf, wird die aktuelle Arbeitskopie der Laufzeitersetzung verworfen. Im Erfolgsfall hingegen wird die Arbeitskopie als neue Laufzeitersetzung übernommen. Zum Schluss werden alle Transitionen entfernt, die Summe anderer Laufzeitersetzungstransitionen sind (Zeilen 9–10).

```

ZERTEILUNG( $N_1, N_2, R, \mathcal{T}$ )
(1)   $B := \{m_b \mid m_b \text{ wird nicht überführt in } \mathcal{T}\}$ 
(2)   $M := \{m \mid \exists t \in \mathcal{T} \text{ mit } m < \bullet t\}$ 
(3)  foreach Markierung  $m \in M$ 
(4)     $\mathcal{T}' := \mathcal{T}$ 
(5)     $B' := B$ 
(6)    if ZERTEILUNGSSCHRITT( $m$ )
(7)       $\mathcal{T} := \mathcal{T}'$ 
(8)       $B := B'$ 
(9)  foreach  $t = \Sigma t_i$  mit  $t \in \mathcal{T}$  und  $t_i \in \mathcal{T}$ 
(10)    $\mathcal{T} := \mathcal{T} \setminus \{t\}$ 
(11) return  $\mathcal{T}$ 

```

Algorithmus 5.1: Berechnung nebenläufiger Laufzeitersetzungen

```

ZERTEILUNGSSCHRITT( $m$ )
(1)   $T := \{t \mid m \leq \bullet t, t \in \mathcal{T}\}$ 
(2)   $m' := \bigcap \{t \bullet \mid t \in T\}$ 
(3)   $\mathcal{T}' := \mathcal{T} \cup \{m \rightarrow \square \rightarrow m'\}$ 
(4)  foreach  $t \in T : m \leq \bullet t$ 
(5)    if  $\bullet t - m = \emptyset$  and  $t \bullet - m' \neq \emptyset$ 
(6)      return false
(7)     $\mathcal{T}' := \mathcal{T}' \cup \{(\bullet t - m) \rightarrow \square \rightarrow (t \bullet - m')\}$ 
(8)  if  $\exists m_b \in B'$  mit  $m \leq m_b$ 
(9)    return false
(10) foreach  $t \in T$ 
(11)   if not ZERTEILUNGSSCHRITT( $\bullet t - m$ )
(12)     return false
(13) return true

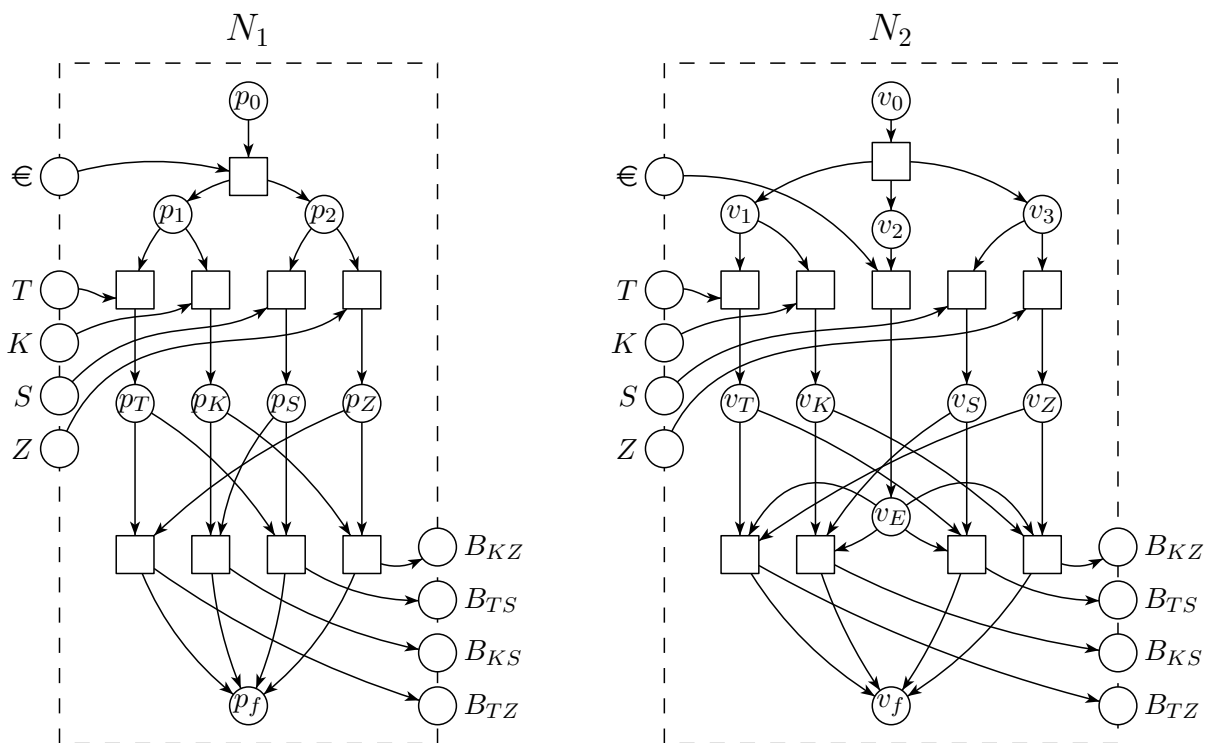
```

Algorithmus 5.2: Zerteilung von Laufzeitersetzungstransitionen

5 Nebenläufige Laufzeitersetzungen

Der Subalgorithmus ZERTEILUNGSSCHRITT erhält als Eingabe eine Markierung m des Ausgangsnetzes. Zuerst bildet er den Schnitt aller Nachbereiche der Transitionen, welche m im Vorbereich enthalten (Zeilen 1–2). Nach der Erzeugung einer neuen Laufzeitersetzungstransition (Zeile 3) wird der Schnitt mit den bestehenden Transitionen gebildet. Es folgt eine mehrschrittige Prüfung auf Deadlocks. Bei Transitionen mit leerem Vorbereich wird die Zerteilung abgebrochen (Zeilen 4–7). Falls eine der nicht überführten Markierungen den Vorbereich überdeckt, wird ebenfalls abgebrochen (Zeilen 8–9). Alle neu entstandenen Transitionen werden zuletzt rekursiv auf Deadlocks überprüft (Zeilen 10–12).

Die genaue Vorgehensweise erläutern wir am Beispiel von Abbildung 5.3. Der Getränkeautomat mit dem Workflownetz N_1 erwartet zuerst die Eingabe von Geld (€) und verarbeitet dann nebenläufig die Wahl eines Getränks (Kaffee C oder Tee T) und die Wahl eines Süßungsmittels (Zucker Z oder Süßstoff S). Zuletzt wird das gewünschte Getränk ausgegeben (G). Der Getränkeautomat gemäß N_2 verhält sich ganz ähnlich. Im Unterschied zu N_1 ist hier die Verarbeitung des Geldes nebenläufig zur Getränke- und Süßungsmittelwahl.



Endmarkierungen sind $[p_f]$ bzw. $[v_f]$

Abbildung 5.3: zwei offene Workflownetze

5.1 Algorithmus zum Erzeugen nebenläufiger Laufzeitersetzungen

Für die betrachteten Beispielnetze verwenden wir folgende instantane Laufzeitersetzung \mathcal{T}_{inst} als Grundlage für die Zerteilung:

$t_1 :$	$[p_0] \rightarrow \square \rightarrow [v_0]$	$t_7 :$	$[p_T, p_S] \rightarrow \square \rightarrow [v_E, v_T, v_S]$
$t_2 :$	$[p_1, p_2] \rightarrow \square \rightarrow [v_1, v_E, v_3]$	$t_8 :$	$[p_T, p_Z] \rightarrow \square \rightarrow [v_E, v_T, v_Z]$
$t_3 :$	$[p_1, p_S] \rightarrow \square \rightarrow [v_1, v_E, v_S]$	$t_9 :$	$[p_K, p_S] \rightarrow \square \rightarrow [v_E, v_K, v_S]$
$t_4 :$	$[p_1, p_Z] \rightarrow \square \rightarrow [v_1, v_E, v_Z]$	$t_{10} :$	$[p_K, p_Z] \rightarrow \square \rightarrow [v_E, v_K, v_Z]$
$t_5 :$	$[p_2, p_T] \rightarrow \square \rightarrow [v_3, v_E, v_T]$	$t_{11} :$	$[p_f] \rightarrow \square \rightarrow [v_f]$
$t_6 :$	$[p_2, p_K] \rightarrow \square \rightarrow [v_3, v_E, v_K]$		

Tabelle 5.4: instantane Laufzeitersetzung zu Abb. 5.3

Beim Aufruf $ZERTEILUNG(N_1, N_2, R, \mathcal{T}_{inst})$ bestimmt unser Algorithmus als erstes die Menge B aller nicht überführten Markierungen (Zeile 2). In unserem Beispiel werden alle erreichbaren Markierungen überführt, somit ist $B = \emptyset$. Später in Abschnitt 5.2.2 behandeln wir ein Beispiel, bei dem B nicht-leer ist.

In Zeile 4 bis 7 iteriert der Algorithmus über alle Teilmarkierungen der Vorbereiche von \mathcal{T} . Sei eine betrachtete solche Markierung $m = [p_1]$. Der Aufruf von $ZERTEILUNGSSCHRITT([p_1])$ bestimmt als erstes alle Laufzeitersetzungstransitionen, in deren Vorbereich m enthalten ist. In unserem Beispiel sind es die drei Transitionen t_2, t_3 und t_4 von Tabelle 5.4, in deren Vorbereich sich $[p_1]$ befindet.

