Fachbereich der Universität Hamburg

Vogt-Kölln-Str. 30 ◇ D-22527 Hamburg / Germany

University of Hamburg - Computer Sience Department

Bereicht Nr. 196 • Report No. 196

Ereignisgesteuerte Prozeßketten und Petri-Netze

Peter Langner, Christoph Schneider, Joachim Wehler

FBI-HH-B-196/97

März 1997

In die Reihe der Berichte des Fachbereichs Informatik aufgenommen durch

Accepted for Publicationin the Report Series of the Department of Computer Sience by

Prof. Dr. Rüdiger Valk und Prof. Dr. Matthias Jantzen

„Die EPK-Methode basiert im wesentlichen auf der Petri-Netz-Theorie (...) und kann als eine Variante des Bedingungs-Ereignisnetzes, welche um logische Verknüpfungsoperationen erweitert wurde, verstanden werden.“ [SNZ95, S. 428]

Ereignisgesteuerte Prozeßketten und Petri-Netze

Peter Langner, Innobis GmbH
Christoph Schneider, Datenerfassung
Joachim Wehler, Softlab GmbH

Peter Langner
INNOBIS Unternehmensberatung und Software GmbH
Willhoop 7
22453 Hamburg
email: P.Langner@innobis.de

Christoph Schneider
Datenerfassung
Paul Robeson Straße 40
10439 Berlin
email: 100 422.133@compuserve.com

Joachim Wehler
Softlab GmbH
Zarndorfer Straße 120
81677 München
email: WEJ@softlab.de

Ereignisgesteuerte Prozeßketten und Petri-Netze

Stichworte: Geschäftsprozeßmodellierung, Business Reengineering, Ereignisgesteuerte Prozeßkette, Gefärbtes Petri-Netz, Bipolares Synchronisationsschema, Verifikation.

Zusammenfassung: Die Wirtschaftinformatik kennt verschiedene Methoden zur Modellierung von Geschäftsprozessen. In kommerziellen Projekten werden hierfür am häufigsten Petri-Netze bzw. ereignisgesteuerte Prozeßketten (EPKs) eingesetzt. In dieser Arbeit wird die Syntax der EPKs festgelegt, und durch Übersetzung in ein höheres Petri-Netz erhalten EPKs auch eine formale Semantik. Die resultierenden Booleschen Netze sind eine Erweiterung von Stellen/Transitions-Netzen um logische Verknüpfungen und bilden eine einfache Teilklasse aller gefärbten Petri-Netze. Boolesche Netze können mit Algorithmen der Graphentheorie und zusätzlich mit Methoden der Kommutativen Algebra analysiert werden. Durch den Reduktionsalgorithmus von Genrich-Thiagarajan läßt sich das korrekte Verhalten von EPKs nachweisen, ohne hierfür ihren Fallgraphen konstruieren zu müssen. Wohlgeformte EPKs stellen sich als ein Baum von bipolaren Synchronisationsschemata heraus und lassen sich zu wohlgeformten Free-Choice Netzen vereinfachen. Damit bietet die Petri-Netztheorie in ihrer klassischen Form ein tragfähiges Fundament zur Verifikation, Animation uns Simulation von EPKs aus dem Bereich der Wirtschaftsinformatik.

Event-driven process chains and Petri nets

Keywords: Process modelling, Business Reengineering, Event-driven process chain, Coloured Petri Net, Bipolar synchronization scheme, Verification.

Abstract: In the domain of Managemant Information Systems there exists different methods to model business processes. Most widely used are Petri nets and event-driven process chains (EPCs).
The paper defines the syntax of EPCs. After translation into Petri nets EPCs acquire a formal semantic. The resulting Boolean nets generalize place/transition nets by adding logical rules. They consitute a simple subclass of coloured Petri nets. Boolean nets can be analysed by algorithms from graph theory as well as by methods from commutative algebra. Therefore well behavedness of EPCs can be proven by the reduction algorithm of Genrich-Thiagarajan, there is no need to unfold the case graph. Well behaved EPCs form a tree of bipolar synchronization schemes. They can be reduced to well behaved free-choice nets. Hence Petri nets theory in its classical form provides a well founded base to verify, animate and to simulate EPCs used in the domain of Management Information Systems.

Inhaltsverzeichnis

Einleitung

Was ist Grundlage der Unternehmensmodellierung? — „Das Datenmodell“, so die Mehrheit der Systemanalytiker seit der Erfindung des relationalen Modells in den 70-Jahren. Andererseits hört man immer häufiger die Anforderung: „Optimierung der Geschäftsprozesse, das ist unser Ziel!“
Wir meinen, beide Antworten sind richtig, man darf sie nur nicht gegeneinander ausspielen. Die Nützlichkeit der Datenmodellierung in Form des Entity-Relationship Modells — und das ist der statische Aspekt eines Unternehmens — ist unbestritten und zeigt sich seit vielen Jahren in jedem Projekt. Auf seiten der Prozeßmodellierung — und jetzt geht es um den dynamischen Aspekt des Unternehmensmodells — gibt es dagegen noch keinen Standard. Hier werden eine Vielzahl von Methoden eingesetzt. Alle basieren auf gerichteten Graphen, danach aber schwört jeder Hersteller auf seine eigene, tool-spezifische Art, das Zusammenspiel der betrieblichen Tätigkeiten abzubilden.
In Deutschland dominieren auf dem Feld betriebswirtschaftlicher Standardsoftware zur Zeit die Systeme R/2 und R/3 der Firma SAP AG. Wenn man ein SAP-System bei einem Kunden einführt, so liegt das Schwergewicht in der Phase „Fachkonzept“ eindeutig auf der Modellierung der Geschäftsprozesse, nicht auf der Datenmodellierung. Diese Betonung der Geschäftsprozesse wird unterstützt durch eine Modellierungsmethode, die ihre Erfinder „Ereignisgesteuerte Prozeßkette (EPK)“ nennen. Diese Methode ist sowohl in dem SAP-eigenen Analyse-Tool, dem R/3-Analyser, implementiert, als auch in einem separaten Tool der IDS Prof. Scheer GmbH, dem ARIS-Toolset.
Wir halten die Verwendung von EPKs für eine nützliche Methode, um die dynamische Sicht eines Kundenunternehmens zu modellieren. Ähnlich wie ein Entity-Relationship Modell sind auch EPKs für den Kunden intuitiv verständlich. Sie werden inzwischen in verschiedenen Projekten zur Unternehmensmodellierung eingesetzt und haben sich in der praktischen Arbeit mit den Mitarbeitern der Fachabteilung bewährt. Was dieser Methode jedoch fehlt, das ist das theoretische Fundament: EPKs haben weder eine explizit definierte Syntax, noch eine explizite Semantik.
Auf der anderen Seite zeichnen sich Petri-Netze unter allen Konkurrenten um die Prozeßmodellierung durch genau diese beiden Eigenschaften aus. Seit ihrer Erfindung vor über 30 Jahren wird kontinuierlich an einer mathematischen Theorie der Petri-Netze gearbeitet.
Es ist vielleicht hilfreich, hier noch einmal die Parallele zur Datenmodellierung heranzuziehen: Jedes erfolgreiche Software-Projekt setzt heute zur Datenmodellierung die Entity-Relationship Methode ein. Worauf beruht diese Akzeptanz? Wir meinen, der Erfolg dieser Methode ist nicht allein das Verdienst des Marketings relationaler Datenbanken. Vielmehr handelt es sich um den Glücksfall, daß eine praktikable Methode für den Projektalltag zugleich eine mathematische Fundierung, hier in Form der Relationenalgebra, besitzt. Die Formalisierung seines Vorgehens zwingt den Modellierer immer, Zweideutigkeiten und Unklarheiten bei der Spezifikation aufzuklären: Über die Interpretation eines Entity-Relationship Modells kann man nicht mehr diskutieren - die Semantik liegt fest.

Petri-Netze besitzen gleichfalls eine mathematische Fundierung. Und auch bei Petri-Netzen kann man nicht mehr über die Semantik des Modells diskutieren. Allerdings ist die Netzanalyse ein wesentlich schwierigeres Unterfangen als die Analyse eines Datenmodells.

Leider ist der Einsatz eines Petri-Netz Tools im Alltag eines kommerziellen Projektes die Ausnahme. Jedoch wird ARIS als Modellierungs-Tool für EPKs in immer mehr Projekten im Bereich der Wirtschaftsinformatik eingesetzt. Unser Ziel ist es daher, Regeln zur Übersetzung einer EPK in ein Petri-Netz zu formulieren. Anhand dieser Regeln kann die Übersetzung dann automatisch ablaufen. Und sobald EPKs in die Theorie der Petri-Netze eingebettet sind, hat man alle Voraussetzungen für weitergehende theoretische Analysen und praktische Simulationen von EPKs erfüllt. Somit ist dieser Schritt der Übersetzung insbesondere eine notwendige Voraussetzung, um EPKs zur Steuerung eines Workflow-Systems heranziehen zu können.

Kapitel 1 handelt von EPKs als Mittel zur Prozeßmodellierung. Wir versuchen, aus verschiedenen Arbeiten ihrer Urheber die Semantik einer EPK zu rekonstruieren — sie steckt vor allem in den logischen Konnektoren. Außerdem geben wir für die Syntax einer EPK eine präzise Definition. Schließlich schlagen wir eine konsequent objektorientierte Beschriftung der Ereignisse und Funktionen einer EPK vor.

Kapitel 2 rekapituliert die Eigenschaften von gefärbten Petri-Netzen, soweit sie für diese Arbeit gebraucht werden. Wir führen eine Teilklasse besonders einfacher gefärbter Netze ein, die wir Boolesche Netze nennen. Ihre Schaltregeln sind Formeln der Aussagenlogik, die den Fluß von Marken mit der Aufschrift „0“ oder „1“ steuern.

Kapitel 3 enthält unsere Regeln zur Übersetzung einer EPK in ein Boolesches Netz. Durch diese Übersetzung wird zugleich die Semantik einer EPK fixiert. Wir vergleichen unseren Übersetzungsvorschlag mit zwei anderen Ansätzen aus der Literatur.

In Kapitel 4 formulieren wir die Schaltregeln der Booleschen Netze als Aussagen über Polynome, indem wir die Darstellung der zweiwertigen Aussagenlogik als Algebra über dem Körper mit zwei Elementen benutzen. Wir behandeln auf dieser Basis zwei Fusionsregeln für Boolesche Netze und charakterisieren diejenigen Booleschen Transitionen, die aus der Übersetzung der logischen Konnektoren einer EPK resultieren.

In Kapitel 5 studieren wir die Netzklasse der Booleschen Schleifenbäume. Diese Netzklasse enthält die Petri-Netze, die aus der Modellierung von EPKs mit inneren Schleifen entstehen. Schleifenbäume sind eine Verallgemeinerung gewisser T-Netze und erlauben als spezielle Free-Choice Netze eine explizite Charakterisierung ihrer minimalen Siphons, Fallen und S-Komponenten.

Kapitel 6 behandelt die Modellierung von *or*-Alternativen. Es werden nur solche or-Alternativen zugelassen, die sich auf eine Schachtelung von *xor*- und *and*-Alternativen zurückführen lassen. Damit ist jedes Petri-Netz, das aus der Übersetzung einer EPK entsteht, ein Baum von bipolaren Synchronisationsschemata. Die Klasse der bipolaren Synchronisationsschemata wurde von Genrich und Thiagarajan eingeführt und studiert.

Im abschließenden Kapitel 7 präzisieren wir diejenigen Eigenschaften einer EPK, die durch eine Netzanalyse geprüft werden sollen. Bei positivem Ausgang der statischen Analyse erhalten wir ein Boolesches Netzsystem, das in der nachfolgenden dynamischen Analyse darauf geprüft wird, ob es von ordentlichem Verhalten ist. Diese Prüfung kann mit einem Algorithmus von Genrich und Thiagarajan für bipolare Synchronisationsschemata ebenfalls in Form einer statischen Analyse durchgeführt werden. Nach erfolgreicher Prüfung kann das Boolesche Netz in ein Free-Choice Netz übersetzt werden, und die Marken mit der Aufschrift „0“ fallen weg.

Für die gesamte Arbeit legen wir als durchgängiges Beispiel eine EPK der Beschaffungslogistik aus der Literatur zugrunde, an der wir alle Schritte erläutern.

Wir danken Herrn H. Genrich für einige hilfreiche Hinweise im Zusammenhang mit

bipolaren Synchronisationsschemata und Herrn J. Desel für seine Anmerkungen und Verbesserungsvorschläge zum Manuskript.
Wir hatten Gelegenheit, die Resultate der vorliegenden Arbeit auf einem Treffen der GI-Arbeitsgruppe „Petrinetze und Informationssysteme in der Praxis“ vorzustellen. Wir danken allen Teilnehmern für ihre Anregungen und Kritik. Ein Teilnehmer prägte dabei folgendes Bild für unser Anliegen, EPKs in Petri-Netze zu übersetzen: *Vergleicht man die Prozeßmodellierung mit dem Bau eines Hauses, so ist die EPK-Methode eine Benutzeroberfläche, damit der Bauherr dem Architekten seine Wünsche mitteilen kann. Petri-Netze haben dagegen die Funktion des Statikers, welcher die Konstruktion auf ihre Tragfähigkeit prüft.*

Kapitel 1

Ereignisgesteuerte Prozeßketten

Ereignisgesteuerte Prozeßketten (EPKs) sind eine Methode der Prozeßmodellierung. Sie werden in kommerziellen Projekten aus dem Bereich der Wirtschaftsinformatik eingesetzt. EPKs haben den großen Vorzug, auch für die Projektpartner auf seiten des Kunden intuitiv verständlich zu sein.
Die Abbildung 1.1 „Ereignisgesteuerte Prozeßkette der Beschaffungslogistik“ stammt aus dem Lehrbuch „Wirtschaftsinformatik“ [Sch94] von A.-W. Scheer. Scheer, Keller und Nüttgens haben in [KNS91] die Prozeßmodellierung mit EPKs erfunden. Und zwar verwenden sie EPKs, um den *Kontroll*fluß zu modellieren, nicht den *Daten*fluß.