Die Schnittmenge der Nachbereiche dieser drei Transitionen ist $m' = [v_1, v_E]$ (Zeile 2), somit wird die Laufzeitersetzungstransition $[p_1] \rightarrow \square \rightarrow [v_1, v_E]$ eingefügt (Zeile 3). Die anderen Transitionen werden gemäß Zeile 7 des Algorithmus folgendermaßen verändert:

$$\begin{aligned}
 ([p_1, p_2] \rightarrow \square \rightarrow [v_1, v_E, v_3]) - ([p_1] \rightarrow \square \rightarrow [v_1, v_E]) &= [p_2] \rightarrow \square \rightarrow [v_3] \\
 ([p_1, p_S] \rightarrow \square \rightarrow [v_1, v_E, v_S]) - ([p_1] \rightarrow \square \rightarrow [v_1, v_E]) &= [p_S] \rightarrow \square \rightarrow [v_S] \\
 ([p_1, p_Z] \rightarrow \square \rightarrow [v_1, v_E, v_Z]) - ([p_1] \rightarrow \square \rightarrow [v_1, v_E]) &= [p_Z] \rightarrow \square \rightarrow [v_Z]
 \end{aligned}$$

Die Zeilen 8 bis 9 des Subalgorithmus $ZERTEILUNGSSCHRITT$ sind hier unbedeutend, weil es keine nicht überführten Markierungen gibt. In Zeile 11 wird nun die veränderte Transitionsmenge rekursiv weiter zerteilt. Es erfolgt also beispielsweise der Aufruf $ZERTEILUNGSSCHRITT([p_2])$. Dabei wird wiederum $[p_2] \rightarrow \square \rightarrow [v_3]$ als neue Transition hinzugefügt. Rekursiv bleiben $[p_T] \rightarrow \square \rightarrow [v_E, v_T]$ und $[p_K] \rightarrow \square \rightarrow [v_E, v_K]$ zum weiteren Zerteilen übrig. In unserem Beispiel liefern diese Aufrufe jeweils **true** als Ergebnis zurück, sodass alle lokalen Änderungen in die Transitionsmenge \mathcal{T} übernommen werden. Zuletzt werden alle diejenigen Transitionen wieder entfernt, die sich als Summe anderer Transitionen darstellen lassen. Abbildung 5.5 zeigt das Ergebnis von $ZERTEILUNG(N_1, N_2, R, \mathcal{T}_{inst})$.

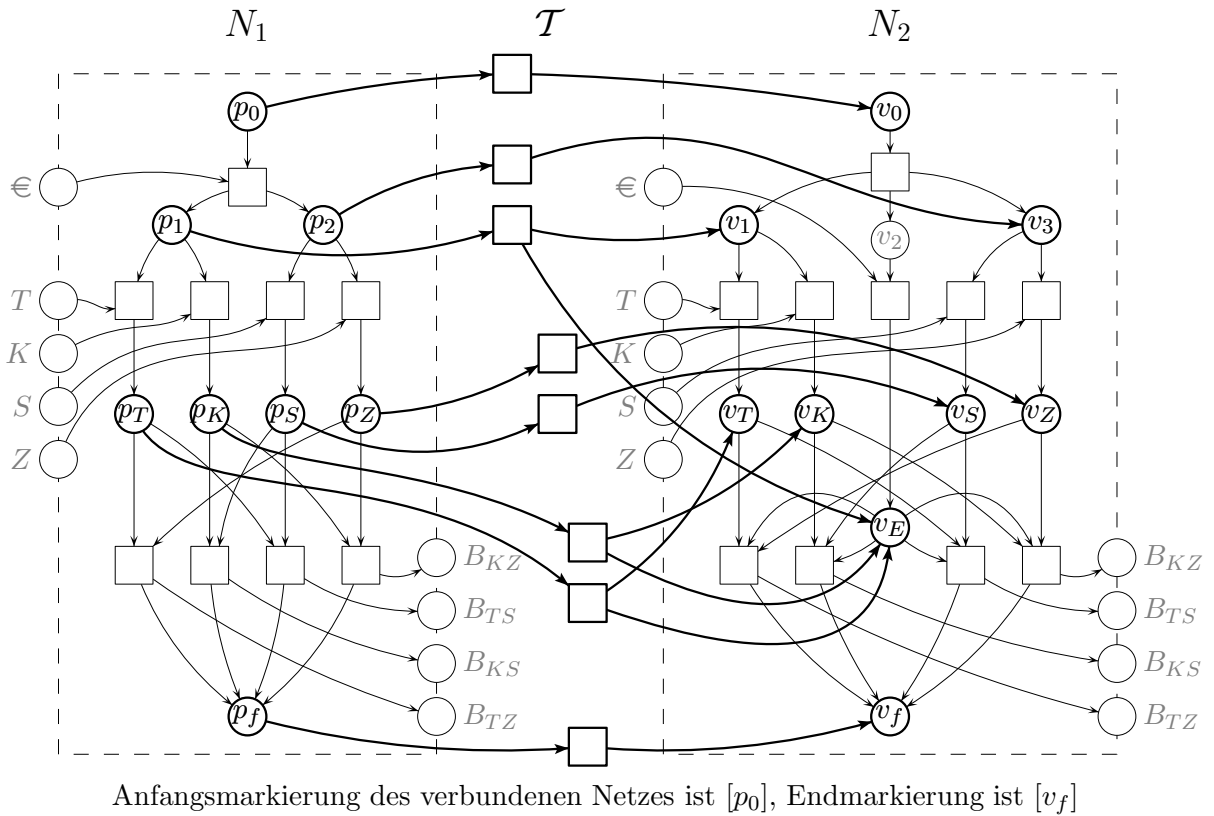


Abbildung 5.5: berechnete nebenläufige Laufzeitersetzung

Man überzeugt sich leicht davon, dass das Ergebnis des Algorithmus ZERTEILUNG nicht eindeutig festgelegt ist. Die zuvor erhaltenen Laufzeitersetzungstransitionen sind zum Vergleich in Tabelle 5.6 aufgelistet. Wählt der Algorithmus in Zeile 3 statt $[p_1]$ zuerst die Markierung $[p_2]$, dann enthält die erzeugte Laufzeitersetzung die Transitionen laut Tabelle 5.7.

$[p_0] \rightarrow \square \rightarrow [v_0]$	$[p_T] \rightarrow \square \rightarrow [v_E, v_T]$
$[p_1] \rightarrow \square \rightarrow [v_1, v_E]$	$[p_K] \rightarrow \square \rightarrow [v_E, v_K]$
$[p_2] \rightarrow \square \rightarrow [v_3]$	$[p_Z] \rightarrow \square \rightarrow [v_Z]$
$[p_f] \rightarrow \square \rightarrow [v_f]$	$[p_S] \rightarrow \square \rightarrow [v_S]$

Tabelle 5.6: zerteilte Laufzeitersetzungstransitionen zur Tab. 5.4

$[p_0] \rightarrow \square \rightarrow [v_0]$	$[p_T] \rightarrow \square \rightarrow [v_T]$
$[p_1] \rightarrow \square \rightarrow [v_1]$	$[p_K] \rightarrow \square \rightarrow [v_K]$
$[p_2] \rightarrow \square \rightarrow [v_3, v_E]$	$[p_Z] \rightarrow \square \rightarrow [v_E, v_Z]$
$[p_f] \rightarrow \square \rightarrow [v_f]$	$[p_S] \rightarrow \square \rightarrow [v_E, v_S]$

Tabelle 5.7: anders zerteilte Laufzeitersetzungstransitionen zur Tab. 5.4

5.2 Korrektheit

Die Korrektheit des im vorherigen Abschnitt vorgestellten Algorithmus ZERTEILUNG wollen wir nun formal beweisen. Zunächst beweisen wir dazu zwei Lemmata für die beiden Abbruchbedingungen des Subalgorithmus ZERTEILUNGSSCHRITT. Diese Aussagen sind für den anschließenden Korrektheitsbeweis notwendig.

5.2.1 Leerer Vorbereich

Transitionen mit leerem Vorbereich können beliebig oft schalten und damit in ihrem Nachbereich beliebig viele Marken erzeugen. Wir fordern jedoch beschränkte Netze für die Interaktion. Dass beim Zerteilen von Transitionen leere Vorbereiche entstehen können, zeigt das folgende Beispiel in Abbildung 5.8. Für dieses Netz erhält man durch den Aufruf ZERTEILUNGSSCHRITT($[p_1]$) zerteilte Transitionen mit leerem Vorbereich.

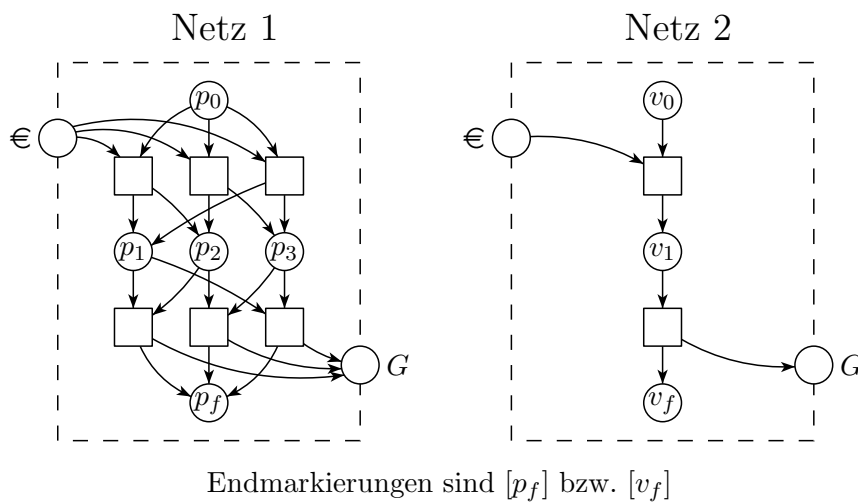


Abbildung 5.8: Test auf leeren Vorbereich

Zunächst wird die Überführung $[p_1] \rightarrow \square \rightarrow [v_1]$ der Menge \mathcal{T}' hinzugefügt. Bei der Subtraktion dieser Transition von den anderen Transitionen aus \mathcal{T}' entstehen die Differenzen $[p_2] \rightarrow \square \rightarrow []$ und $[p_3] \rightarrow \square \rightarrow []$. Infolge dessen werden diese neuen Transitionen auch von der Transition $[p_2, p_3] \rightarrow \square \rightarrow [v_1]$ subtrahiert. Damit bleibt als Ergebnis $[] \rightarrow \square \rightarrow [v_1]$ übrig. Mit dieser Transition wäre das verbundene Netz allerdings nicht mehr beschränkt. Deswegen verwirft der Subalgorithmus ZERTEILUNGSSCHRITT in Zeile 5 und 6 diese Modifikation.