Bemerkung 1.1 Eine EPK ist ein gerichteter Graph mit vier Arten von Knoten: Funktionen, Prozeßwegweisern, Ereignissen und logischen Konnektoren.

1. Funktionen sind die aktiven Elemente des Graphen. Sie modellieren Aufgaben oder Tätigkeiten. Funktionen erzeugen als Resultat ihrer Bearbeitung die Ereignisse. Falls verschiedene Ereignisse alternativ eintreten, kann die Auswahl des jeweiligen Ereignisses explizit als eigene Entscheidungsfunktion modelliert werden. Leitender Gesichtspunkt bei der Modellierung einer Funktion im Rahmen einer EPK ist es nicht, welche Daten die Funktion verändert, sondern wie sie den Kontrollfluß steuert.

   Beispiele für Funktionen:

   - „Lieferantenangebot einholen“
   - „Rechnung korrigieren“
   - „Auftragsfreigabe prüfen“

2. Prozeßwegweiser sind spezielle Funktionen, die als eigene Prozesse an anderer Stelle weiter verfeinert werden können. Ein Prozeßwegweiser kann in einer EPK an allen Stellen stehen, an denen auch eine Funktion stehen kann.

3. Ereignisse sind passive Elemente des Graphen. Ereignisse treffen keine Entscheidungen. (Allerdings haben sich die Autoren an diese in [KNS91, S. 14], von ihnen aufgestellte Regel selbst nicht in allen Fällen gehalten, siehe [Sch94, S. 450].)

   Beispiele für Ereignisse sind:

   - „Kundenauftrag eingetroffen“
   - „Fertigungsauftrag freigegeben“

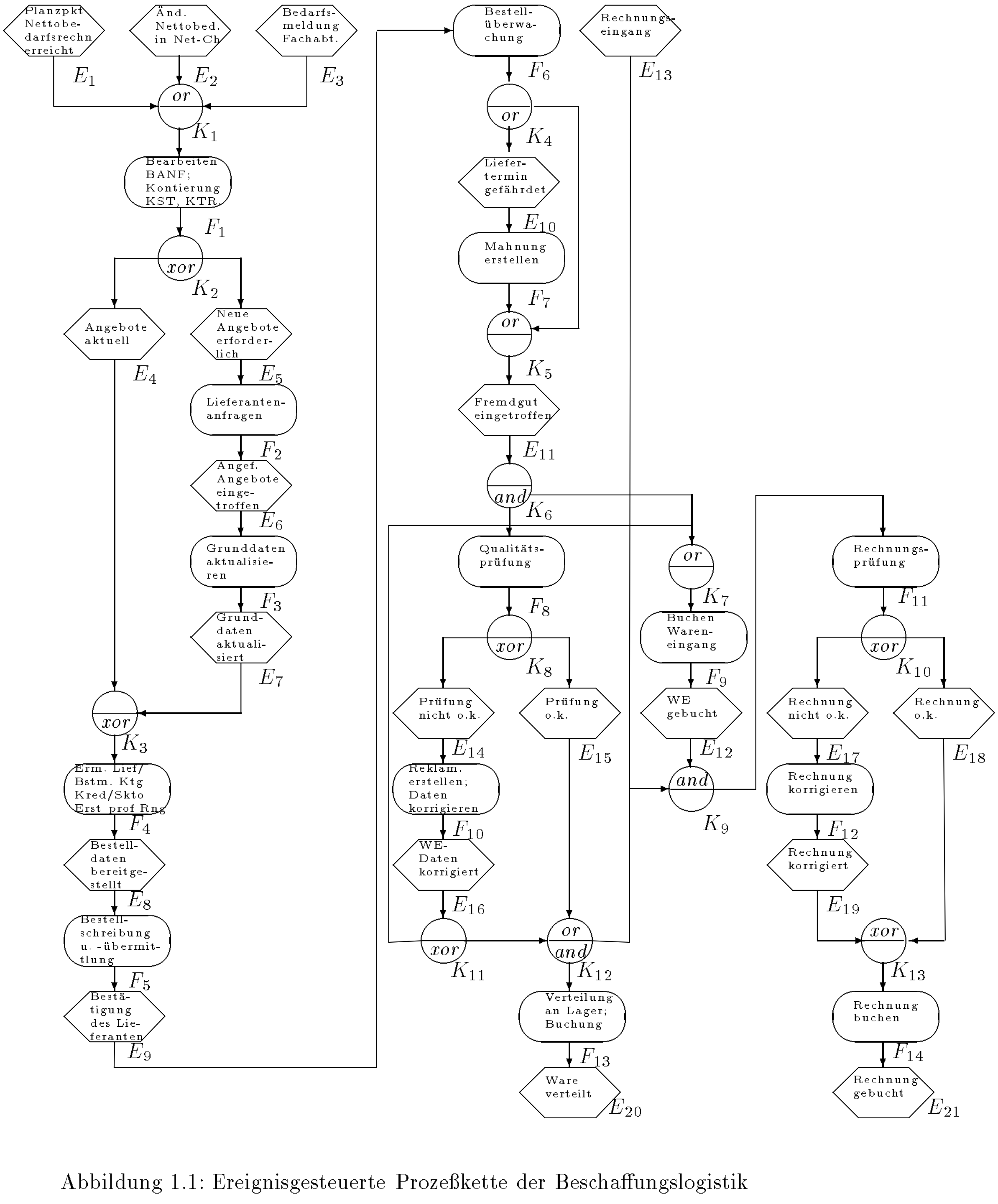


Abbildung 1.1: Ereignisgesteuerte Prozeßkette der Beschaffungslogistik

- „Rechnung gebucht“

In der Praxis der Geschäftsprozeßmodellierung ist es hilfreich zu fragen: Welche Ereignisse lösen eine gegebene Funktion aus, worauf muß die Funktion warten, um zu beginnen? I.a. wird man nicht das Vorliegen der Stammdaten, wohl aber das Eintreffen der Bewegungsdaten als Ereignis modellieren, bei einem Logistikprozeß also nicht „Stückliste vorhanden“, sondern das Ereignis „Primärbedarf freigegeben“.

Dennoch kann man das Modellierungselement „Ereignis“ einer EPK natürlich auch in einer allgemeineren Bedeutung verwenden: Zur Modellierung eines Zeitpunktes, eines Zustandes oder einer Vor- bzw. Nachbedingung. Hierbei kann man dann zwischen hinreichenden und notwendigen Vorbedingungen unterscheiden.

Überhaupt spiegelt die Modellierung eines Kontrollflusses erst an zweiter Stelle den zeitlichen Funktionsablauf, primär geht es immer um einen Kausalzusammenhang. Zwei Ereignisse sind genau dann durch einen gerichteten Weg miteinander verbunden, wenn sie in einem kausalen Zusammenhang stehen. Es wird i.a. jedoch viele Ereignisse geben, die nicht durch einen solchen Weg verbunden sind, sondern nebenläufig zueinander eintreten. Für ihr Eintreten ist dann keine zeitliche Reihenfolge festgelegt.

Die Ausführung einer Funktion annulliert alle ihre Eingangsereignisse, d.h. ihre Vorbedingungen, und aktualisiert alle ihre Ausgangsereignisse, die Nachbedingungen. Dadurch wird ein Zustandsübergang bewirkt.

Eine besondere Rolle spielen die Start- und Zielereignisse der gesamten EPK: Ein Prozeß kann erst dann starten, wenn besonders gekennzeichnete Startereignisse eingetreten sind, und er ist erst dann ordnungsgemäß beendet, wenn besonders gekennzeichnete Zielereignisse erreicht werden. Falls dabei mehrere Start- oder Zielereignisse möglich sind, so müssen die erlaubten Kombinationen explizit modelliert werden.

4. Logische Konnektoren einer EPK sind „und“, „oder“, „entweder-oder (xor)“. Sie charakterisieren Stellen, an denen sich ein Prozeß nach einer Entscheidung in Teilprozesse verzweigen kann, oder Stellen, an denen Teilprozesse zu einem einzigen Prozeß zusammengeführt werden.

◇

Eine EPK modelliert nicht einen individuellen Prozeß, z.B. die Abwicklung eines bestimmten Kundenauftrages, sondern einen Prozeßtyp, die Abwicklung von Kundenaufträgen. Es werden nicht einzelne Instanzen betrachtet, sondern eine EPK modelliert den Lebenslauf der gesamten Klasse „Kundenauftrag“. Genauer gesprochen handelt es sich sogar um die Modellierung aller Varianten im Lebenslauf dieser Klasse: Denn der Prozeß kann sich an verschiedenen Stellen in Alternativen verzweigen, und zusätzlich können unter den möglichen Anfangsereignissen alternativ verschiedene Ereignismengen markiert werden.

Bemerkung 1.2 [Syntax einer EPK]

1. Ein Blick in die Literatur [KM94], [Sch94] zeigt schnell, daß die Syntax einer EPK in der Praxis noch nicht standardisiert ist. Vielmehr wird die Syntax pragmatisch dem jeweiligen Zusammenhang angepaßt, um dem Leser ein möglichst einfaches und intuitiv einsichtiges Modell zu präsentieren. Die größte Vielfalt der Darstellung findet sich dabei bei der Modellierung der logischen Konnektoren und ihrer anstoßenden Kanten.

2. Eine andere Mehrdeutigkeit betrifft die Modellierung der Start- und Zielereignisse bei den in der Literatur auftretenden EPKs. Offensichtlich entsprechen Randstellen ohne Eingangskanten den Startereignissen, und analog Randstellen ohne Ausgangskanten den Zielereignissen. Aber es bleibt mitunter der Interpretation des Lesers überlassen, welche Kombination von Start- und Zielereignissen er als zulässig, d.h. als Abbild der Realität, ansehen soll.

   So wird man bei der EPK von Graphik 1.1 davon ausgehen, daß die logistische Kette der Beschaffung gestartet werden kann, sobald mindestens eines der Ereignisse

   - „Planungszeitpunkt der Nettobedarfsrechnung erreicht“
   - „Änderung Nettobedarf im Net-Change“
   - „Bedarfsmeldung der Fachabteilung“

   vorliegt. Und man wird den Prozeß erst dann als erfolgreich abgeschlossen betrachten, wenn beide Zielereignisse

   - „Ware verteilt“
   - „Rechnung gebucht“

   eingetreten sind. Offensichtlich kann dieses Ziel nur erreicht werden, wenn zu den oben genannten Startereignissen in jedem Fall das Ereignis

   - „Rechnungseingang“

   als weitere Vorbedingung hinzukommt. Diese Konjunktion von Startereignissen wird von der EPK jedoch nicht explizit modelliert.

$\diamond$

Wir rekapitulieren einige Grundbegriffe der Graphentheorie und legen dabei zugleich unsere Notation fest.

Definition 1.3 [Grundbegriffe der Graphentheorie]

1. Wir verstehen unter einem *gerichteten Graphen G* ein Tupel $G = (V, A, N)$ bestehend aus einer Knotenmenge V, einer Kantenmenge A und einer Abbildung

   $$N : A \rightarrow V \times V,$$

   die jeder Kante $a \in A$ das geordnete Paar

   $$N(a) \in V \times V$$

   zuordnet, das aus dem Eingangsknoten und dem Ausgangsknoten der Kante a besteht. In dieser Definition sind sowohl Parallelkanten wie Schlingen zugelassen.

   Wir werden im folgenden nur gerichtete Graphen betrachten, so daß wir i.a. den Zusatz „gerichtet“ weglassen.

2. An den meisten Stellen der Arbeit werden wir zudem nur einfache Graphen betrachten, d.h. es werden weder Parallelkanten noch Schlingen zugelassen. Die Kanten eines einfachen Graphen sind durch die Angabe ihres Anfangs- und ihres Endpunktes bestimmt. Daher verwenden wir zur Bezeichnung eines einfachen gerichteten Graphen auch die Notation G = (V, A) mit einer Knotenmenge V und einer irreflexiven Relation $A \subset V \times V$ als Kantenmenge.

3. Ein *Weg* γ in einem einfachen Graphen $G = (V, A)$ von einem Knoten v_a zu einem Knoten v_e ist eine endliche Folge von Knoten $v_i \in V, i = 0, \ldots, n$,

$$\gamma = (v_0, v_1, \ldots, v_n), v_0 = v_a, v_n = v_e,$$

so daß je zwei aufeinanderfolgende Knoten eine Kante

$$(v_i, v_{i+1}) \in A, i = 0, \ldots, n-1$$

bilden und alle Kanten paarweise verschieden sind. Man nennt n die Länge des Weges. Der Weg γ heißt einfach, wenn auch die Knoten $v \in \gamma$ paarweise verschieden sind.

Ein *Zyklus* ist ein Weg $\gamma = (v_0, v_1, \ldots, v_n)$ mit $v_0 = v_n$ und paarweise verschiedenen Knoten

$$v_i, i = 0, \ldots, n-1.$$

4. Ein Graph $G = (V, A)$ heißt *zusammenhängend*, wenn zwischen je zwei Knoten $v, w \in V$ ein Weg von v nach w oder ein Weg von w nach v führt.

5. Ein Graph $G = (V, A)$ heißt *stark-zusammenhängend*, wenn zwischen je zwei Knoten $v, w \in V$ sowohl ein Weg von v nach w als auch ein Weg von w nach v führt.

□

Wir legen die Syntax einer EPK durch folgende Definition fest:

Definition 1.4 [Syntax einer EPK] Eine *ereignisgesteuerte Prozeßkette* (EPK) ist ein gerichteter, zusammenhängender, einfacher Graph $EPK = (V, A)$ mit folgenden Eigenschaften:

1. Die *Knotenmenge* V ist die Vereinigung von vier paarweise disjunkten Mengen

   E = Menge der Ereignisse $\neq \emptyset$
   F = Menge der Funktionen $\neq \emptyset$
   P = Menge der Prozeßwegweiser
   L = Menge der binären Konnektoren vom Typ *and* , *or* , *xor* .

2. Die *Kantenmenge* A ist eine antisymmetrische Relation auf $V \times V$ mit folgenden Eigenschaften:

   - Alle Kanten verbinden zwei Knoten von jeweils unterschiedlichem Typ, d.h.

     $$A \cap (E \times E \cup F \times F \cup P \times P \cup L \times L) = \emptyset.$$

   - Die Vorgänger eines Knotens gehören alle zu einem einzigen Knotentyp, und ebenso gehören alle Nachfolger zu einem einzigen Typ, d.h. für jeden Knoten $v \in V$ gilt:

     Entweder ${}^\bullet v \subseteq E$ oder ${}^\bullet v \subseteq F$ oder ${}^\bullet v \subseteq P$ oder ${}^\bullet v \subseteq L$ und entweder $v^\bullet \subseteq E$ oder $v^\bullet \subseteq F$ oder $v^\bullet \subseteq P$ oder $v^\bullet \subseteq L$

3. Ereignisse, Funktionen und Prozeßwegweiser sind *unverzweigt*, genauer: Für alle Ereignisse $e \in E$ gilt

$$| {}^{\bullet}e | \leq 1 \text{ und } | e^{\bullet} | \leq 1,$$

für alle Funktionen $f \in F$ gilt

$$| {}^{\bullet}f | = | f^{\bullet} | = 1$$

und für alle Prozeßwegweiser $p \in P$ gilt

$$| {}^{\bullet}p | = | p^{\bullet} | = 1.$$

4. Der Rand des Graphen besteht nur aus Ereignissen. Es gibt mindestens

   - ein Startereignis s_a, d.h. ein $s_a \in E$ mit

   $${}^{\bullet}s_a = \emptyset,$$

   und mindestens

   - ein Zielereignis s_e, d.h. ein $s_e \in E$ mit

   $$s_e^{\bullet} = \emptyset.$$

□

Bemerkung 1.5 [Zum Aufbau einer EPK]

1. In der Unverzweigtheit von Ereignissen, Funktionen und Prozeßwegweisern einer EPK drückt sich die Absicht ihrer Erfinder aus, die Verzweigungen eines Prozesses ausschließlich durch Einführung logischer Konnektoren zu modellieren. In wieweit sich diese Absicht durchhalten läßt, darauf werden wir in Kapitel 5 bei der Modellierung von Prozeßschleifen zurückkommen.

2. Die Forderung des Zusammenhanges von G bedeutet, daß zwei isolierte, von einander unabhängige Graphen als zwei verschiedene EPKs angesehen werden. In diesem Fall kann jede EPK für sich, unabhängig von der anderen, studiert werden, so daß die Forderung des Zusammenhanges keine wesentliche Einschränkung darstellt.

3. Wenn man sich die Startereignisse einer EPK als mögliche Ausgangspunkte eines realen Prozesses vorstellt, so sollte die EPK eine Situation modellieren, in welcher die Bearbeitung des Prozesses schließlich zum Erreichen eines der möglichen Zielereignisse führt. Der Prozeß sollte weder an inneren Ereignissen „hängenbleiben“, noch an anderen Stellen als den vorgegebenen Zielereignissen terminieren. Außerdem sollte die EPK keine Netzteile enthalten, die nie durchlaufen werden können. Daher werden wir nach der Übersetzung einer EPK in ein Petri-Netz eine Methode zur Netzanalyse (vgl. Kapitel 7) angeben, um diese Eigenschaften der „Lebendigkeit“ formal zu prüfen.

4. Zur Unterscheidung von Kontroll- und Datenfluß bei der Modellierung mit EPKs machen die Autoren in [KNS91] folgende Bemerkung:

> „Neben der Ausweisung des Kontrollflusses (ereignisgesteuerte Prozeßkette) kann bei der Gestaltung von integrierten Informationssystemen die Analyse der in einen Funktionstyp ein- und ausgehenden Informationsobjekte von Interesse sein. Dies geschieht über die Input/Output-Zuordnung der Informationsobjekte bzw. Attribute im Funktionsmodell.“

Hier wird den EPKs explizit die Modellierung des Kontrollflusses zugewiesen, während die beteiligten Informationsobjekte in einer anderen Komponente des Unternehmensmodells, dem statischen Funktionsmodell, ausmodelliert werden. Über die EPKs fließt also nicht der Datenstrom, sondern nur die benötige Steuerungsinformation.

5. Im Unterschied zu EPKs modellieren andere Verfahren auf der Basis höherer Petri-Netze gerade den Datenfluß realer Objekte. Durch ein solches Petri-Netz fließen dann z. B. simultan 20 verschiedene Marken, die jede einen individuellen Auftrag repräsentieren. Diese weitergehende Form der Prozeßmodellierung verwendet gefärbte Petri-Netze — zur Theorie siehe [Jen92], [Jen95]; für Anwendungen höherer Petri-Netze in der Wirtschaftsinformatik siehe zusätzlich [Reu95]. Außerdem werden auf dem Markt verschiedene Petri-Netz Tools angeboten, die auch bereits heute (1996) Simulationskomponenten enthalten.

◇

Bemerkung 1.6 [Objektorientierung]

1. Hinsichtlich der Objektorientierung von ARIS und der EPK-Methode durch die Möglichkeit, Klassendiagramme zu zeichnen und zu referenzieren, äußert Scheer in [Sch96]:

   > „Durch die zuvor beschriebenen Erweiterungen besteht mit der ARIS-Architektur erstmals die Möglichkeit, Unternehmen bzw. Anwendungssysteme auf der fachlichen Ebene vollständig objektorientiert zu beschreiben, ohne hierbei auf die Aspekte der Aufbau- und Ablauforganisation zu verzichten.“

   Wir können uns dieser Charakterisierung nicht anschließen.

   Die objektorientierte Prozeßmodellierung ist eine nicht-triviale Aufgabe, die u. E. noch nicht befriedigend gelöst ist. Die EPK-Methode ist nicht aus einem objektorientierten Ansatz entstanden. Im Rahmen von ARIS stellt sie vielmehr in Form einer „Steuerungssicht“ eine nachträgliche Verbindung von Daten-, Funktions- und Organisationssicht dar. Dagegen wären bei einem objektorientierten Ansatz von vornherein Daten und Funktionen als zwei Seiten desselben Begriffes, des Objektes, behandelt worden.

2. Wir können den Anhängern einer objektorientierten Prozeßmodellierung, welche die EPK-Methode einsetzen, daher nur raten, zumindest in Gedanken alle Ereignisse und Funktionen einer EPK jeweils einer Klasse zuzuordnen und diesen Bezug durch die Beschriftung der Netzelemente auszudrücken. Ereignisse sind dann die möglichen Zustände aller Objekte der Klasse, und Funktionen sind die Operationen der Klasse. Die Prozesse der EPKs operieren immer mit Objekten. Der Kontrollfluß steuert Funktionen, die Operationen eines bestimmten Objektes sind, und löst Ereignisse aus, die Zustände von bestimmten Objekten sind. Auf keinen Fall sollte man das Funktionsmodell konzeptionell vom Prozeßmodell trennen.

   Als objektorientierte Beschriftung bietet sich folgende Namenskonvention an:

- <Klasse>.<Ereignis>
- <Klasse>.<Funktion>

Falls das Objektmodell eines Unternehmens in einem höheren Detaillierungsgrad vorliegt als das Prozeßmodell, können für die Beschriftung statt der Klassen auch Cluster (von Klassen) gewählt werden. Dieses Vorgehen bietet sich z.B. bei einem Projekt auf Basis des SAP-Datenmodells an.

3. Im Unterschied zur Kontrollfluß-Modellierung halten wir die Datenfluß-Modellierung für einen aussichtsreichen Kandidaten einer objektorientierten Prozeßmodellierung. Hierbei repräsentieren die individuellen Marken die einzelnen Objekte. Vgl. auch NR/T-Netze in [Obe96].

$\diamond$

Beispiel 1.7 [EPK der Beschaffungslogistik] Wir erläutern unsere bisherigen Überlegungen an der anfangs gezeigten Graphik 1.1 der EPK der Beschaffungslogistik. Um diese EPK in die Syntax von Definition 1.4 zu bringen, gehen wir in folgenden Schritten vor:

1. Korrektur der Druckfehler:

   Die Abbildung enthält vermutlich zwei Druckfehler:

   - Der Konnektor K_{11} nach dem Ereignis „Wareneingangsdaten korrigiert" muß statt „ *xor* " ein *and*-Konnektor K_{110} sein. Mit „ *xor* " wird entweder eine Korrektur gebucht, aber keine Ware eingelagert, oder es wird Ware eingelagert, aber keine Korrektur gebucht. Mit „ *and* " wird dagegen sowohl Korrektur gebucht als auch Ware eingelagert. Wir ersetzen daher den *xor*-Konnektor K_{11} durch einen *and*-Konnektor K_{110}.
   - Die „ *or* "-Konnektoren K_4 und K_5 in der Alternative nach der Funktion „Bestellüberwachung" müssen *xor*-Konnektoren K_{40} bzw. K_{50} sein. Entweder ist der Liefertermin gefährdet oder er ist nicht gefährdet, beides zugleich ist nicht möglich. Wir ersetzen daher die beiden *or*-Konnektoren K_4 und K_5 durch zwei *xor*-Konnektoren K_{40} bzw. K_{50}.

2. Auflösen von Mehrfach-Konnektoren:

   Der logische Konnektor K_{12} wird in zwei logische Konnektoren aufgelöst, einen *or*-Konnektor K_{121} für die Verknüpfung der Eingänge und einen *and*-Konnektor K_{122} für die Verknüpfung der Ausgänge.

3. Einfügen von Ereignisen:

   Um die Bedingung 2 von Definition 1.4 zu erfüllen,

   - daß eine Kante nur Knoten von verschiedenem Typ verbindet und
   - daß alle Vorgänger bzw. alle Nachfolger eines Knotens vom selben Typ sind,

   haben wir an verschiedenen Stellen der EPK Ereignisse eingefügt.

   Die resultierende EPK in Abbildung 1.2 besitzt

   - vier Startereignisse E_1, E_2, E_3, E_{13} und
   - zwei Zielereignisse E_{20} und E_{21}.

$\diamond$

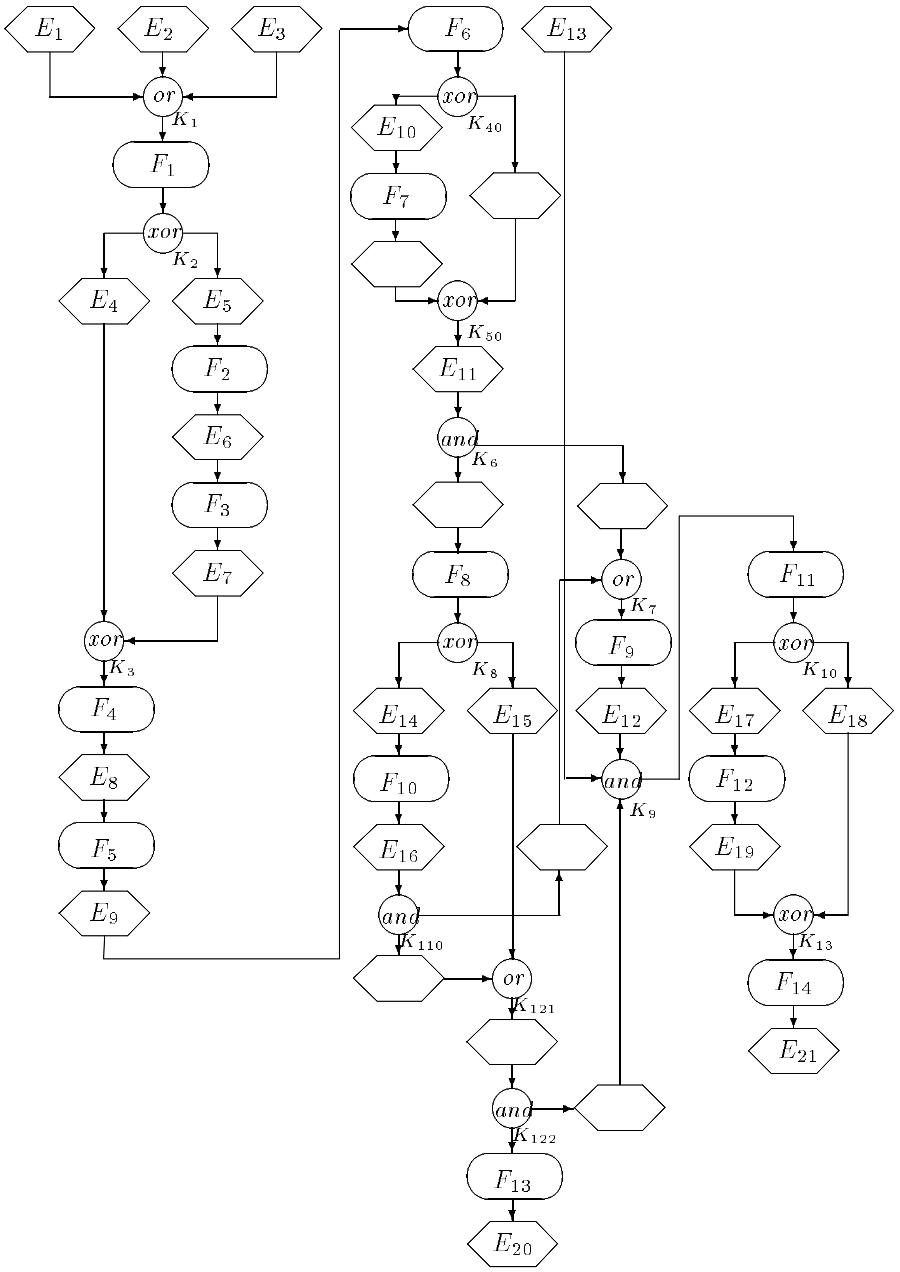


Abbildung 1.2: EPK der Beschaffungslogistik

Bemerkung 1.8 [Semantik der logischen Konnektoren]
Bei Vorgabe von drei Arten logischer Konnektoren für eine EPK gibt es offensichtlich 12 paarweise verschiedene logische Verknüpfungen. Wir notieren sie in Anlehnung an die Reihenfolge in [KNS91] und verweisen jeweils auf ein betriebswirtschaftlich relevantes Beispiel aus [Sch94] und [KM94]; die Beschriftungen der EPKs sind gemäß Bemerkung 1.6 angepaßt.
Wir gehen die 12 Fälle der Reihe nach durch und versuchen auf Grund von [KNS91] und den praktischen Beispielen aus [Sch94], die jeweils gemeinte Semantik der logischen Verknüpfung als Schaltregel zu rekonstruieren. Allerdings gelingt das nicht in allen Fällen — es bleiben Fragen offen. Wir werden diese offenen Fragen später in Bemerkung 3.4 dadurch beantworten, daß wir die EPK zusammen mit ihren Schaltregeln in ein Petri-Netz übersetzen. Da die Semantik eines Petri-Netzes wohldefiniert ist, legen wir also die Semantik einer EPK durch Angabe unserer Übersetzungsregeln (Definition 3.2) fest.
Fall 1.