Die Überprüfung in Zeile 5 ist wesentlich für das folgende Lemma.

Lemma 5.3 *Sei \mathcal{T}_{inst} eine instantane Laufzeitersetzung von N_1 nach N_2 gemäß $OG = R^\phi$, bei der die Vorbereiche der Laufzeitersetzungstransitionen paarweise verschieden sind. Sei \mathcal{T} die Transitionsmenge die von $ZERTEILUNG(N_1, N_2, R, \mathcal{T}_{inst})$ bestimmte Transitionsmenge vor dem Löschen der Summentransitionen, also vor Zeile 9 des Algorithmus. Wann immer mehrere Transitionen $t_1, \dots, t_n \in \mathcal{T}$ in der Summe im Vorbereich genau den Vorbereich einer Transition t von \mathcal{T} ergeben, ist $\sum_i t_i = t$.*

Beweis:

Wir zeigen dies induktiv über n .

Induktionsanfang: Für $n = 1$ ist $\bullet t = \bullet t_1$.

Ist t_1 eine instantane Laufzeitersetzungstransition, dann gilt $t = t_1$ offensichtlich. Schließlich enthält \mathcal{T}_{inst} keine zwei Transitionen mit demselben Vorbereich. Sei also $t_1 = m_1 \rightarrow \square \rightarrow m_2$ eine zerteilte Transition. Wir wollen die umgekehrte Annahme, $t \bullet \neq m_2$, zum Widerspruch führen.

In einem Rekursionsschritt des Algorithmus $ZERTEILUNG$ wird auch $ZERTEILUNGSSCHRITT(m_1)$ aufgerufen, da sonst t_1 nicht in \mathcal{T} hätte aufgenommen werden können. In Zeile 2 wird dabei der Schnitt m' der Nachbereiche aller Transitionen, die m_1 im Vorbereich enthalten, gebildet. Zu diesen Nachbereichen gehören auch m_2 und $t \bullet$. Wegen $t \bullet \neq m_2$ ist der Schnitt m' echt kleiner als m_2 .

Der Algorithmus erzeugt eine neue Transition $m_1 \rightarrow \square \rightarrow m'$, die anschließend unter anderem von t_1 subtrahiert wird. Wegen $t_1 - m_1 \rightarrow \square \rightarrow m' = \square \rightarrow \square \rightarrow [m_2 - m']$ hat diese Differenz einen leeren Vorbereich und einen nicht-leeren Nachbereich. Der Algorithmus liefert dann **false** zurück (Zeile 6) und verwirft somit den entsprechenden Zerteilungsschritt. Dies ist ein Widerspruch zur Existenz von t und t_1 in \mathcal{T} , womit die ursprüngliche Behauptung $t_1 = t$ bewiesen ist.

Induktionsbehauptung: Wenn die Aussage für je n Transitionen gilt, gilt sie auch für $n + 1$ Transitionen.

Induktionsbeweis: Seien Transitionen $t_1, \dots, t_{n+1} \in \mathcal{T}$ gegeben, sodass der summierte Vorbereich $\sum_{i=1}^{n+1} \bullet t_i$ auch Vorbereich einer einzelnen Transition t aus \mathcal{T} ist.

Sei $t_1 = m \rightarrow \square \rightarrow m'$. Innerhalb des Aufrufes $ZERTEILUNGSSCHRITT(m)$ ist auch die Differenz $d := t - t_1$ bestimmt worden (Zeile 7). Weil der Vorbereich der Transitionen t_2, \dots, t_{n+1} in der Summe $\bullet d$ ergibt, gilt induktiv $d = \sum_{i=2}^{n+1} t_i$.

Ebenfalls muss $\text{ZERTEILUNGSSCHRITT}(\bullet d)$ aufgerufen worden sein, womit $t - d$ erzeugt worden ist. Es gilt nach Induktionsvoraussetzung $t - d = t_1$. Insgesamt gilt also $t = t_1 + d = \sum_{i=1}^{n+1} t_i$.

□

Als Spezialfall aus dem Lemma 5.3 ergibt sich die folgende Proposition.

Proposition 5.4 *Sei \mathcal{T} die von $\text{ZERTEILUNG}(N_1, N_2, R, \mathcal{T}_{inst})$ erzeugte Transitionsmenge. Wann immer $t_1, \dots, t_n \in \mathcal{T}$ zusammen im Vorbereich eine erreichbare innere Markierung von N_1 ergeben, ist ihre Summe eine instantane Laufzeitersetzungstransition aus \mathcal{T}_{inst} .*

5.2.2 Nicht überführbare Markierungen

Im Allgemeinen sind nicht alle Markierungen, die im Ausgangsnetz erreicht werden können, durch eine instantane Laufzeitersetzungstransition in das Zielnetz überführbar. Entständen bei der Zerteilung instantaner Laufzeitersetzungstransitionen solche Transitionen, deren summierte Vorbereiche eine nicht überführbare Markierung m_b ergeben, so wäre das verbundene Netz mitunter nicht mehr bedienbar. Die Transitionen stellen dann nämlich zusammen eine instantane Überführung dieser Markierung m_b dar. Nach Voraussetzung gibt es keine solche Laufzeitersetzungstransition, daher erhielten wir einen Deadlock in der Komposition. Wir zeigen, dass ist es problematisch ist, nur einen Teil einer nicht überführbaren Markierung als Vorbereich einer Transition zu haben.

In Abbildung 5.9 sind dazu zwei Getränkeautomaten dargestellt, die neben dem Getränk G auch eine Zettel Z ausgeben. Das Netz 1 gibt den Zettel im Falle von Kaffee K dabei unabhängig von der Bezahlung € . Im Falle von Tee T erhält man den Zettel ebenso wie in Netz 2 erst nach dem Bezahlen. Die Strategien der Benutzer sind durch den Automaten R festgelegt. Er ist hier mit den Wissensfunktionen K_1 und K_2 abgebildet.

Bei diesem Beispiel können einige erreichbare Markierungen des Ausgangsnetzes nicht überführt werden. Als instantane Laufzeitersetzung sei folgendes \mathcal{T}_{inst} gegeben:

$$\begin{array}{l|l} [p_0] \rightarrow \square \rightarrow [v_0] & [p_f] \rightarrow \square \rightarrow [v_f] \\ [p_1, p_2] \rightarrow \square \rightarrow [v_1, v_2] & [p_2, p_3] \rightarrow \square \rightarrow [v_2, v_3] \\ [p_1, p_4] \rightarrow \square \rightarrow [v_1, \text{€}] & [p_3, p_4] \rightarrow \square \rightarrow [v_2, \text{€}] \end{array}$$

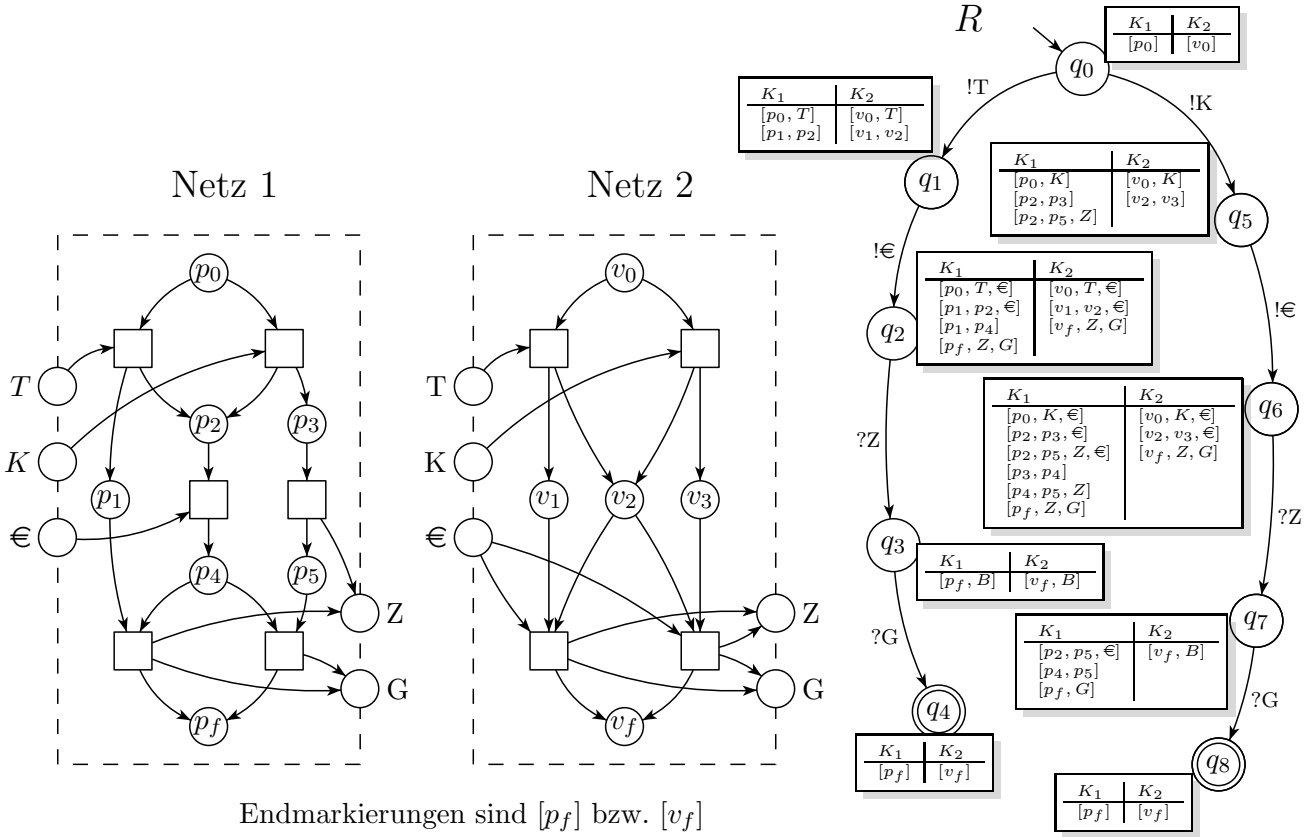


Abbildung 5.9: Test auf nicht-überführbare Markierungen

Die Menge B des Algorithmus ZERTEILUNG enthält die Markierungen

$$[p_2, p_5, Z], [p_2, p_5, \epsilon], [p_2, p_5, Z, \epsilon], [p_4, p_5] \text{ und } [p_4, p_5, Z].$$

Bleibe diese Menge B unberücksichtigt, so entstünden beim Zerteilen die Transitionen $[p_1] \rightarrow \square \rightarrow [v_1]$, $[p_2] \rightarrow \square \rightarrow [v_2]$, $[p_3] \rightarrow \square \rightarrow [v_3]$ sowie $[p_4] \rightarrow \square \rightarrow [\epsilon]$. Damit könnte beispielsweise von der Markierung $[p_4, p_5]$ aus die Laufzeitersetzungstransition $[p_4] \rightarrow \square \rightarrow [\epsilon]$ schalten, sodass die Markierung $[p_5]$ im Ausgangsnetz übrig bliebe. Dies ist ein Deadlock in der Komposition aus dem verbundenen Netz und seinem Partner.

Um diesen Deadlock auszuschließen, überprüft ZERTEILUNGSSCHRITT($[p_4]$) in Zeile 8, ob $[p_4]$ als Teil einer der Markierung aus B vorkommt. Wegen $[p_4, p_5] \in B$ ist dies der Fall und der Aufruf liefert den Wert **false**, wodurch die oben beschriebene, fehlerhafte Teilung verworfen wird.

Für beliebige Netze funktioniert diese Deadlockprüfung ebenfalls, wie das folgende Lemma zeigt.

Lemma 5.5 *Sei \mathcal{T}_{inst} eine instantane Laufzeitersetzung von N_1 nach N_2 gemäß $OG = R^\phi$, bei der die Vorbereiche der Laufzeitersetzungstransitionen paarweise verschieden sind. Sei \mathcal{T} die Transitionsmenge, die von $ZERTEILUNG(N_1, N_2, R, \mathcal{T}_{inst})$ berechnet wurde. Wenn in der Komposition aus verbundenem Netz $N_1 \oplus \mathcal{T} \oplus N_2$ und R keine Transition aus \mathcal{T} schalten kann, obwohl N_1 noch nicht leer ist, dann kann eine Transition aus N_1 schalten.*

Beweis:

Angenommen, N_1 ist nicht leer und keine Transition aus N_1 oder \mathcal{T} kann schalten.

Solange überhaupt keine Laufzeitersetzungstransition geschaltet hat, kann kein Deadlock erreicht worden sein. Die bereits ausgeführten Transitionen aus \mathcal{T} seien t_1, \dots, t_n und m_{akt} sei die aktuelle Markierung von N_1 . Dann ist $m := m_{akt} + \sum_i \bullet t_i$ eine erreichbare

Markierung von N_1 . Gäbe es eine instantane Laufzeitersetzungstransition mit m^N als Vorbereich, dann hätte der Subalgorithmus $ZERTEILUNGSSCHRITT$ auch die Differenz aus dieser Transition und t_1, \dots, t_n zu \mathcal{T} hinzugefügt. Somit könnte jetzt eine Transition aus \mathcal{T} schalten, im Widerspruch zur Voraussetzung.

So ist m^N nicht Vorbereich einer instantanen Laufzeitersetzungstransition, also eine *nicht überführte* Markierung ($m \in B$). Wegen der Bedingung in Zeile 8 von $ZERTEILUNGSSCHRITT(m)$ wäre dann keines der t_i zu \mathcal{T} hinzugefügt worden, Widerspruch. Wenn N_1 nicht leer ist und keine Transition aus \mathcal{T} aktiviert ist, kann somit eine Transition aus N_1 schalten. \square

5.2.3 Korrektheit des Algorithmus Zerteilung

Mithilfe der vorherigen Lemmata führen wir nun den formalen Korrektheitsbeweis für den Algorithmus $ZERTEILUNG$.

Satz 5.6 *Sei \mathcal{T}_{inst} eine instantane Laufzeitersetzung, bei welcher die Vorbereiche der Transitionen paarweise verschieden sind. Die von $ZERTEILUNG(N_1, N_2, R, \mathcal{T}_{inst})$ erzeugte Transitionsmenge \mathcal{T} stellt eine korrekte Laufzeitersetzung von N_1 nach N_2 gemäß $OG = R^\phi$ dar.*

Beweis:

Die erzeugte Menge \mathcal{T} ist offensichtlich eine Transitionsmenge von N_1 nach N_2 . Es genügt somit zu zeigen, dass die Komposition des verbundenen Netzes $N_1 \oplus \mathcal{T} \oplus N_2$ mit Interaktionspartnern gemäß OG keine Deadlocks enthält. Wir unterscheiden dazu zwei Fälle – je nachdem, ob in der aktuellen Markierung noch innere Plätze des Ausgangsnetzes markiert sind oder nicht.

Fall 1: P_1 ist nicht leer.

Sollte noch keine Laufzeitersetzungstransition geschaltet haben, dann hat die bisherige Interaktion nur zwischen dem Partner und N_1 stattgefunden. Dabei werden keine Deadlocks erreicht, weil $OG \sqsubseteq OG_{N_1}$.

Haben hingegen bereits Transitionen $t_1, \dots, t_n \in \mathcal{T}$ geschaltet, dann ist ebenfalls kein Deadlock erreicht. Schließlich kann gemäß Lemma 5.5 eine weitere Laufzeitersetzungstransition, eine Transition des Ausgangsnetzes oder der Interaktionspartner schalten.

Fall 2: P_1 ist leer.

Dann haben bereits die Laufzeitersetzungstransitionen $t_1, \dots, t_n \in \mathcal{T}$ geschaltet mit $t_\Sigma := \sum_{i=1}^n \bullet t_i = m_1$, wobei m_1 eine erreichbare Markierung von N_1 ist. Gemäß Proposition 5.4 ist t_Σ eine instantane Laufzeitersetzungstransition, weswegen der Nachbereich eine in N_2 fortführbare Markierung ist. Demnach kann in der Komposition kein Deadlock erreicht werden.

In beiden Fällen ist die Komposition deadlockfrei, \mathcal{T} ist also eine Laufzeitersetzung von N_1 nach N_2 gemäß OG . \square

5.3 Heuristik

Der vorangegangene Korrektheitsbeweis hat gezeigt, dass der Algorithmus ZERTEILUNG in der Lage ist, aus einer instantanen Laufzeitersetzung eine nebenläufige Laufzeitersetzung zu erzeugen. Wir werden sehen, dass der Algorithmus – so wie er beschrieben ist – nicht immer die „schönste“ Lösung findet. Begründet ist dies in der Prüfung in Zeile 8 des Subalgorithmus ZERTEILUNGSSCHRITT auf potentiell deadlock-verursachende Transitionen. Offensichtlich werden umso häufiger Zerteilungsschritte verworfen, je mehr Markierungen in der Menge B enthalten sind.

In diesem Abschnitt diskutieren wir verschiedene Möglichkeiten, seltener Zerteilungsschritte zu verwerfen, unter Bewahrung der Deadlockfreiheit der Komposition aus verbundenem Netz und Interaktionspartner. Einige Ansätze, dies zu erreichen, sind:

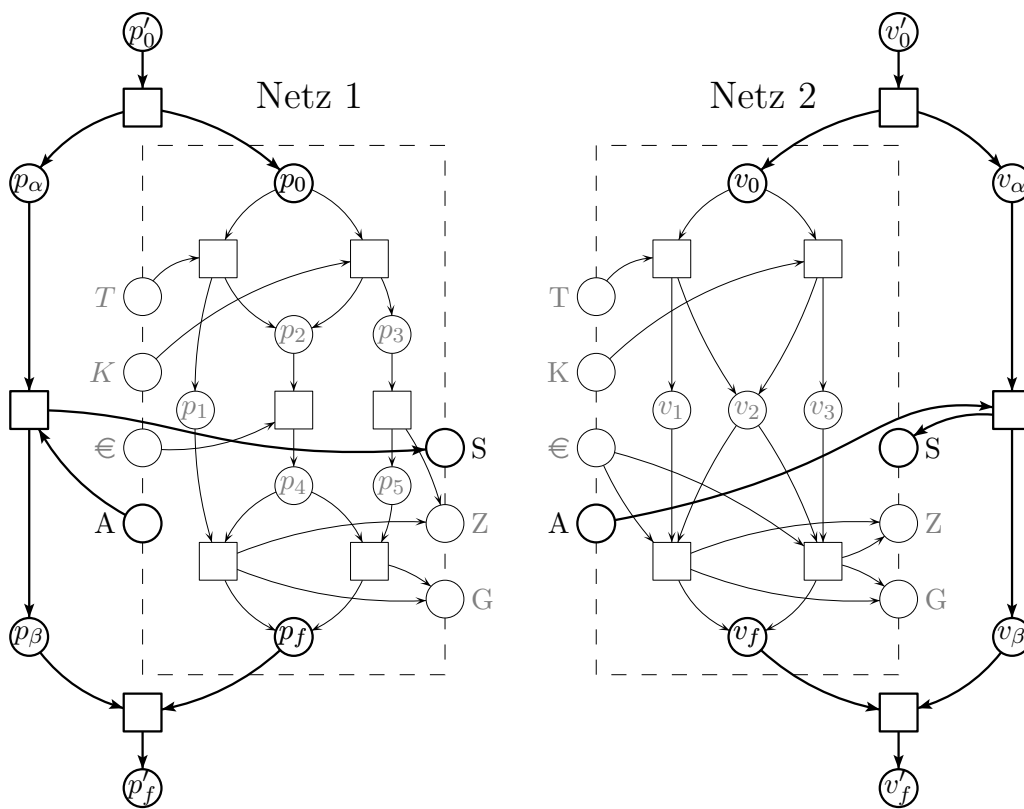
- alle überführbaren Markierungen in der Transitionsmenge \mathcal{T} auch tatsächlich zu überführen – damit enthält B nur *nicht überführbare* Markierungen,
- differenzierte Deadlockprüfung – für manche nicht überführbaren Markierungen kann das Netz auch dann schalten, wenn ein Teil bereits überführt wurde,

- die Menge B verkleinern – bei manchen nicht überführbaren Markierungen kann in der Komposition stets der Partner R agieren.