Beispiel:	[Sch94, S. 420]
Ereignisse:	Angebote.Aktuell, Grunddaten.Aktualisiert
Konnektor:	*xor*
Funktion:	Lieferant.Ermitteln
Schaltregel:	Wenn entweder das eine oder das andere Ereignis eingetreten ist, kann die anschließende Funktion starten.

Was soll die Schaltregel bedeuten, wenn beide Ereignisse nacheinander eintreten? Kann die Funktion dann ein zweites Mal ablaufen: Das erste Mal nach Eintreten des ersten Ereignisses und vor Eintreten des zweiten, das zweite Mal nach Eintreten des zweiten Ereignisses? Welchen Sachverhalt will der Modellierer ausdrücken, wenn er diesen Konnektor verwendet?

- „Wenn beide Ereignisse gleichzeitig eintreten, blockieren sie die nachfolgende Funktion“ oder
- „Beide Ereignisse können nicht gleichzeitig eintreten“ oder
- „Wenn die nachfolgende Funktion startet, so war vorher genau eines der beiden Ereignisse eingetreten“?

Fall 2.

Beispiel:	[KM94], Faltblatt FI Finanzwesen, Debitorenmahnung
Ereignisse:	Mahnung.ParameterFestgelegt, Mahnung.Fällig
Konnektor:	*and*
Funktion:	Mahnung.PrüfenKonto
Schaltregel:	Wenn beide Ereignisse eingetreten sind, kann die Funktion starten.

Fall 3.

Beispiel:	[Sch94, S. 420]
Ereignisse:	Anforderung.PlanungszeitpunktErreicht, Bedarfsmeldung.Angelegt
Konnektor:	*or*
Funktion:	Anforderung.Bearbeiten
Schaltregel:	Wenn mindestens eines der beiden Ereignisse eingetreten ist, kann die anschließende Funktion starten. Wenn beide Ereignisse gleichzeitig eintreten, kann die Funktion aber nur einmal starten.

In diesem Fall stellen sich ähnliche Fragen wie in Fall 1.

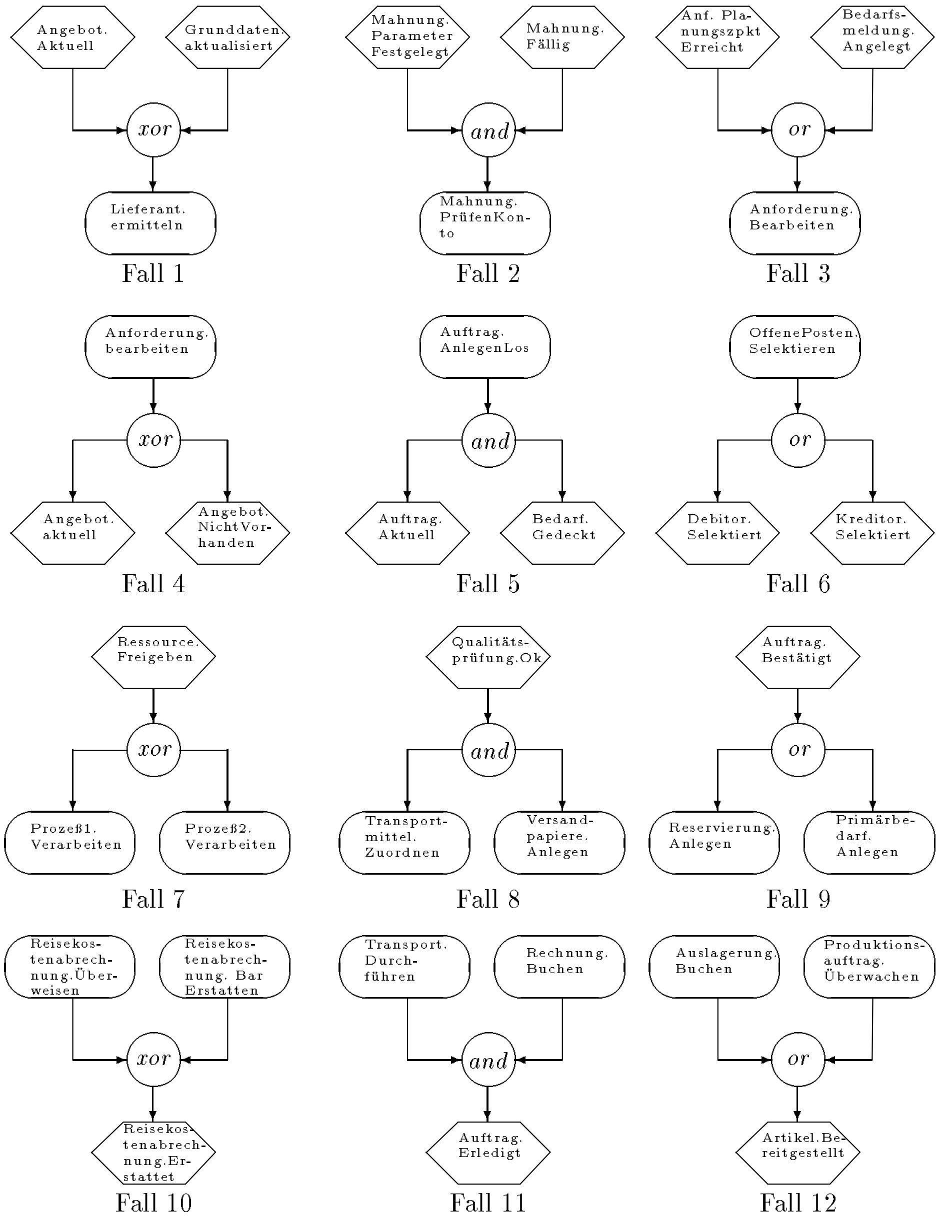


Abbildung 1.3: Binäre logische Konnektoren einer EPK.

Fall 4.

Beispiel:	[Sch94, S. 420]
Funktion:	Anforderung.Bearbeiten
Konnektor:	*xor*
Ereignisse:	Angebot.Aktuell, Angebot.NichtVorhanden
Schaltregel:	Bei Ausführung der Funktion wird entschieden, ob entweder das eine oder das andere Ereignis eintritt.

Man könnte erwarten, daß ein mit einem „ *xor* “-Konnektor verzweigter Prozeß („branch“) an späterer Stelle auch wieder mit einem „ *xor* “-Konnektor („merge“) gemäß Fall 7. oder 10. geschlossen wird. In der EPK „Beschaffungslogistik“ aus Beispiel 1.7 wird ein mit „ *xor* “ verzweigter Prozeß stattdessen jedoch mit einem „ *or* “-Konnektor gemäß Fall 3 geschlossen.

Fall 5.

Beispiel:	[Sch94, S. 177]
Funktion:	Auftrag.AnlegenLos
Konnektor:	*and*
Ereignisse:	Auftrag.Aktuell, Bedarf.Gedeckt
Schaltregel:	Durch Ausführung der Funktion treten beide Ereignisse ein.

Fall 6.

Beispiel:	[KM94], Faltblatt FI Finanzwesen, Manuelle Zahlung
Funktion:	OffenePosten.Selektieren
Konnektor:	*or*
Ereignisse:	Debitor.Selektiert, Kreditor.Selektiert
Schaltregel:	Bei Ausführung der Funktion wird entschieden, ob nur das eine, nur das andere, oder beide Ereignisse eintreten.

Fall 7.

Es handelt sich um den Fall eines „ *xor* “-Konnektors mit einem einzelnen Ereignis als Eingang und zwei Funktionen als Ausgängen. Für diesen Fall haben wir in [KM94] oder in [Sch94] kein Beispiel gefunden, der Fall wird in [KNS91] verboten. Man könnte das eingehende Ereignis als eine Störung interpretieren, welche genau eine von zwei möglichen Reaktionen auslöst, oder als Freigabe einer Ressource, die von zwei Funktionen nur exklusiv genutzt werden kann.

Fall 8.

Beispiel:	[Sch94, S. 260]
Ereignis:	Qualitätsprüfung.OK
Konnektor:	*and*
Funktionen:	Transportmittel.Zuordnen, Versandpapiere.Anlegen
Schaltregel:	Nach Eintritt des Ereignisses können beide Funktionen starten.

Fall 9.

Beispiel:	[Sch94, S. 450]
Ereignis:	Auftrag.Bestätigt
Konnektor:	*or*
Funktionen:	Reservierung.Anlegen, Primärbedarf.Anlegen
Schaltregel:	Nach Eintritt des Ereignisses wird entschieden, ob nur die eine, nur die andere, oder beide Funktionen starten können.

Auch dieser Fall wurde in [KNS91] verboten. In [Sch94] wird darauf hingewiesen, daß man in solchen Fällen eine explizite Entscheidungsfunktion einführen kann. Dabei würde dann Fall 9 durch Fall 6 ersetzt werden.

Fall 10.	
Beispiel:	[KM94], Faltblatt HR Personalwirtschaft, Reiseantragsbearbeitung
Funktionen:	Reisekostenrechnung.überweisen, Reisekostenrechnung.BarErstatten
Konnektor:	*xor*
Ereignis:	Reisekostenrechnung.Erstattet
Schaltregel:	Das Ereignis tritt ein, wenn genau eine der beiden Funktionen abgelaufen ist.

In diesem Fall stellen sich ähnliche Fragen wie in Fall 1.

Fall 11.	
Beispiel:	[Sch94, S. 450]
Funktionen	Transport.Durchführen, Rechnung.Buchen
Konnektor	*and*
Ereignis	Auftrag.Erledigt
Schaltregel	Erst wenn beide Funktionen ausgeführt sind, tritt das Ereignis ein.

Fall 12.	
Beispiel:	[Sch94, S. 450],
Funktionen:	Auslagerung.Buchen, Produktionsauftrag.überwachen
Konnektor:	*or*
Ereignis:	Artikel.Bereitgestellt
Schaltregel:	Nach Ausführung jeder der beiden Funktionen tritt das Ereignis ein.

In diesem Fall stellen sich ähnliche Fragen wie in Fall 1. $\diamond$

Bemerkung 1.9 In [KNS91, S. 14], werden die beiden Fälle 7 und 9 verboten, da Ereignisse keine Entscheidungen treffen dürfen. Da die Autoren für Fall 9 aber selbst ein Beispiel geben, sehen wir keinen Grund, diese beiden Fälle auszuschließen. Zudem werden wir im Rahmen von Definition 3.2 vorschlagen, in jedem Fall zwischen eine Funktion und einen logischen Konnektor ein Ereignis einzuschieben. Dadurch werden die beiden Fälle 4 und 6 auf die Fälle 7 und 9 zurückgeführt. $\diamond$

Kapitel 2

Boolesche Netze

Bemerkung 2.1 [Boolesche Netze]

1. Wir wollen in dieser Arbeit EPKs in Petri-Netze übersetzen. Welche Klasse von Petri-Netzen ist geeignet, um den Kontrollfluß eines Systems zu modellieren?

   Wir haben uns für eine einfache Klasse gefärbter Petri-Netze entschieden — wir nennen sie „Boolesche“ Netze — mit

   - nur einem einzigen Datentyp $Boole := \{0, 1\}$ als Farbe
   - jeweils einer einzigen Variablen als Kantenbeschriftung
   - einer Transitionsbeschriftung in der Sprache der Aussagenlogik.

2. Dabei verstehen wir unter der Sprache BOOLE der Aussagenlogik — wie üblich — die Menge der Booleschen Formeln oder Booleschen Ausdrücke, d.h. die Menge der durch folgende Grammatik erzeugten Zeichenreihen:

   (a) Die *Menge der Terminale* besteht aus
   - den Variablen „x“, „y“, etc.,
   - den beiden Konstanten TRUE und FALSE,
   - den syntaktischen Zeichen „(“ und „)“
   - den logischen Operatoren „ *or* “, „ *and* “, „ *xor* “, „$\neg$“, „$\Rightarrow$“, „=“.

   (b) Die *Menge der Nichtterminale* besteht aus
   - Ausdruck
   - Variable

   (c) Es gibt ein *Startelement* S.

   (d) Die *Produktionen* sind
   - S → Ausdruck
| „(“ Ausdruck „=“ Ausdruck „)“
| „(“ Ausdruck „ *or* “ Ausdruck „)“
| „(“ Ausdruck „ *and* “ Ausdruck „)“
| „(“ Ausdruck „ *xor* “ Ausdruck „)“
| „(“ „$\neg$“ Ausdruck „)“
| „(“ Ausdruck „$\Rightarrow$“ Ausdruck „)“
   - Ausdruck → Variable | TRUE | FALSE
   - Variable → x | y | ....

Klammern können nach den üblichen Regeln weggelassen werden. Weitere aussagenlogische Operatoren wie „$\Leftrightarrow$“, „$|$“ (Sheffer-Strich) sind als Abkürzungen aufzufassen, z.B. $x \mid y := \neg(x \textit{ and } y)$.

3. Ein Boolesches Netz ist insbesondere ein gefärbtes Petri-Netz im Sinne von [Jen92] und speziell auch ein Prädikat/Transitionsnetz im Sinne von [Rei86]. Für diese Netze werden keine Multimengen gebraucht, der übliche Teilmengen-Begriff reicht für Boolesche Netze aus. Jedes Boolesche Netz hat ein unterliegendes Stellen/Transitionsnetz, und die Schaltregeln für Boolesche Markierungen werden die Schaltregeln des unterliegenden Stellen/Transitionssystems respektieren.

$\diamond$

Bemerkung 2.2 [Netze]

1. Eine Teilklasse aller einfachen Graphen ist die Klasse der Netze. Netze sind bipartite, einfache Graphen $N = (S, T; A)$: Sie haben als Knotenmenge eine Vereinigung von zwei disjunkten Mengen, der Menge S der Stellen, und der Menge T der Transitionen. Hinzu kommt die Menge A der Kanten, diese verlaufen nur zwischen Knoten verschiedener Art:

$$A \subseteq S \times T \cup T \times S.$$

2. Sei $N = (S, T; A)$ ein Netz.

   (a) Das Netz N heißt *T-Netz* (oder Synchronisationsgraph), wenn jede seiner Stellen unverzweigt ist, d.h.

   $$\mid s^\bullet \mid = \mid {}^\bullet s \mid = 1 \text{ für alle } s \in S.$$

   (b) Das Netz N heißt *S-Netz*, wenn jede seiner Transitionen unverzweigt ist, d.h.

   $$\mid t^\bullet \mid = \mid {}^\bullet t \mid = 1 \text{ für alle } t \in T.$$

   (c) Das Netz N heißt *Free-Choice Netz*, wenn für jede seiner Kanten $(s, t) \in A$ mit $s \in S$ und $t \in T$ gilt:

   $${}^\bullet t \times s^\bullet \subseteq A.$$

3. Offensichtlich entsprechen die T-Netze bijektiv den Graphen mit nur einer einzigen Knotenart: Aus einem T-Netz TN erhält man den zugehörigen Graphen G, indem man jede Transition von TN zu einem Knoten von G macht und jede Stelle von TN zusammen mit ihren beiden Kanten durch eine einzige Kante von G ersetzt. Die resultierenden Graphen haben also keine Stellen mehr, sondern nur noch Knoten und Kanten. Allerdings sind die Graphen i.a. nicht mehr einfach, sondern können sowohl Schlingen wie Parallelkanten besitzen. In dieser Arbeit werden wir diese Stellenreduktion nicht vornehmen und alle T-Netze mit expliziten Stellen betrachten. Dagegen verwenden die Autoren in [GT84] alle T-Netze in der stellenreduzierten Form.

4. Man sieht leicht, daß der Begriff „Free-Choice Netz“ eine gemeinsame Verallgemeinerung der beiden Begriffe „S-Netz“ und „T-Netz“ ist.

$\diamond$

Definition 2.3 [Boolesches Netz]

1. Ein *Boolesches Netz* $BN = (N; x, g)$ ist ein Netz $N = (S, T; A)$, das in der Sprache der Aussagenlogik beschriftet ist. Die Beschriftung besteht aus:

2. Einer Kantenbeschriftung

   $$x : A \to Var(BOOLE),$$

   die jeder Kante $a \in A$ eine Boolesche Variable $x(a)$ zuordnet. Dabei ist $Var(BOOLE)$ die Variablenmenge der Sprache der Aussagenlogik.

3. Einer Transitionsbeschriftung

   $$g : T \to BOOLE, t \mapsto g_t,$$

   die jeder Transition $t \in T$ einen Booleschen Ausdruck g_t, die Guard-Formel von t, zuordnet. Die Variablen von g_t sind alle Variablen $x(a)$ der Ein- und Ausgangskanten a von t.

   Wir nennen jede Transition t von N zusammen mit ihrer Guard-Formel g_t eine *Boolesche Transition* von BN.

4. Das Boolesche Netz $BN = (N; x, g)$ heißt *Boolesches Free-Choice Netz*, bzw. *Boolesches T-Netz*, bzw. *Boolesches S-Netz*, wenn das unterliegende Netz N ein Free-Choice Netz, bzw. ein T-Netz, bzw. ein S-Netz ist.

□

Im folgenden rekapitulieren wir einige Grundbegriffe und Eigenschaften aus der Theorie der Stellen/Transitionsnetze (ohne Gewichte und mit Kapazität ∞). Als einschlägige Lehrbücher empfehlen wir [Bau90], [DE95], [Rei86].

Definition 2.4 [Stellen/Transitionsnetz] Es sei $N = (S, T; A)$ ein Netz.

1. Eine *Markierung* von N ist eine Abbildung

   $$M : S \to \mathbb{N},$$

   wobei die natürliche Zahl $M(s), s \in S$, die Markenanzahl der Stelle s angibt.

   Der *Träger einer Markierung* M ist die Menge

   $$supp(M) := \{s \in S : M(s) \neq 0\}.$$

   Im Spezialfall, daß alle Stellen höchstens eine einzige Marke tragen, ist die Markierung die charakteristische Funktion ihres Trägers

   $$M = \chi_{supp(M)} : S \to \{0, 1\}$$

   mit

   $$\chi_{supp(M)}(s) = \begin{cases} 1 & , s \in supp(M) \\ 0 & , s \notin supp(M). \end{cases}$$

2. Eine Markierung M *aktiviert* eine Transition $t \in T$, wenn
$$^\bullet t \in supp(M),$$
d.h. wenn alle Eingangsstellen von t mindestens eine Marke tragen.

3. Eine unter der Markierung M_{pre} aktivierte Transition t erzeugt durch ihr *Schalten* die neue Markierung M_{post} mit
$$M_{post}(s) = \begin{cases} M_{pre}(s) - 1 & , s \in {}^\bullet t \setminus t^\bullet \\ M_{pre}(s) & , s \in {}^\bullet t \cap t^\bullet \\ M_{pre}(s) + 1 & , s \in t^\bullet \setminus {}^\bullet t \\ M_{pre}(s) & , s \notin {}^\bullet t \cup t^\bullet \end{cases}$$
Man schreibt $M_{pre}[t > M_{post}$ und sagt, daß die Markierung M_{post} durch *Schalten* der Transition t aus der Markierung M_{pre} hervorgeht.

4. Die Auswahl einer *Anfangsmarkierung* M_0 definiert ein Netzsystem $NS = (N; M_0)$.

5. Man bezeichnet mit $[M_0 >$ die *Menge der erreichbaren Markierungen* von $NS = (N; M_0)$, d.h. derjenigen Markierungen, die durch das sukzessive Schalten einer endlichen Folge von Transitionen aus der Markierung M_0 hervorgehen.

□

Definition 2.5 Es sei $N = (S, T; A)$ ein Netz.

1. Eine Markierung M von N heißt *sicher*, wenn für alle Markierungen $M_1 \in [M >$ gilt:
$$M_1(s) \leq 1 \text{ für alle } s \in S.$$

2. Eine Markierung N von M heißt *lebendig*, wenn jede Transition immer wieder schalten kann, d.h. zu jeder Markierung
$$M_1 \in [M > \text{ und zu jeder Transition } t \in T$$
existiert eine Markierung
$$M_2 \in [M_1 >,$$
welche die Transition t aktiviert.

3. Das Netz N heißt *wohlgeformt*, wenn es eine sichere und lebendige Markierung M besitzt.

□

Definition 2.6 [Boolesches Netzsystem] Es sei $BN = (N; x, g)$ ein Boolesches Netz, dessen unterliegendes Netz $N = (S, T; A)$ stark-zusammenhängt.

1. Eine *Boolesche Markierung* von BN ist eine Abbildung

$$BM = (BM_0, BM_1) : S \to \mathbb{N} \times \mathbb{N},$$

die auf N eine lebendige und sichere Markierung

$$M : S \to \mathbb{N}, M(s) := BM_0(s) + BM_1(s)$$

induziert.

Der *Träger von BM* ist die Menge

$$supp(BM) := \{s \in S : BM(s) \neq (0,0)\}.$$

2. Nach Wahl einer Booleschen Markierung BM heißt das Paar $BNS = (BN; BM)$ ein *Boolesches Netzsystem*.

□

Bemerkung 2.7 [Boolesches Netzsystem]

1. Von den beiden Komponenten einer Booleschen Markierung

$$BM = (BM_0, BM_1)$$

gibt $BM_0(s)$ die Anzahl der Marken „0“ und $BM_1(s)$ die Anzahl der Marken „1“ auf einer Stelle s an. Aus der vorausgesetzten Sicherheit der induzierten Markierung $BM_0 + BM_1$ folgt, daß auf jeder Stelle s höchstens eine einzige Marke liegt. Im Falle $BM_0(s) = 1$ heißt die Stelle *deaktiviert*, falls $BM_1(s) = 1$ heißt die Stelle *aktiviert*.

2. Für ein Boolesches Netzsystem $BNS = (BN; BM_0)$ faßt man die Markierung BM_0 als den Anfangszustand auf.

3. Man erhält aus einem gegebenen Booleschen Netz

$$BN = (N; x, g)$$

das unterliegende Netz N zurück, indem man die Beschriftung der Kanten und Booleschen Transitionen „vergißt“.

In der Sprache der Kategorientheorie ausgedrückt ist also nicht nur der Übergang von den Booleschen Netzen zu den Stellen/Transitionsnetzen „funktoriell“, sondern auch der Übergang von den Booleschen Netzsystemen zu den Stellen/Transitionssystemen. Beim Übergang zu den Stellen/Transitionssystemen vergißt man auch noch den Unterschied der beiden Markenarten „0“ und „1“, denn die Marken eines Stellen/Transitionssystems sind ununterscheidbar.

Wir werden die Semantik eines Booleschen Netzsystems so definieren, daß sie verträglich ist mit der Semantik des unterliegenden Stellen/Transitionssystems.

4. Wir haben für eine Boolesche Markierung die Sicherheit der unterliegenden Markierung verlangt. Hierin drückt sich die naheliegende Forderung aus, daß eine Stelle eines Booleschen Netzes

- weder mit mehreren „1“-Marken aktiviert
- noch mit mehreren „0“-Marken deaktiviert
- noch gleichzeitig mit einer „1“-Marke und einer „0“-Marke markiert ist.

Die Lebendigkeit der unterliegenden Markierung bedeutet, daß zumindest aus Sicht des unterliegenden Stellen/Transitionsnetzes keine Hindernisse bestehen, alle Transitionen immer wieder zu aktivieren.

5. Was gewinnt man für Boolesche Netze aus den Überlegungen bezüglich der unterliegenden Stellen/Transitionsnetze und ihrer Markierungen?

   Zunächst ist der starke Zusammenhang von N eine notwendige Bedingung für die Existenz einer lebendigen und sicheren Markierung, vgl. [DE95], Chapter 2.

   Wir werden außerdem in Satz 5.20 sehen, daß die Untersuchung des unterliegenden Netzes in Verbindung mit einem allgemeinen Satz über Free-Choice Netze (Satz 5.17) die Existenz einer Booleschen Markierung auf solchen Booleschen Netzen garantiert, die aus der Übersetzung einer EPK stammen.

$\diamond$

Definition 2.8 [Schaltregel] Es sei $BN = (N; x, g)$ ein stark-zusammenhängendes Boolesches Netz und t eine vorgegebene Boolesche Transition von BN.

1. Ein *Zuweisungselement* von BN ist ein Paar (t, b) mit einer Wertzuweisung der Variablen

$$b : \{x(s,t) : s \in {}^\bullet t\} \cup \{x(t,s) : s \in t^\bullet\} \to BOOLE,$$

   so daß die Guard-Formel $g_t(b_{in}, b_{out})$ der Transition t erfüllt ist; dabei wurde zur Abkürzung gesetzt:

$$\begin{aligned} b_{in} &= (b(x(s,t)))_{s \in {}^\bullet t}, \\ b_{out} &= (b(x(t,s)))_{s \in t^\bullet}. \end{aligned}$$

2. Eine Boolesche Markierung BM von BN *aktiviert das Zuweisungselement* (t, b), wenn sie jede Eingangsstelle $s \in {}^\bullet t$ mit der Zuweisung der zugehörigen Kante markiert, d.h.

$$BM(s) = (1,0), \text{ wenn } b(x(s,t)) = 0,$$

   und

$$BM(s) = (0,1), \text{ wenn } b(x(s,t)) = 1.$$

3. Wenn das unter der Booleschen Markierung BM_{pre} aktivierte Zuweisungselement (t, b) eintritt, erzeugt es die direkte Folgemarkierung BM_{post}, unter der

   - jede Eingangsstelle $s \in {}^\bullet t \setminus t^\bullet$ unmarkiert ist, d.h.

$$BM_{post}(s) = (0,0),$$

- jede Ausgangsstelle $s \in t^\bullet$ mit ihrer Zuweisung markiert ist, d.h.

$$BM_{post}(s) = (1,0), \text{ wenn } b(x(t,s)) = 0,$$

  und

$$BM_{post}(s) = (0,1), \text{ wenn } b(x(t,s)) = 1.$$

- und die Markierung aller übrigen Stellen unverändert ist, d.h. für alle $s \notin {}^\bullet t \cup t^\bullet$:

$$BM_{post}(s) = BM_{pre}(s).$$

Man schreibt „$BM_{pre}[(t,b) > BM_{post}$“ und sagt: Die Markierung M_{post} ist *direkt erreichbar* von M_{pre} aus durch das Eintreten des Zuweisungselementes (t,b).