Der erste Ansatz bezieht sich auf die Menge \mathcal{T}_{inst} , die beim Aufruf von ZERTEILUNG(N_1 , N_2 , R , \mathcal{T}_{inst}) als Eingabe gegeben ist. Die anderen beiden Ansätze sollen mithilfe geeigneter Modifikationen des Algorithmus umgesetzt werden. Wir präsentieren jeweils eine Heuristik mit erklärendem Beispiel. Zusätzlich beweisen wir formal die Korrektheit der vorgestellten Algorithmusvarianten.

5.3.1 differenzierte Deadlockprüfung

Motiviert ist die folgende heuristische Analyse durch das Beispiel in Abbildung 5.10. Die beiden Netze sind fast identisch mit denen aus Abbildung 5.9. Der einzige Unterschied ist, dass nebenläufig zu dem oben beschriebenen Verhalten noch eine weitere Eingabe A verarbeitet und die Nachricht S ausgegeben wird. Der Automat R in Abbildung 5.11 beschreibt die betrachtete Strategie.



Anfangsmarkierungen sind $[p'_0]$ bzw. $[v'_0]$, Endmarkierungen sind $[p'_f]$ bzw. $[v'_f]$

Abbildung 5.10: Beispielnetze für Heuristiken

5 Nebenläufige Laufzeitersetzungen

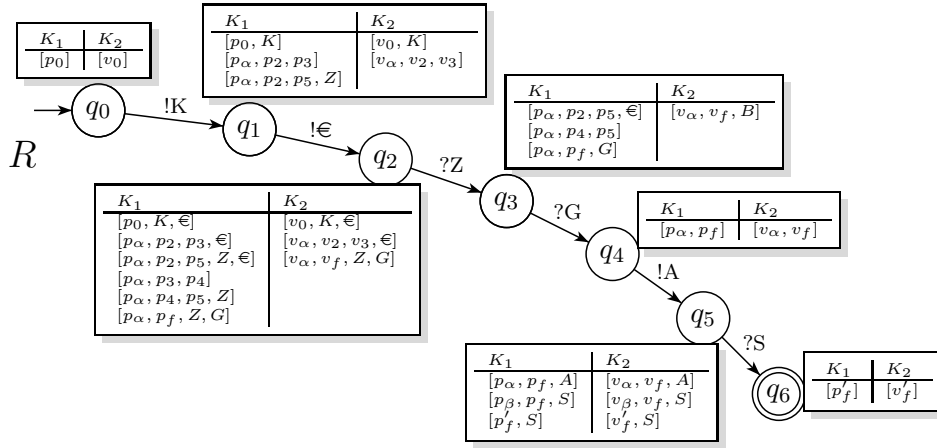


Abbildung 5.11: Serviceautomat mit Wissensfunktion für die Netze aus Abb. 5.10

Man erkennt leicht, dass die Transitionen der Laufzeitersetzung für das Beispiel aus Abbildung 5.9 zusammen mit den Transitionen $[p_\alpha] \rightarrow \square \rightarrow [v_\alpha]$ und $[p_\beta] \rightarrow \square \rightarrow [v_\beta]$ eine nebenläufige Laufzeitersetzung für die zwei Netze aus Abbildung 5.10 sind. Die Menge B enthält in diesem Beispiel nur nicht überführbare Markierungen von N_1 , siehe Tabelle 5.12. Erreicht das verbundene Netz $N_1 \oplus \mathcal{T} \oplus N_2$ bei der Interaktion mit einem Service-

$[p_\alpha, p_2, p_5, Z]$	$[p_\alpha, p_4, p_5, Z]$
$[p_\alpha, p_2, p_5, Z, €]$	$[p_\alpha, p_4, p_5]$
$[p_\alpha, p_2, p_5, €]$	

Tabelle 5.12: nicht überführbare Markierungen

automaten R die Markierung $[p_\alpha, p_2, p_5, €]$, so kann N_1 weiterschalten, auch wenn die Überführung $[p_\alpha] \rightarrow \square \rightarrow [v_\alpha]$ bereits stattgefunden hat. Das Ausgangsnetz kann von der Markierung $[p_2, p_5, €]$ schalten nach $[p_f, G]$. Damit hat das Netz wieder eine überführbare Markierung erreicht. Die Markierung $[p_\alpha, p_2, p_5, €]$ soll daher im folgenden nicht mehr zum Abbruch beim Zerteilen der Transition führen.

Verallgemeinert soll kein Abbruch der Zerteilung erfolgen, wenn N_1 schalten kann nachdem bereits ein Teil der Marken einer nicht überführbaren Markierung durch Laufzeitersetzungstransitionen in das Zielnetz überführt worden sind. Dafür ist sicherzustellen, dass die von N_1 benötigten Eingabeplätze nicht von den Transitionen aus N_2 verarbeitet werden.

Mit einer Modifikation des Zerteilungsalgorithmus lässt sich dieses Ziel erreichen: Falls der Vorbereich einer zerteilten Transition in einer Markierung m_b in B enthalten ist, bricht der Algorithmus nicht notwendigerweise ab, sondern subtrahiert zunächst den Vorbereich der Transition von m_b . Dann prüft er, ob es für die Differenzmarkierung noch aktivierte Transitionen in N_1 gibt. Benötigen diese aktivierten Transitionen zum Schalten auch Eingabeplätze, so muss sichergestellt sein, dass N_2 keine Marken von

diesen Plätzen entfernt. Dazu wird bestimmt, welche Transitionen in N_2 potentiell durch den Nachbereich der neu erzeugten Laufzeitersetzungstransition aktiviert werden. Sobald eine der potentiell aktivierten Transitionen einen für N_1 notwendigen Eingabeplatz im Vorbereich hat, wird die aktuelle Zerteilung verworfen.

Wir definieren eine Abbildung V von den Eingabeplätzen P_I auf Mengen von Plätzen in N_2 , die folgende Eigenschaft besitzt: Wenn $q \in V(p_i)$, dann gibt es entlang der Bögen in N_2 einen Pfad von q zu einer Transition, die auch p_i im Vorbereich hat. Formal sei

$$S(p_i) := \{t \mid \exists \bullet t(p_i) > 0\}$$

die Menge der Transitionen, die die Eingabe l verwenden, und

$$V(p_i) := \{p \mid \exists t \in S(p_i) : \text{es gibt einen Pfad von } p \text{ nach } t\}$$

die entsprechend gesuchte Platzmenge des Zielnetzes.

Nachstehend zeigen wir zum Vergleich den originalen Ausschnitt des Subalgorithmus und die modifizierte Algorithmusvariante.

ZERTEILUNGSSCHRITT(m)

```

:
(7)  ...
(8)  if  $\exists m_b \in B'$  mit  $m \leq m_b$ 
(9)    return false
(10) ...
:

```

ZERTEILUNGSSCHRITT*(m)

```

:
(7)  ...
(8)  foreach  $m_b \in B' : m \leq m_b$ 
(9a)    $m'_b := m_b - m$ 
(9b)   if  $m'_b$  aktiviert keine Transition in  $T_1$ 
(9c)    return false
(9d)    $L := \{p_i \in P_I \mid \exists t \in T_1 : t \text{ ist aktiviert in } m'_b \text{ und } [p_i] \leq \bullet t\}$ 
(9e)   foreach  $p_i \in L$ 
(9f)    if  $m' \cap V(p_i) \neq \emptyset$ 
(9g)    return false
(9h)    $B' := B' \cup \{(m_b - m)\}$ 
(10)  ...
:

```

Algorithmus 5.13: modifizierter Ausschnitt aus ZERTEILUNGSSCHRITT

Wir erläutern die Rechenschritte beispielhaft am Aufruf ZERTEILUNGSSCHRITT*([p_α]). Der Algorithmus prüft in einem Zerteilungsschritt die mögliche Laufzeitersetzungstransition [p_α] \rightarrow \square \rightarrow [v_α]. In Zeile 8 wird dabei die Markierung $m_b = [p_\alpha, p_2, p_5, \text{€}] \in B$ ausgewählt. Als Differenz aus m_b und $m = [p_\alpha]$ erhalten wir in Zeile 9a $m'_b = m_b - m = [p_2, p_5, \text{€}]$. Genau eine Transition von N_1 ist aktiviert in m'_b (Zeile 9b). Damit bestimmt der Algorithmus als nächstes die Menge L der verwendeten Eingabepätze zu $L = \{\text{€}\}$ (Zeile 9d).

$V(\text{€}) = \{v'_0, v_0, v_1, v_2, v_3\}$ sind diejenigen Plätze, welche zur Aktivierung der €-verarbeitenden Transition in N_2 beitragen könnten (Zeile 9e). Man sieht, dass in Zeile 9f die Schnittmenge $\{v_\alpha\} \cap \{v'_0, v_0, v_1, v_2, v_3\}$ leer ist, daher wird die Laufzeitersetzungstransition [p_α] \rightarrow \square \rightarrow [v_α] an dieser Stelle nicht verworfen (Zeile 9g).

Wir zeigen nun, dass für diesen erweiterten Algorithmus das Lemma 5.5 in ähnlicher Form zutrifft.