□

Definition 2.9 [Erreichbare Netzmarkierungen] Es sei $BN = (N; x, g)$ ein starkzusammenhängendes Boolesches Netz.

1. Es seien BM_{pre} und BM_{post} zwei Boolesche Markierungen von BN. Dann heißt BM_{post} *erreichbar von* BM_{pre} *aus*, wenn

   - eine endliche Folge $(t_j, b_j), j = 0, \ldots, n-1$, von Zuweisungselementen
   - und eine endliche Folge von Booleschen Markierungen $BM_j, j = 0, \ldots, n$, existieren mit

$$BM_j[(t_j, b_j) > BM_{j+1}, j = 0, \ldots, n-1,$$

   wobei $BM_{pre} = BM_0$ und $BM_{post} = BM_n$.

2. Nach Auszeichnung einer Anfangsmarkierung BM_0, d.h. für das Boolesche Netzsystem $BNS = (BN; BM_0)$, heißt eine Markierung BM *erreichbar*, wenn sie von BM_0 aus erreichbar ist. Man bezeichnet mit $[BM_0 >$ die *Menge aller erreichbaren Markierungen* von BNS.

3. Eine Boolesche Markierung BM heißt *Heimatmarkierung eines Booleschen Netzsystems* $BNS = (BN; BM_0)$, wenn BM erreichbar ist von jeder erreichbaren Markierung von BNS aus.

□

Beispiel 2.10 Das Beispiel in Abbildung 2.1 zeigt ein Boolesches Netzsystem $BNS = (BN; BM_0)$ mit einem Booleschen Netz $BN = (N; x, g)$ und das unterliegende Stellen/Transitionssystem $NS = (N; M_0)$. ◇

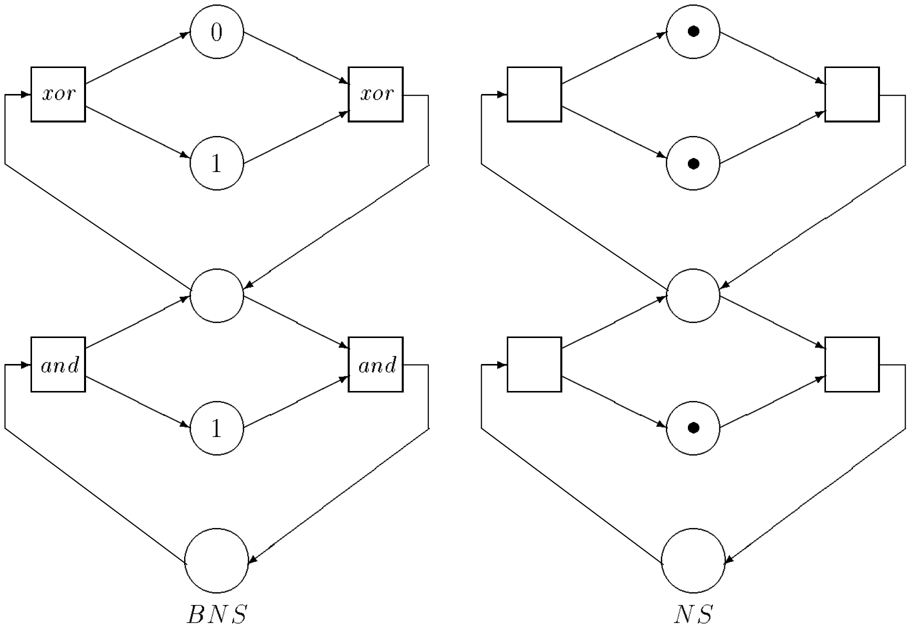


Abbildung 2.1: Boolesches Netzsystem BNS und unterliegendes Stellen/Transitionssystem NS

Kapitel 3

Übersetzungsregeln für EPKs

> „Die EPK-Methode basiert im wesentlichen auf der Petri-Netz-Theorie (...) und kann als eine Variante des Bedingungs-Ereignisnetzes, welche um logische Verknüpfungsoperatoren erweitert wurde, verstanden werden".[SNZ95, S. 428]

Wie im Einzelnen soll man nun die Netzelemente einer EPK in die Netzelemente eines Petri-Netzes übersetzen? Ja, wir haben sogar die Frage gehört: Soll man überhaupt eine EPK in ein Petri-Netz übersetzen?
Wir übersetzen EPKs in Petri-Netze, um die Petri-Netze anschließend einer Netzanalyse zu unterziehen, und um EPKs mit korrektem Verhalten simulieren zu können. Zur Zeit gibt es im Software Engineering den Trend, Prozesse nicht nur zu modellieren, sondern die modellierten Prozesse dann auch zur Steuerung von Abläufen (Workflow) einzusetzen. Es ist geplant, formale Modelle über eine normierte Schnittstelle in die Ablaufsteuerung einfließen zu lassen und sie dort durch die Workflow Engine auszuführen. Spätestens an dieser Stelle werden eventuelle Fehler in den Modellen dann zutage treten.
Hinter obiger Frage nach dem Sinn einer Übersetzung steht die weitergehende Frage, mit welcher Präzision Geschäftsprozesse modelliert werden sollen? Unsere Antwort hierauf ist einfach: Alles, was man modelliert, muß richtig sein, aber man muß nicht alles modellieren.
Eine gewisse Vagheit zu Beginn kann durchaus gewünscht sein. Aber nur in dem Sinne, daß man Details wegläßt, nicht in dem Sinne, daß man die Unterschiede der verschiedenen logischen Konnektoren verwischt. Das Prinzip der Hierarchisierung erlaubt, Details erst in einem späteren Schritt anzubringen. Am Anfang eines Projektes zur Prozeßmodellierung empfiehlt es sich, nur die EPK der obersten Ebene zu modellieren. Die Funktionen dieser EPK werden dann im weiteren Projektverlauf mit eigenen EPKs verfeinert.
Einem weitergehenden Plädoyer für generelle Vagheit können wir uns nicht anschließen. Vielmehr werden wir auf ein solches Plädoyer mit der Aufforderung antworten, ein quantitatives Maß für Vagheit anzugeben. Gerade die Mathematik ist in der Lage, auch Vagheit in einem präzisen Modell abzubilden.

Bemerkung 3.1 [Heuristik der Übersetzung]

- Randereignis: Die Anfangsereignisse unter den Randstellen der EPK werden mit einem ausgezeichneten Ereignis „Start/Ziel" der EPK verbunden, und ebenso werden die Zielereignisse unter den Randstellen mit diesem Ereignis verbunden. Hierdurch wird die EPK zu einem Zyklus geschlossen.

- Ereignis: Jedes Ereignis einer EPK wird auf eine Stelle des Petri-Netzes abgebildet. Diese Stelle wird als Bedingung mit den drei möglichen Werten „erfüllt“, „nicht erfüllt“ und „undefiniert“ aufgefaßt.

- Markentyp: Naheliegend ist ein Ansatz, bei dem man den Kontrollfluß als einen Fluß verschiedener Aktivierungsmarken durch das Netz modelliert: Es werden genau diejenigen Stellen markiert, an denen sich der Kontrollfluß gerade befindet. Falls mehrere Teilprozesse nebenläufig stattfinden, können gleichzeitig verschiedene Netzteile markiert sein. Dann enthält das Netz mehrere Aktivierungsmarken. Bei diesem Ansatz mit einer einzigen Markenart stößt man jedoch bei den schließenden *or*- und *xor*-Konnektoren auf folgendes Problem: Wann soll der schließende Konnektor schalten?

  Schaltet er bereits beim Erhalt der ersten Aktivierungsmarke, so würde er ein zweites Mal schalten, wenn er eine zweite Marke erhält. Im Falle, daß bei einem schließenden *or*-Konnektor zwei Marken gleichzeitig eintreffen, schaltet der Konnektor dagegen nur ein einziges Mal. Das Modell würde hier ein Zeitverhalten besitzen, das nicht dem realen Sachverhalt entspricht. Denn die vorliegenden EPKs modellieren stets Sachverhalte, bei denen nur ein einziges Mal geschaltet werden soll.

  Daher schlagen wir einen anderen Ansatz vor. Dieser Ansatz stellt sicher, daß ein schließenden Konnektor erst dann schalten kann, wenn an allen Eingängen eine Information vorliegt. Und zwar besagt diese Information entweder, daß

  - das Ereignis eingetreten ist,

  oder sie besagt, daß

  - das Ereignis nicht eintreten kann.

  Diesen Unterschied modellieren wir durch zwei verschiedene Markenarten: Die Information, daß ein Ereignis eingetreten ist, wird durch eine Marke mit dem Wert „1 = Aktiviert“ modelliert, die Information, daß ein Ereignis nicht eintreten kann, durch eine Marke mit dem Wert „0 = Deaktiviert“. Ebenso wie die „1“-Marken müssen auch die „0“-Marken durch das Netz weitergeschaltet werden. Der Informationsfluß zerfällt in zwei Komponenten, eine aktivierende und in eine deaktivierende Komponente. Und die nicht markierten Stellen bedeuten die Abwesenheit des Kontrollflusses.

- Initialisierung: Zur Initialisierung wird das Ereignis „Start/Ziel“ mit der Marke „1“ markiert.

- Konnektor: Viele Beispiele von EPKs enthalten nicht nur binäre logische Konnektoren, sondern auch Konnektoren mit $n > 2$ Ein- oder Ausgängen. Man kann Konnektoren nun sowohl auf Ebene der EPKs, als auch auf Ebene der resultierenden Petri-Netze zerlegen bzw. zusammenfassen.

  Wir entscheiden uns im folgenden dafür, die Konnektoren einer gegebenen EPK zunächst vollständig in binäre Konnektoren zu zerlegen. Nach der Übersetzung dieser EPK in ein Petri-Netz und erfolgreicher Netzanalyse können die binären Booleschen Transitionen dann mit einer Fusionsregel (Definition 4.5) zu mächtigeren Booleschen Transitionen verschmolzen werden.

  Die resultierenden binären Konnektoren einer EPK bedeuten nun entweder Beginn und Ende einer Schleife — wenn die EPK innere Schleifen enthält. Oder sie bedeuten eine Verzweigung bzw. Zusammenführung eines Prozesses und seiner Teilprozesse. Im ersten Fall wird der logische Konnektor einer EPK in eine Stelle, im zweiten Fall in eine Transition des Booleschen Netzes übersetzt. Wir behandeln die Modellierung von inneren Schleifen in einem eigenen Kapitel 5 behandelt.

- Schaltregel: Wenn man den Kontrollfluß als Fluß von zwei verschiedenartigen Marken modelliert, so müssen die Schaltregeln nicht nur den Fluß der Marken mit dem Wert „1", sondern auch den Fluß der Marken mit dem Wert „0" steuern. Transitionen müssen auch dann schalten können, wenn alle ihre Eingänge mit „0" markiert sind. Wenn die Transition in diesem Fall schaltet, legt sie auf alle ihre Ausgänge die Marke „0". Schon das einfache Beispiel einer *xor*-Alternative mit einem öffnenden und einem schließenden *xor*-Konnektor zeigt: Auch die nicht aktivierte Alternative muß schließlich an ihrem Eingang des schließenden *xor*-Konnektors eine Marke beisteuern, und zwar eine „0"-Marke. Daher müssen alle Transitionen dieser Alternative diese „0"-Marke zuvor weiter gereicht haben.

  In dem Beispiel von Abbildung 1.2 muß z.B. der schließende *xor*-Konnektor K_{50} auch dann schalten, wenn seine beiden Eingänge mit einer -Marke belegt sind; und zwar muß der Konnektor dann eine „0"-Marke auf seinem Ausgang erzeugen. Wenn beide Eingänge dagegen gleichzeitig mit einer „1"-Marke belegt sind, was nur bei einem Modellierungsfehler auftreten kann, soll der Konnektor nicht schalten und keine Marke auf seine Ausgangsstelle legen. Der Konnektor blockiert und kann nicht mehr schalten. Das wird bei Terminierung des Prozesses daran erkannt, daß an den Eingängen des Konnektors Ereignisse markiert sind, die nicht zu den erwünschten Endereignissen der EPK gehören.

  Durch die Einführung der Marken mit dem Wert „0" und die zugehörigen Schaltregeln der logischen Konnektoren erhält die EPK nach ihrer Übersetzung in ein Petri-Netz eine Semantik. Auf die Übersetzung folgt dann eine Netzanalyse, die u. a. das Verhalten des Netzes prüft. Falls das Netz diese Prüfung erfolgreich besteht — wir sprechen dann von einem „ordentlichen" Netzverhalten —, kann auf die „0"-Marken verzichtet werden: Ein ordentliches Netzverhalten läßt sich allein durch den Fluß der „1"-Marken beschreiben. In diesem Sinne kann man die „0"-Marken als ein technisches Hilfsmittel ansehen, um den logischen Konnektoren einer EPK eine Semantik zu geben und das Netzverhalten auf Fehler zu analysieren. Wir werden die Einzelheiten der Netzanalyse in Kapitel 7 ausführen.

- Funktion: Funktionen sind aktive Elemente einer EPK und werden daher auf Transitionen des Petri-Netzes abgebildet. Notwendig zur Aktivierung einer Funktions-Transition ist die Markierung ihrer Vorbedingung mit einer „0"-Marke oder einer „1"-Marke. Die jeweilige Marke wird beim Schalten der Transition von der Eingangsstelle auf die Ausgangsstelle weitergereicht. Das Schalten einer Funktions-Transition, deren Vorbedingung als „falsch" markiert ist, bedeutet, daß die zugehörige Aktivität in der Realität nicht ausgeführt, sondern übersprungen wird.

- Prozeßwegweiser: Sie werden analog zu Funktionen behandelt.