Lemma 5.7 *Sei \mathcal{T}_{inst} eine instantane Laufzeitersetzung von N_1 nach N_2 gemäß $OG = R^\phi$, bei der die Vorbereiche der Laufzeitersetzungstransitionen paarweise verschieden sind. Sei \mathcal{T} die Transitionsmenge, die von ZERTEILUNG($N_1, N_2, R, \mathcal{T}_{inst}$) mit dem modifiziertem Subalgorithmus ZERTEILUNGSSCHRITT* berechnet wurde. Wenn in der Komposition aus verbundenem Netz $N_1 \oplus \mathcal{T} \oplus N_2$ und R keine Transition aus \mathcal{T} schalten kann, obwohl N_1 noch nicht leer ist, dann kann eine Transition aus N_1 schalten.*

Beweis:

Angenommen, N_1 ist nicht leer und keine Transition aus N_1 oder \mathcal{T} kann schalten.

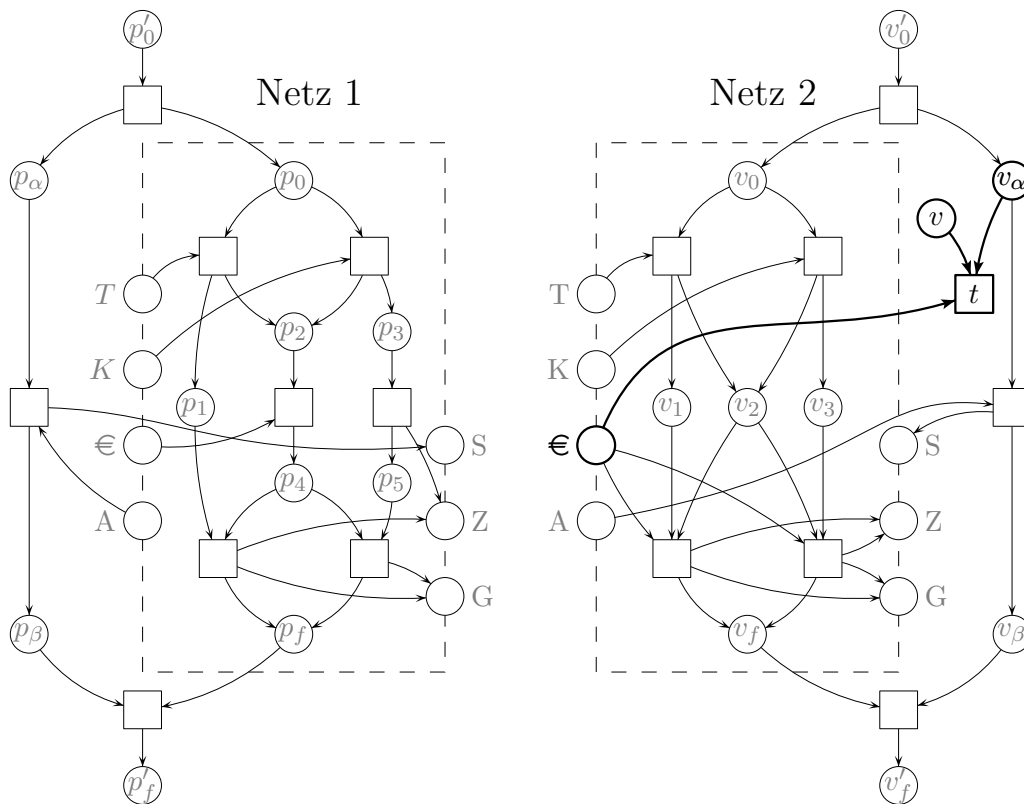
Wir können die Argumentation aus dem Beweis zu Lemma 5.5 an dieser Stelle zum großen Teil übernehmen. So ist nur noch der Fall zu betrachten, dass bereits die Laufzeitersetzungstransitionen t_1, \dots, t_n geschaltet haben und die aktuelle Markierung m_{akt} von N_1 nicht Vorbereich einer instantanen Transition aus \mathcal{T}_{inst} ist.

In diesem Fall ist $m := m_{akt} + \sum_i \bullet t_i$ eine nicht überführte Markierung. Nach der Initialisierung der Menge B im Algorithmus ZERTEILUNG gilt $m \in B$. Bei der Erzeugung der Transitionen t_i ist jeweils die Differenz aus den Markierungen von B und dem Vorbereich $\bullet t_i$ zu B hinzugenommen worden (Zeile 9h). Somit gilt $m_{akt} \in B$.

Gemäß Zeile 9b und 9c des Algorithmus, gibt es aktivierte Transitionen in m_{akt} . Wegen der Überprüfung in den Zeilen 9d bis 9g ist im Zielnetz keine Transition aktiviert, die den Vorbereich mit den im Ausgangsnetz aktivierten Transitionen teilt. Somit kann eine der aktivierten Transitionen von N_1 schalten. \square

Die angegebenen Modifikationen des Zerteilungsalgorithmus arbeiten, wie wir gesehen haben, korrekt, sind jedoch nicht vollständig. So kann es ebenso wie im ursprünglichen Subalgorithmus ZERTEILUNGSSCHRITT passieren, dass Laufzeitersetzungstransitionen ausgeschlossen werden, welche in der Komposition tatsächlich gar keinen Deadlock verursachen.

Ein derartiges Szenario zeigt die Abbildung 5.14. Hier gibt es eine Transition t , in deren Vorbereich ein niemals markierter Platz liegt. Bei der heuristischen Vorgehensweise wird durch die Überprüfung in Zeile 9f der Subalgorithmus ZERTEILUNGSSCHRITT* abbrechen, obwohl die Transition t tatsächlich nie eine Marke vom Eingabepplatz ϵ entfernt.



Anfangsmarkierungen sind $[p'_0]$ bzw. $[v'_0]$, Endmarkierungen sind $[p'_f]$ bzw. $[v'_f]$

Abbildung 5.14: Problembeispiel für Heuristiken

Dieses Beispiel zeigt, dass der vorgeschlagene Algorithmus lediglich heuristischen Charakter hat. Als Daumenregel lässt sich aber festhalten, dass für Transitionspfade, die nebenläufig im Zielnetz sind, der modifizierte Algorithmus diese Nebenläufigkeiten auch findet.

5.3.2 Verkleinerung der Menge B

Im oben beschriebenen Beispiel von Abbildung 5.10 (Seite 59) wird die Zerteilung der instantanen Laufzeitersetzungstransitionen in die Teiltransition $[p_\alpha] \rightarrow \square \rightarrow [v_\alpha]$ beim Aufruf von `ZERTEILUNGSSCHRITT`($[p_\alpha]$) abgebrochen. Der Grund ist, dass N_1 in der nicht überführbaren Markierung $[p_\alpha, p_2, p_5, Z]$ nicht schalten kann. Allerdings kann der Automat R aus Abbildung 5.11 bei dieser Markierung des Ausgangsnetzes noch die Nachricht € senden und anschließend N_1 kann durchaus weiterschalten.

Die genannte Markierung soll daher im folgenden nicht mehr Abbruchgrund beim Zerteilen der Transition sein. Ein Nachfolgezustand von $([p_\alpha, p_2, p_5, Z], q_1)$ in der Komposition ist $([p_\alpha, p_2, p_5, Z, \text{€}], q_2)$, d.h. der Partner hat die Eingabe € vorgenommen. Verarbeitet er noch die Ausgabe Z , so erhalten wir die Markierung $m = [p_\alpha, p_2, p_5, \text{€}]$. Im Aufruf `ZERTEILUNGSSCHRITT`($[p_\alpha]$) wurde $[p_\alpha, p_2, p_5, Z]$, $[p_\alpha, p_2, p_5, Z, \text{€}]$ und $[p_\alpha, p_2, p_5, \text{€}]$ geprüft. Stattdessen soll nur noch für $[p_\alpha, p_2, p_5, \text{€}]$ geprüft werden, ob diese Markierung einen Deadlock in der Komposition ergibt.

Allgemein ist in der Komposition kein Deadlock erreicht, solange der Partner schalten kann. Wir ergänzen den Zerteilungsalgorithmus dahingehend, dass entsprechende Markierungen aus B entfernt werden. Ob der Partner R bei einer bestimmten Markierung m_b des Ausgangsnetzes schalten kann, wird anhand der Nachfolger derjenigen Zustände entschieden, für die diese Markierung erreichbar ist ($m_b \in K_1(q)$). Damit R über eine Kante (q, l, q') nach q' wechseln kann, muss – falls l ein Ausgabeplatz des Netzes ist – l auch markiert sein in m_b .

Nachstehend zeigen wir zum Vergleich den originalen Ausschnitt des Algorithmus `ZERTEILUNG`. Die Erweiterung dieser Initialisierung der Menge B ist auf der folgenden Seite algorithmisch dargestellt.

```
ZERTEILUNG( $N_1, N_2, R, \mathcal{T}$ )
(1)   $B := \{m_b \mid m_b \text{ wird nicht überführt in } \mathcal{T}\}$ 
(2)  ...
    ⋮
```

Anhand einiger Beispielaufufe von `PARTNERKANNSCHALTEN`(m_b) für verschiedene Markierungen m_b verdeutlichen wir die Vorgehensweise des modifizierten Algorithmus. Beim Aufruf `PARTNERKANNSCHALTEN`($[p_\alpha, p_2, p_5, Z]$) existiert in Zeile i nur einen Zustand, nämlich q_1 , für den die Markierung m_b erreichbar ist. Im Automaten R gibt es einen Nachfolger von q_1 , die Kante ist mit dem Eingabeplatz € beschriftet. Somit liefert dieser Aufruf als Rückgabewert **true** und die Markierung $[p_\alpha, p_2, p_5, Z]$ wird aus der Menge B gelöscht.

```

ZERTEILUNG*( $N_1, N_2, R, \mathcal{T}$ )
(1a)  $B := \{m_b \mid m_b \text{ wird nicht überführt in } \mathcal{T}\}$ 
(1b) foreach  $m_b \in B$ 
(1c)     if PARTNERKANNSCHALTEN( $m_b$ )
(1d)          $B := B \setminus \{m_b\}$ 
(2)     ...
      ...
      ...

```

```

PARTNERKANNSCHALTEN( $m_b$ )
(i)   foreach  $q \in Q$  mit  $m_b \in K_1(q)$ 
(ii)   if  $\nexists q' \in Q$  mit  $(q, l, q') \in \delta$ 
(iii)   return false
(iv)   foreach  $q' \in Q$  mit  $(q, l, q') \in \delta$ 
(v)     if  $l \in P_O$  und  $m_b(l) = 0$ 
(vi)     return false
(vii)  return true

```

Algorithmus 5.15: modifizierter Ausschnitt aus ZERTEILUNG

Beim Aufruf PARTNERKANNSCHALTEN($[p_\alpha, p_2, p_5, Z, \text{€}]$) gibt es ebenfalls genau einen Zustand, $q = q_2$, auf den die Bedingung $m_b \in K_1(q)$ zutrifft. Der Nachfolger von q_2 ist über eine Kante mit der Beschriftung Z erreichbar. Diese Nachricht gehört zu den Ausgabekanälen P_O . Wegen $m_b(Z) = 1$ kann die Komposition im Zustand ($[p_\alpha, p_2, p_5, Z, \text{€}], q_2$) weiterschalten. Der Algorithmus liefert daher **true**, sodass $[p_\alpha, p_2, p_5, Z, \text{€}]$ ebenfalls aus B entfernt wird.

Beim dritten Beispielaufruf, PARTNERKANNSCHALTEN($[p_\alpha, p_2, p_5, \text{€}]$), enthält die Erreichbarkeitsfunktion von Zustand q_3 die Markierung m_b . Allerdings ist der Ausgabeplatz G , mit der die Kante zum einzigen Nachfolger von q_3 beschriftet ist, in m_b nicht markiert. Der Algorithmus liefert **false** zurück und $[p_\alpha, p_2, p_5, \text{€}]$ bleibt in der Menge B enthalten.

Mit dem modifizierten Algorithmus erreichen wir die angestrebte Lösung für das Beispiel aus Abbildung 5.10 (Seite 59). Verwendet man für den Zerteilungsalgorithmus sowohl die veränderte Menge B als auch die angepasste Deadlockprüfung, so erhält man eine „schöne“ nebenläufige Laufzeitersetzung, welche die Überführungen $[p_\alpha] \rightarrow \square \rightarrow [v_\alpha]$ und $[p_\beta] \rightarrow \square \rightarrow [v_\beta]$ einschließt.

Die Korrektheit des Zerteilungsalgorithmus bei dieser Modifikation der Menge B ist durch folgendes Lemma gegeben.

Lemma 5.8 *Sei \mathcal{T}_{inst} eine instantane Laufzeitersetzung von N_1 nach N_2 gemäß $OG = R^\phi$, bei der die Vorbereiche der Laufzeitersetzungstransitionen paarweise verschieden sind. Sei \mathcal{T} die Transitionsmenge, die vom modifiziertem Algorithmus ZERTEILUNG* ($N_1, N_2, R, \mathcal{T}_{inst}$) mit dem modifiziertem Subalgorithmus ZERTEILUNGSSCHRITT* berechnet wurde. Wenn in der Komposition aus verbundenen Netz $N_1 \oplus \mathcal{T} \oplus N_2$ und R keine Transition aus \mathcal{T} schalten kann, obwohl N_1 noch nicht leer ist, dann kann eine Transition aus N_1 schalten oder R kann Nachrichten empfangen/senden.*

Beweis:

Angenommen, N_1 ist nicht leer und keine Transition aus N_1 oder \mathcal{T} kann schalten.

Wir können die Argumentation aus dem Beweis zu Lemma 5.7 an dieser Stelle zum großen Teil übernehmen. So ist nur noch der Fall zu betrachten, dass bereits die Laufzeitersetzungstransitionen t_1, \dots, t_n geschaltet haben und die aktuelle Markierung m_{akt} von N_1 nicht Vorbereich einer instantanen Transition aus \mathcal{T}_{inst} ist.

In diesem Fall ist $m := m_{akt} + \sum_i \bullet t_i$ eine nicht überführte Markierung. Ist nun $m_{akt} \in B$ so gilt die gesuchte Aussage analog zu Lemma 5.7.

Anderenfalls hat der Aufruf PARTNERKANNSCHALTEN(m) **true** zurückgegeben, sodass die Markierung m aus B entfernt worden ist. Der Partner R besitzt also von seinem aktuellen Zustand aus Nachfolgezustände (Zeilen i–iii). Die Überführung zu den Nachfolgezuständen ist sichergesellt, weil die benötigten Nachrichten (Ausgabekanäle des offenen Workflownetzes) vorhanden sind. Dies ist durch die Überprüfung in den Zeilen iv–vii sichergestellt.

Wenn N_1 nicht leer ist und keine Transition aus \mathcal{T} aktiviert ist, kann somit eine Transition aus N_1 schalten oder R kann Nachrichten empfangen/senden. \square

Der Korrektheitsbeweis des Gesamtalgorithmus, Satz 5.6, behält für die beiden vorgestellten Heuristiken seine Gültigkeit. In der Argumentation des Beweises kann das Lemma 5.5 durch das Lemma 5.7 und Lemma 5.8 ersetzt werden.

5.4 Größenreduktion durch nebenläufige Laufzeitersetzungen

Im diesem Kapitel haben wir einen Algorithmus angegeben, der die Laufzeitersetzungstransitionen einer instantanen Laufzeitersetzung durch iteratives Zerteilen in kleinere, nebenläufige Transitionen trennt. Zwei heuristische Ansätze helfen, die Transitionen

noch besser aufteilen zu können. Im letzten Abschnitt erläutern wir nun den Vorteil, der sich aus der Anwendung dieser Algorithmen ergibt.

Wir zeigen, dass die erzeugte nebenläufige Laufzeitersetzung \mathcal{T} wesentlich kleiner ist als die instantane Laufzeitersetzung \mathcal{T}_{inst} , die der Zerteilungsalgorithmus als Eingabe erhält. Die möglichen Ersetzungsschritte gehen dabei nicht verloren – jede Laufzeitersetzungstransition von \mathcal{T}_{inst} lässt sich als Summe von Transitionen aus \mathcal{T} darstellen.

Abbildung 5.16 zeigt ein offenes Workflownetz, bei dem die Laufzeitersetzung mit einer Kopie von sich durch exponentiell viele instantane Laufzeitersetzungstransitionen möglich ist, in Bezug auf die Größe des Netzes. Als Strategien sind alle Strategien des Netzes zugelassen.

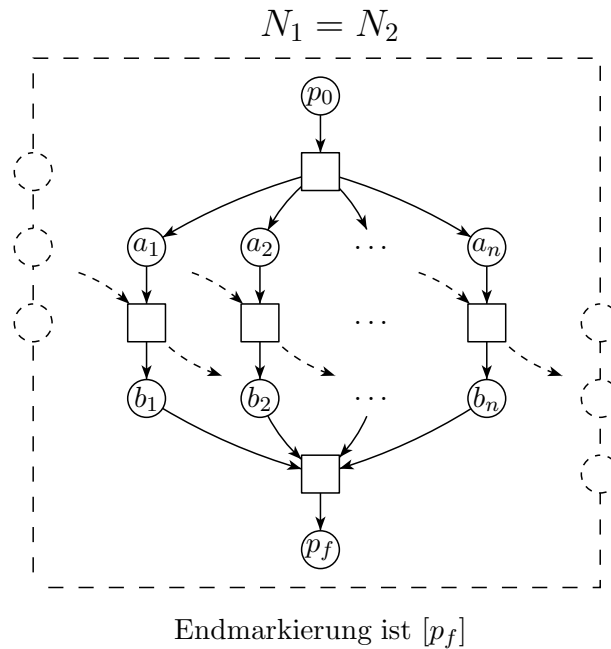


Abbildung 5.16: exponentiell viele instantane Laufzeitersetzungstransitionen

Dieses Netz enthält $2n + 2$ innere Plätze und $n + 2$ Transitionen. Die gestrichelten Pfeile in der Abbildung deuten an, dass n nebenläufigen Transitionen verschiedene Eingaben verarbeiten und Ausgaben erzeugen. Es gibt offenbar mindestens $2^n + 2$ erreichbare Markierungen. Jede dieser Markierungen von N_1 kann zur Laufzeit instantan in die gleiche Markierung in N_2 überführt werden. Damit enthält die vollständige instantane Laufzeitersetzung \mathcal{T}_{inst} mindestens $2^n + 2$ Transitionen.

Es gibt für dieses Beispiel eine offensichtlich viel kleinere nebenläufige Laufzeitersetzung \mathcal{T} . Sie entspricht der Identität der inneren Plätze, das bedeutet, für jeden inneren Platz von N_1 gibt es eine Transition in den gleichen Platz des Zielnetzes. Alle instantanen Lauf-

5 Nebenläufige Laufzeitersetzungen

zeitersetzungstransitionen können als Summe der entsprechenden nebenläufigen Transitionen dargestellt werden. Die Größe von \mathcal{T} ist so mit $2n + 2$ Transitionen exponentiell kleiner als \mathcal{T}_{inst} .

Wesentlich für die Größenreduktion ist am Ende des Zerteilungsalgorithmus das Löschen all jener Laufzeitersetzungstransitionen, die als Summe anderer dargestellt werden können. Anderenfalls wäre das Ergebnis noch größer als die ursprüngliche instantane Laufzeitersetzung.

Größenreduktion tritt nicht nur bei der Ersetzung identischer Netze auf. Im Abschnitt 5.1 (Seite 48ff) haben wir die Laufzeitersetzung der Netze zweier Getränkeautomaten (Abbildung 5.3) betrachtet. Als zu bearbeitende, instantane Laufzeitersetzung wurde eine Transitionsmenge mit 11 Transitionen festgelegt (Tabelle 5.4). Durch die Anwendung unseres Algorithmus konnte die Zahl auf 8 nebenläufige Transitionen reduziert werden (Tabelle 5.6).