$\diamond$

Definition 3.2 [Übersetzungsregel für eine EPK] Eine vorgegebene EPK wird in folgenden Schritten in ein Boolesches Netz BN übersetzt:

1. Die EPK wird in die Syntax $EPK = (V, A)$ von Definition 1.4 gebracht. Dabei werden n-stellige Konnektoren, $n > 2$, in eine Folge binärer Konnektoren aufgelöst.

2. Es wird ein neues ausgezeichnetes Ereignis „Start/Ziel“ der EPK eingeführt. Genau diejenigen Kombinationen von Startereignissen der EPK, die der Modellierer als zulässig ansieht, werden mit logischen Konnektoren von diesem Ereignis „Start/Ziel“ abgeleitet. Und analog werden genau die zulässigen Kombinationen von Zielereignissen mit logischen Konnektoren an dieses Ereignis herangeführt.

3. Die EPK wird nach folgender Ersetzungsregel in ein Stellen/Transitionsnetz abgebildet:

   - Ereignisse werden zu Stellen.
   - Funktionen und Prozeßwegweiser werden zu Transitionen (Funktions-Transition).
   - Konnektoren, an denen Schleifen beginnen oder enden, werden zu Stellen (Angelpunkt).
   - Alle übrigen Konnektoren werden zu Transitionen (Konnektor-Transition).
   - Kanten werden zu Kanten.
   - Zwischen zwei benachbarte Stellen wird jeweils eine zusätzliche Funktions-Transition, und zwischen zwei benachbarte Transitionen wird jeweils eine zusätzliche Stelle eingefügt.

4. Das Stellen/Transitionsnetz wird beschriftet und zu einem Booleschen Netz BN erweitert:

   - Jede Konnektor-Transition des Netzes hat drei Kanten. Die beiden gleichgerichteten werden mit den Variablen x bzw. y beschriftet, die dritte Kante mit der Variablen z. Eine Konnektor-Transition erhält die Guard-Formel:
   $$(x \ or \ y \ or \ z) \Rightarrow [(x \, op \, y) \ and \ z],$$
   wobei der logische Operator $op \in \{ \ or \ , \ and \ , \ xor \ \}$ auf der rechten Seite der Formel identisch ist mit dem logischen Operator des Konnektors.
   - Jede Funktions-Transition des Netzes hat genau eine eingehende und genau eine ausgehende Kante. Die eingehende Kante wird mit der Variablen x, die ausgehende Kante mit der Variablen z beschriftet. Die Funktions-Transition erhält die Guard-Formel
   $$x \Leftrightarrow z.$$

□

Bemerkung 3.3 [Semantik der EPKs]

1. Die in der Literatur auftretenden EPKs enthalten keine Aussage darüber, ob die EPK ein zweites Mal gestartet werden kann, solange der erste Prozeß noch nicht sein Ziel erreicht hat. Wenn ja, dürfen sich dann zwei Prozesse auf einer gegebenen EPK überholen? Wir haben auf diese Frage aus den Beispielen und aus der Literatur keine Antwort gefunden. Daher erlauben wir uns, diese Mehrdeutigkeit durch obigen Vorschlag dahingehend zu entscheiden, daß wir für einen Prozeß

   - keine mehrfache Aktivierung mit gegenseitiger Wechselwirkung, wohl aber
   - mehrfache, von einander unabhängige Aktivierungen

zulassen. In einer objektorienterten Sprechweise ist ein Prozeß eine Klasse. Von einer Klasse können mehrere Instanzen erzeugt werden, die voneinander unabhängig sind. Durch die Einführung des Ereignisses „Start/Ziel“ und die Anbindung der fachlichen Start- bzw. Zielereignisse wird sichergestellt:

- Jeder individuelle Prozeß beginnt mit einer fachlich korrekten Kombination von Startereignissen.
- Wenn der Prozeß das Ereignis „Start/Ziel“ wieder erreicht, so hat er zuvor eine fachlich korrekte Kombination von Zielereignissen durchlaufen.

In diesem Sinne ist die Einführung des ausgezeichneten Ereignisses „Start/ Ziel“ ein technisches Mittel, um die aus fachlicher Sicht erwünschten Start- bzw. Zielereignisse zusammenzufassen. Darüberhinaus ermöglicht der Übergang zu stark-zusammenhängenden Netzen die Behandlung von EPKs im Rahmen der klassischen Theorie der Free-Choice Netze.

2. Der hierachischen Gliederung eines Prozesses durch einen Baum von EPKs entspricht auf der Seite der Petri-Netze eine Transitionsverfeinerung. In der Praxis verfeinert man eine EPK, indem man die Ereignisse in der Nachbarschaft der zu verfeinernden Funktion als Randereignisse in der verfeinernden EPK wiederholt. Bei der Auflösung der Verfeinerung werden diese Randereignisse der tieferen Ebene mit den korrespondierenden Ereignissen der höheren Ebene identifiziert. Wir werden uns in dieser Arbeit jedoch auf die EPKs einer einzigen Ebene beschränken.

3. In Kapitel 7 werden wir das resultierende Boolesche Netz BN zunächst einer statischen Netzanalyse unterwerfen (Definition 7.1). Nach erfolgreicher Analyse erhält das Netz BN die Boolesche Anfangsmarkierung BM_0, welche genau die Stelle „Start/Ziel“ mit der Marke „1“ belegt und alle anderen Stellen unmarkiert läßt. Wir erhalten ein Boolesches Netzsystem

$$BNS = (BN; BM_0).$$

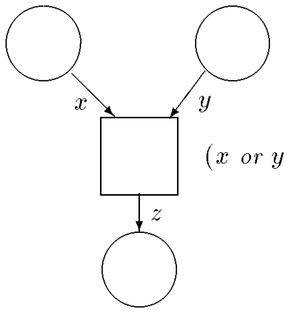
Durch die Wahl der Booleschen Anfangsmarkierung BM_0 erhält das Netzsystem BNS eine Dynamik, bei der verschiedene Ereignisfolgen (occurrence sequences) ablaufen können. Es ist dann Aufgabe der dynamischen Netzanalyse (Definition 7.9), das Boolesche Netzsystem BNS auf ein ordentliches Verhalten (well behavedness) zu prüfen.

$\diamond$

Bemerkung 3.4 [Elementare Boolesche Netze und Transitionen]

1. Die Übersetzungsregel (Definition 3.2) faßt jeweils zwei EPKs in der Aufzählung von Bemerkung 1.8 zu einem einzigen Fall auf seiten des Netzes zusammen. Die Anzahl der zu betrachtenden Fälle wird damit von 12 auf 6 reduziert. Im einzelnen werden paarweise die Fälle 1 und 10, 2 und 11, 3 und 12, 4 und 7, 5 und 8 sowie 6 und 9 zusammengefaßt. Durch Inversion der Kanten unter Beibehaltung des Guards kann man dabei die letzten drei Fälle in die ersten drei überführen, so daß von ursprünglich zwölf nur drei wesentlich verschiedene Fälle übrig bleiben. Ihnen entsprechen die folgenden „elementaren“ Booleschen Netze:

Fall 1, 10:

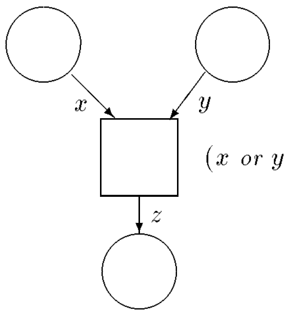


$(x\ or\ y\ or\ z) \Rightarrow (x\ xor\ y)\ and\ z$

Zuweisungselemente:

x	y	z
1	0	1
0	1	1
0	0	0

Fall 2, 11:

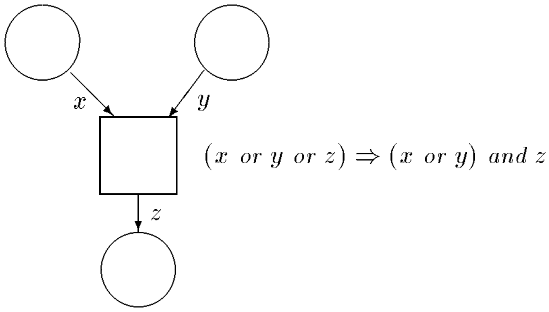


$(x\ or\ y\ or\ z) \Rightarrow (x\ and\ y)\ and\ z$

Zuweisungselemente:

x	y	z
1	1	1
0	0	0

Fall 3, 12:

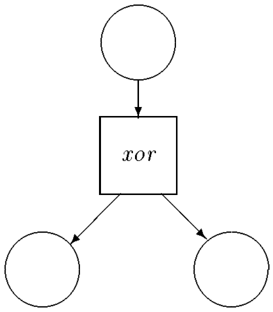


Zuweisungselemente:

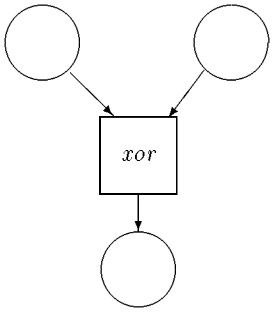
x	y	z
1	0	1
0	1	1
1	1	1
0	0	0

2. Durch unsere Übersetzungsregeln haben wir den logischen Konnektoren einer EPK eine Semantik gegeben, wie sie bei der Behandlung paralleler Prozesse üblich ist. Wir beziehen uns im folgenden auf die Fallunterscheidung von Bemerkung 1.8.

   (a) Wir nennen die Boolesche Transition von Fall 4 und 7

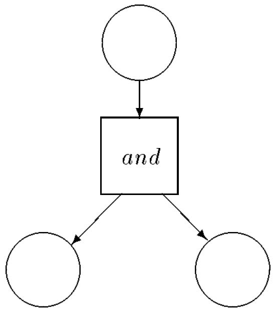
   

   eine *Branch-Transition*, und die Boolesche Transition von Fall 1 und 10

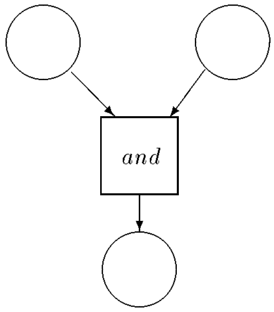
   

   eine *Merge-Transition*. An einer Branch-Transition aktiviert der Kontrollfluß genau eine der beiden Alternativen und deaktiviert die jeweils andere. Eine Merge-Transition synchronisiert eine aktivierte und eine deaktivierte Alternative.

(b) Wir nennen die Boolesche Transition von Fall 5 und 8

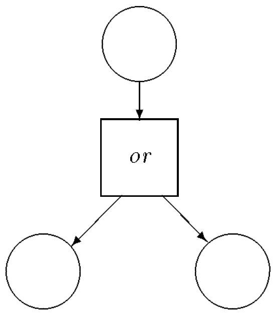


eine *Fork-Transition*, und die Boolesche Transition von Fall 2 und 11

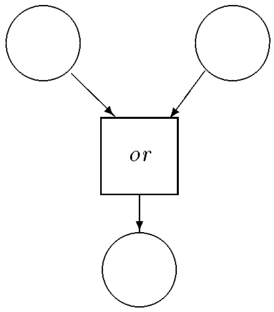


eine *Join-Transition*. An einer Fork-Transition aktiviert der Kontrollfluß beide Alternativen, und eine Join-Transition synchronisiert zwei aktivierte Alternativen.

(c) Wir nennen die Boolesche Transition von Fall 6 und 9

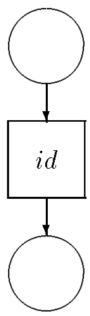


eine *öffnende or -Transition*, und die Boolesche Transition von Fall 3 und 12



eine *schließende or-Transition*. An einer öffnenden *or*-Transition aktiviert der Kontrollfluß mindestens eine der beiden Alternativen und deaktiviert die ggf. übrigbleibende andere Alternative. Eine schließende *or*-Transition synchronisiert zwei Alternativen mit beliebigem Aktivierungsverhalten.

(d) Wir nennen die Boolesche Transition, die aus der Übersetzung einer Funktion oder eines Prozeßwegweisers resultiert,



die *identische Boolesche Transition*. Die identische Boolesche Transition verändert den Kontrollfluß nicht.

3. Wir hatten in Bemerkung 1.8 bei verschiedenen Fällen eine Reihe von Fragen offen gelassen über die Bedeutung des schließenden Konnektors einer EPK. Diese Fragen haben wir durch unsere Übersetzung in ein Boolesches Netz nun folgendermaßen beantwortet: Der schließende Konnektor einer EPK kann erst dann schalten, wenn beide Eingangsereignisse markiert sind, d.h. sobald man über jedes der beiden Ereignisse Bescheid weiß. Solange lediglich eines der beiden Ereignisse markiert ist, kann der Konnektor nicht schalten. Vielmehr muß er in dieser Situation auf die Information warten,

   - ob entweder auch das andere Ereignis eingetreten ist
   - oder ob das andere Ereignis nicht eintreten kann.

   Trifft eine Marke „1“ ein, so weiß man, daß das andere Ereignis jetzt eingetreten ist. Dagegen bedeutet das Eintreffen einer Marke „0“, daß das andere Ereignis nicht eintreten kann. Der Konnektor kann also in jedem Fall höchstens ein einziges Mal schalten, da er zuvor die beiden Teilprozesse synchronisieren muß. Sobald die Information über beide Ereignisse vorliegt, schaltet der Konnektor, in Fall 1 z. B. in folgenden beiden Situationen:

   - Eines der beiden Ereignisse ist eingetreten (Marke „1“), das andere Ereignis kann nicht eintreten (Marke „0“).
   - Keines der beiden Ereignisse kann eintreten (Zweimal Marke „0“).

   Im ersten Fall wird am Ausgang eine Marke „1“ erzeugt, im zweiten Fall eine Marke „0“. In allen anderen Situationen tritt ein Deadlock auf, das System ist an diesem Konnektor blockiert.