6 Fazit

In dieser Arbeit wurde ein theoretisches Konzept für die *Laufzeitersetzung* von Geschäftsprozessen in Form offener Workflownetze eingeführt. Übertragen auf reale Anwendungen lassen sich mithilfe der Laufzeitersetzungen aktive Geschäftsprozesse in andere (neue) Geschäftsprozesse überführen. Auf diese Weise könnten zum Beispiel laufende Verträge einer Firma verkauft werden an eine andere Firma, wobei die Interaktion mit den Kunden schrittweise übergeben wird.

Unser Konzept sieht die freie Wahl der zugrundeliegenden Strategien für die beteiligten Kommunikationspartner vor. Damit lassen sich für verschiedene Kundengruppen individuelle Laufzeitersetzungen berechnen.

Auf der Modellebene erweitert die Laufzeitersetzung den Begriff der Austauschbarkeit von Geschäftsprozessen, welcher in verschiedenen Arbeiten bereits untersucht worden ist. Die Laufzeitersetzung zweier offener Workflownetze ist definiert als eine Menge von Transitionen, die zur Laufzeit Marken vom Ausgangs- in das Zielnetz überführen. Ziel dieser Ersetzung ist, dass ein Serviceautomat, welcher als Interaktionspartner für das Ausgangsnetz dient, ebenso mit dem *verbundenen Netz* aus Ausgangsnetz, Zielnetz und Laufzeitersetzung kommunizieren kann. Die Startmarkierung des verbundenen Netzes liegt im Ausgangsnetz, während die Endmarkierung nur Plätze des Zielnetzes belegt. Bei der Laufzeitersetzung ist garantiert, dass das verbundene Netz mit dem Partner deadlockfrei interagieren kann. Auf diese Weise wird der durch das Ausgangsnetz beschriebene Geschäftsprozess zur Laufzeit durch den Geschäftsprozess des Zielnetzes ersetzt.

Voraussetzung für die Erzeugung von Laufzeitersetzungen ist, dass es sich um *beschränkte Netze* handelt sowie dass die Interaktion mit den Partnern eine *beschränkte Kommunikation* darstellt. Zusätzlich wird gefordert, dass die Netze keine leeren Transitionen im inneren Netz enthalten. Als Menge der zu berücksichtigenden Strategien der Interaktionspartner darf eine beliebige Untermenge der Strategien des Ausgangsnetzes angegeben werden. Diese Strategiemenge wird dabei als annotierter Serviceautomat gegeben.

Wie in dieser Arbeit gezeigt, kann die Laufzeitersetzung *instantan*, also in einem einzigen Schritt, oder aber *nebenläufig*, in mehreren Teilschritten, erfolgen. Die grundlegende Idee der instantanen Laufzeitersetzungen für offene Workflownetze basiert auf der Lauf-

zeitersetzung von Serviceautomaten, welche in einer vorausgegangenen Studienarbeit vorgestellt worden ist. Es konnte in der vorliegenden Arbeit ein Algorithmus angegeben und in seiner Korrektheit formal bewiesen werden, welcher die eindeutig definierte, *vollständige instantane Laufzeitersetzung* zweier offener Workflownetze gemäß einer vorgebenen Menge von *Strategien* berechnet.

Der Schwerpunkt der Arbeit liegt auf der Berechnung nebenläufiger Laufzeitersetzungen. Es wurde der Algorithmus ZERTEILUNG angegeben, welcher die Transitionen einer gegebenen instantanen Laufzeitersetzung in möglichst kleine Teiltransitionen zerteilt. Wir konnten zeigen, dass die Größe einer nebenläufigen Laufzeitersetzung exponentiell kleiner sein kann als eine instantane Laufzeitersetzung, wenngleich sich alle Transitionen der instantanen Laufzeitersetzung als Summe nebenläufiger Laufzeitersetzungstransitionen darstellen lassen. Dieser Zerteilungsalgorithmus sowie zwei modifizierte Varianten davon wurden formal in ihrer Korrektheit bewiesen. Die Modifikationen beschreiben heuristische Ansätze, welche der Berechnung noch „schönerer“ Ergebnisse dienen.

Mit dem in Kapitel 4 vorgestellten Algorithmus INSTANTANELE lässt sich stets die vollständige instantane Laufzeitersetzung finden. Dabei kann, wie wir gesehen haben, die Anzahl der Laufzeitersetzungstransitionen exponentiell groß in Bezug auf die beteiligten Netze sein. Es ist durchaus denkbar, dass ganz ein anderer Ansatz – der nicht auf den instantanen Überführungen basiert – zur Erzeugung von Laufzeitersetzungen verwendet werden kann.

Die heuristischen Ansätze in Abschnitt 5.3 zur Verbesserung der nebenläufigen Laufzeitersetzungen bieten Raum für weitere Untersuchungen. Die Frage, inwiefern sich die Reihenfolge der Zerteilungsschritte im Algorithmus ZERTEILUNG auf die Größe der erzeugten, nebenläufigen Laufzeitersetzung auswirkt, kann ebenfalls Gegenstand weiterer Arbeiten auf diesem Gebiet sein.

Literaturverzeichnis

- [AAA⁺06] Alexandre Alves, Assaf Arkin, Sid Askary, Ben Bloch, Francisco Curbera, Yaron Goland, Neelakantan Kartha, Sterling, Dieter König, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, and Alex Yiu. Web services business process execution language version 2.0. OASIS Committee Draft, May 2006.
- [Aal98] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
- [BK06] F. Breugel and M. Koshkina. Models and verification of BPEL. <http://www.cse.yorku.ca/franck/research/drafts/tutorial.pdf>, 2006.
- [GGKS02] Karl D. Gottschalk, Stephen Graham, Heather Kreger, and James Snell. Introduction to web services architecture. *IBM Systems Journal*, 41(2):170–177, 2002.
- [Gla01] R. van Glabbeek. The linear time – branching time spectrum i: The semantics of concrete, 2001.
- [HDvdA⁺05] Jan Hidders, Marlon Dumas, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Jan Verelst. When are two workflows the same? In *CATS '05: Proceedings of the 2005 Australasian symposium on Theory of computing*, pages 3–11, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.
- [HSS05] Sebastian Hinz, Karsten Schmidt, and Christian Stahl. Transforming BPEL to Petri Nets. In Wil M. P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proceedings of the Third International Conference on Business Process Management (BPM 2005)*, volume 3649 of *Lecture Notes in Computer Science*, pages 220–235, Nancy, France, September 2005. Springer-Verlag.

- [Lis07] Nannette Liske. Laufzeitersetzbarkeit von Services. Studienarbeit, Humboldt-Universität zu Berlin, April 2007.
- [LMSW06] Niels Lohmann, Peter Massuthe, Christian Stahl, and Daniela Weinberg. Analyzing interacting BPEL processes. In *Business Process Management*, pages 17–32, 2006.
- [LMW07] Niels Lohmann, Peter Massuthe, and Karsten Wolf. Operating guidelines for finite-state services. In Jetty Kleijn and Alex Yakovlev, editors, *28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007, Siedlce, Poland, June 25-29, 2007, Proceedings*, volume 4546 of *Lecture Notes in Computer Science*, pages 321–341. Springer-Verlag, 2007.
- [Loh08] Niels Lohmann. A feature-complete Petri net semantics for WS-BPEL 2.0. In Marlon Dumas and Reiko Heckel, editors, *Web Services and Formal Methods, Forth International Workshop, WS-FM 2007 Brisbane, Australia, September 28-29, 2007, Proceedings*, volume 4937 of *Lecture Notes in Computer Science*, pages 77–91. Springer-Verlag, 2008.
- [Mar03] Axel Martens. *Verteilte Geschäftsprozesse - Modellierung und Verifikation mit Hilfe von Web Services*. Dissertation, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, 2003. erschienen in WiKi: Stuttgart, Berlin & Paris.
- [Mil71] Robin Milner. An algebraic definition of simulation between programs. Technical report, Stanford, CA, USA, 1971.
- [Mil89] Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
- [MRS05] Peter Massuthe, Wolfgang Reisig, and Karsten Schmidt. An Operating Guideline Approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics*, 1(3):35–43, 2005.
- [MS05] Peter Massuthe and Karsten Schmidt. Operating Guidelines - an Automata-Theoretic Foundation for the Service-Oriented Architecture. In Kai-Yuan Cai, Atsushi Ohnishi, and M.F. Lau, editors, *Proceedings of the Fifth International Conference on Quality Software (QSIC 2005)*, pages 452–457, Melbourne, Australia, September 2005. IEEE Computer Society.
- [Pap01] Mike P. Papazoglou. Agent-oriented technology in support of e-business. *Commun. ACM*, 44(4):71–77, 2001.

- [RCS⁺08] Seung Hwan Ryu, Fabio Casati, Halvard Skogsrud, Boualem Benatallah, and Régis Saint-Paul. Supporting the dynamic evolution of web service protocols in service-oriented architectures. *TWEB*, 2(2), 2008.
- [Rei82] Wolfgang Reisig. *Petrinetze, Eine Einführung*. Springer, 1982.
- [Ric02] Wolf Richter. Spezifikation und Implementation organisationsübergreifender Geschäftsprozesse mit Petrinetzen. Diplomarbeit, Humboldt-Universität zu Berlin, 2002.
- [Sch00] Karsten Schmidt. LoLA: A Low Level Analyser. In Mogens Nielsen and Dan Simpson, editors, *Application and Theory of Petri Nets, 21st International Conference (ICATPN 2000)*, volume 1825 of *Lecture Notes in Computer Science*, pages 465–474. Springer-Verlag, June 2000.
- [Sch05] Karsten Schmidt. Controllability of Open Workflow Nets. In Jörg Desel and Ulrich Frank, editors, *Enterprise Modelling and Information Systems Architectures*, volume P-75 of *Lecture Notes in Informatics (LNI)*, pages 236–249, Bonn, 2005. Entwicklungsmethoden für Informationssysteme und deren Anwendung (EMISA, RWTH Aachen), Köllen Druck+Verlag GmbH.
- [SMB08] Christian Stahl, Peter Massuthe, and Jan Bretschneider. Deciding substitutability of services with operating guidelines. Informatik-Berichte 222, Humboldt-Universität zu Berlin, April 2008.