◇

Definition 3.5 [Aktivierungstreue]

1. Eine Transition t eines Booleschen Netzes

   - mit Variablen $x = (x_1, \dots, x_n)$ der Eingangskanten
   - und mit Variablen $z = (z_1, \dots, z_m)$ der Ausgangskanten

heißt *aktivierungstreu*, wenn für jedes Zuweisungselement $(t, (b_{in}, b_{out}))$ gilt:

$$b_{in} = 0 \Leftrightarrow b_{out} = 0,$$

$b_{in} := (b(x_1), \ldots, b(x_n)), b_{out} := (b(z_1), \ldots, b(z_m))$.

2. Ein Boolesches Netz heißt *aktivierungstreu*, wenn jede seiner Transitionen aktivierungstreu ist.

□

Aktivierungstreue Transitionen mit „0"-Marken an allen Eingängen können also keine „1"-Marke an einem ihrer Ausgänge erzeugen, und ebenso können sie nicht alle „1"-Marken an ihren Eingängen in „0"-Marken an ihren Ausgängen umwandeln. Nach Bemerkung 3.4 ist jedes Boolesche Netz, das aus der Übersetzung einer EPK resultiert, aktivierungstreu.

Bemerkung 3.6 [Deadlock]

1. Bei Booleschen Netzen, die aus der Übersetzung einer EPK resultieren, können an den Merge- und Join-Transitionen Deadlock-Situationen auftreten. Wenn ein Netzsystem einen Deadlock zuläßt, so betrachten wir das als einen Modellierungsfehler. Die Möglichkeit, eine EPK auf Deadlock-Situationen zu überprüfen, schafft daher ein zusätzliches Mittel der Netzanalyse.

2. Wenn man auf die Blockade durch eine Deadlock-Situation verzichten will, kann man in Teil 4 von Definition 3.2 eine andere Guard-Formel wählen und den logischen Konnektor einer EPK in eine solche Boolesche Transition des Netzes übersetzen, die zu jedem Eingang mit Hilfe des logischen Operators des Konnektors einen Ausgang berechnet. Man verwendet dann

   - statt der Join-Transition eine schließende *and*-Transition
   - und statt der Merge-Transition eine schließende *xor*-Transition.

   Dementsprechend ist in Definition 3.2, Teil 4, die Guard-Formel

   $$(x \text{ } or \text{ } y \text{ } or \text{ } z) \Rightarrow [(x \, op \, y) \text{ } and \text{ } z]$$

   durch folgende Alternative zu ersetzen:

   $$z \Leftrightarrow (x \, op \, y).$$

   Während der Netzsimulation werden dann „fehlerhafte" Zuweisungselemente protokolliert, und im Anschluß an einen gesamten Durchlauf können Simulationen zur Prüfung ausgewählter Eingangsmarkierungen durchgeführt werden.

◇

Bemerkung 3.7 [Aktivierung von Booleschen Transitionen]

1. Eine Guard-Formel wie z.B.

   $$(x \text{ } or \text{ } y \text{ } or \text{ } z) \Rightarrow [(x \text{ } xor \text{ } y) \text{ } and \text{ } z]$$

   die aus den Fällen 1, 4, 7 und 10 von Bemerkung 1.8 entsteht, bedeutet, daß genau diejenigen Zuweisungen

$$(x, y, z) \in Boole^3$$

die Transition aktivieren können, für die obige Formel wahr ist. Das sind offensichtlich die drei Zuweisungen

$$(x, y, z) \in \{(1, 0, 1), (0, 1, 1), (0, 0, 0)\}.$$

Falls die Eingangsstellen dann Marken tragen, die den jeweiligen Kantenbeschriftungen entsprechen, ist die Transition aktiviert.

2. Man beachte, daß in Definition 3.2 alle Guard-Formeln von Konnektor-Transitionen im Fall $(x, y, z) = (0, 0, 0)$ erfüllt sind, und zwar unabhängig von der Konklusion auf der rechten Seite

$$[(x\, op\, y)\ and\ z],\ op \in \{\ or\ ,\ and\ ,\ xor\ \}.$$

◇

Beispiel 3.8 Wir führen an der EPK der Beschaffungslogistik die einzelnen Schritte nach Definition 3.2 vor. Abbildung 1.2 enthält eine EPK gemäß Definition 3.2. Die Übersetzung in das Boolesches Netz von Abbildung 3.1 geschieht in folgenden Schritten:

1. Auflösen von n-stelligen logischen Konnektoren in binäre Konnektoren:

   - Aus dem *or*-Konnektor K_1 werden zwei binäre *or*-Konnektoren K_{103} und K_{104},

   und

   - aus dem *and*-Konnektor K_9 werden zwei binäre *and*-Konnektoren K_{91} und K_{92}.

2. Einfügen des Ereignisses „Start/Ziel“:

   Wir haben das zusätzliche Ereignis „Start/Ziel“ eingeführt. Die bisherigen Startereignisse E_1, E_2, E_3 und E_{13} werden mit dem Ereignis „Start/Ziel“ durch die zusätzlichen Konnektoren K_{100}, K_{101} und K_{102} verbunden. Dabei sind die Ereignisse E_1, E_2 und E_3 als *or*-Alternativen untereinander gleichgeordnet. Diese *or*-Alternative wird dem weiteren Startereignis E_{13} in Form einer *and*-Alternative parallelgestellt.

   Analog werden die bisherigen Zielereignisse E_{20} und E_{21} mit dem Ereignis „Start/Ziel“ über den Konnektor $K200$ verbunden. Die beiden Zielereignisse E_{20} und E_{21} werden als zwei Äste einer *and*-Alternative gleichgestellt.

3. Übersetzen von logischen Konnektoren in Boolesche Transitionen:

   Die EPK enthält keine inneren Schleifen. Daher werden alle logischen Konnektoren in die entsprechende Boolesche Transition übersetzt.

4. Übersetzen von Ereignissen in Stellen.

5. Übersetzen von Funktionen in identische Boolesche Transitionen.

6. Prozeßwegweiser treten nicht auf.

7. Übernahme der Kanten.

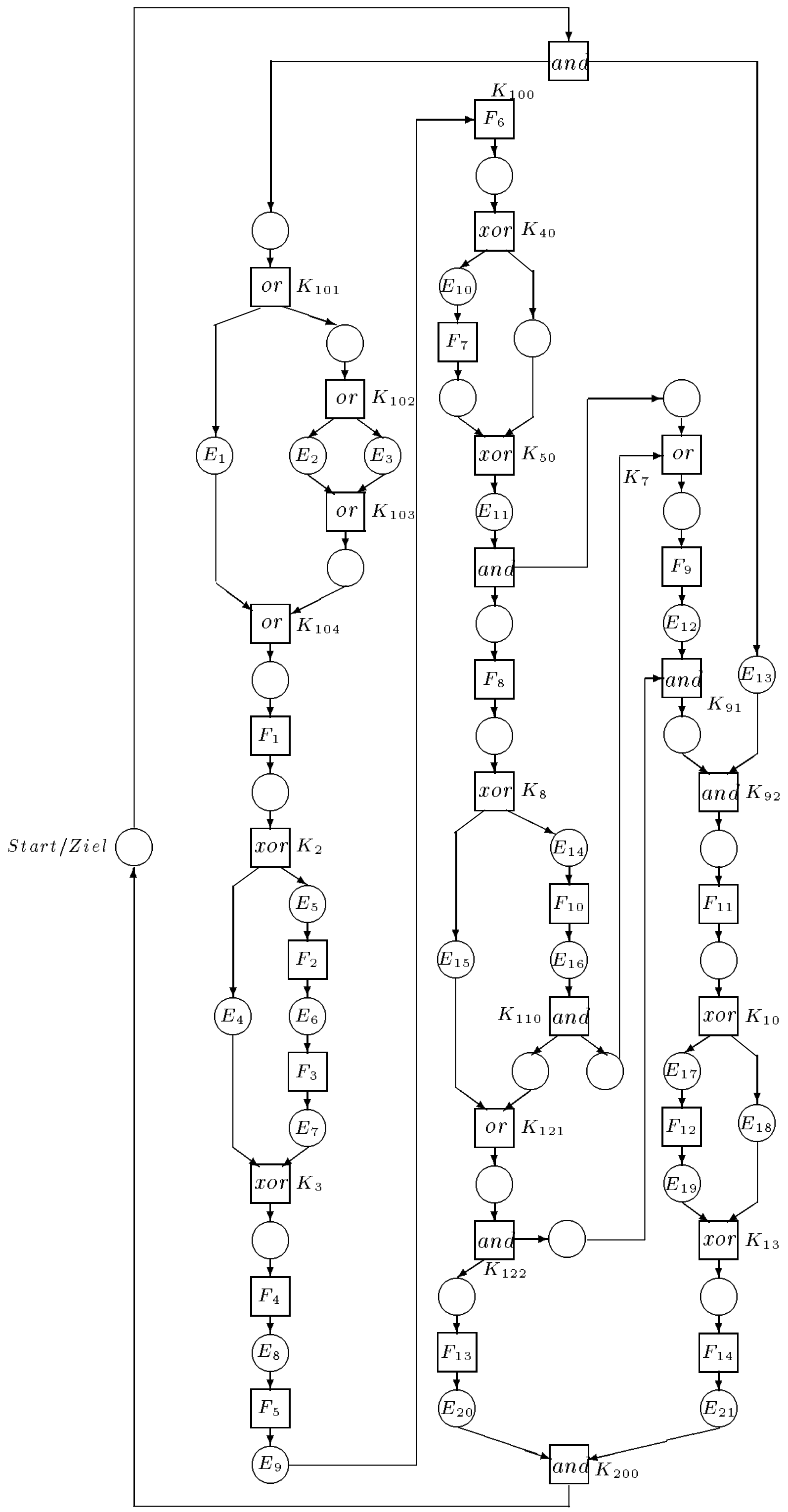


Abbildung 3.1: Boolesches Netz Beschaffungslogistik

8. Einfügen von Ereignissen: An verschiedenen Punkten wurden weitere Ereignisse eingeführt.

◇

Bemerkung 3.9 Seit der Erfindung der EPKs durch Keller, Nüttgens und Scheer im Jahre 1991 sind in Deutschland eine Reihe von Arbeiten zum Bezug von EPKs und Petri-Netzen erschienen. Speziell zum Thema der Übersetzung von EPKs in Petri-Netze haben wir von folgenden zwei Arbeiten Kenntnis erhalten:

1. In der Arbeit [CS94] aus dem Jahre 1994 stellen sich Chen und Scheer die Aufgabe, EPKs in gefärbte Petri-Netze zu übersetzen. Und zwar übersetzen sie

   - eine Funktion oder ein Ereignis der EPK in eine Stelle des Petri-Netzes und
   - einen logischen Konnektor der EPK in eine Transition bzw. in ein Teilnetz des Petri-Netzes.

   Alle Stellen des resultierenden Petri-Netzes erhalten die Kapazität 1. Die Autoren betonen, daß gefärbte Petri-Netze die Unterscheidung verschiedener Markenarten erlauben und setzen diese Unterscheidung bei der Modellierung der *or*-Konnektoren ein.

   Bei der Übersetzung der logischen Konnektoren einer EPK gehen die Autoren im einzelnen wie folgt vor:

   Der *and*-Konnektor einer EPK wird in eine Transition ohne Guard-Funktion übersetzt. An dieser Stelle wird noch einmal deutlich, daß Transitionen in elementaren Petri-Netzen „von Haus aus" das Schaltverhalten eines *and*-Konnektors besitzen.

   Ein schließender *xor*-Konnektor der EPK wird in zwei Transitionen mit jeweils einem Ein- und Ausgang übersetzt, wobei beide Ausgänge an der nachfolgenden Stelle zusammengeführt werden. Durch die Beschränkung der Stellenkapazität auf 1 wird sichergestellt, daß nicht gleichzeitig über beide Eingänge der Stelle eine Marke fließen kann. Das Schaltverhalten des *xor*-Konnektors der EPK wird auf der Ebene des Petri-Netzes also durch die Beschränkung der Stellenkapazität nachgebildet.

   Analog zum schließenden wird ein öffnender *xor*-Konnektor übersetzt. Dabei wird wieder eine grundlegende Eigenschaft ausgenutzt, welche die Stellen eines elementaren Petri-Netzes besitzen: Jede Marke einer Stelle kann nur über höchstens einen Ausgang, aber nicht gleichzeitig über mehrere Ausgänge abfließen.

   Offensichtlich lassen sich die bisherigen logischen Konnektoren noch im Rahmen der B/E-Netze nachbilden. Für die Modellierung des *or*-Konnektors verwenden die Autoren nun typische Erweiterungen der gefärbten Petri-Netze. Sie betonen, daß einem schließenden *or*-Konnektor ein analoger öffnender *or*-Konnektor vorausgehen sollte und daß beide Konnektoren im selben Modus schalten sollten. Im Falle eines n-fachen Konnektors gibt es $2^n - 1$ verschiedene Modi, sie entsprechen denjenigen Belegungen, welche der Adjunktion der n Variablen den Wahrheitswert „wahr" geben. Die Transition des öffnenden *or*-Konnektors entnimmt nun beim Schalten aus einem Vorrat von $2^n - 1$ individuell gefärbten Marken diejenige Marke, welche dem aktuellen Modus entspricht und übermittelt diese Referenzmarke auf einem separaten Kanal an die Stelle, welche der schließenden Transition vorgelagert ist. Diese Stelle prüft, ob die Belegung ihrer n Eingänge mit der Referenzmarke übereinstimmt, und nur in diesem Fall ist die nachfolgende Transition aktiviert.

Die Autoren führen diese Übersetzungsregeln an einem Beispiel vor. Im vorletzten Abschnitt mit dem Namen „Validierung von Prozeßketten durch Netzanalyse unter dynamischen Aspekten" betonen sie die Notwendigkeit, die in der Praxis entstehenden Prozeßketten auf Erreichbarkeitseigenschaften, Deadlock-Situationen und tote Transitionen zu untersuchen. Die Methode ihrer Wahl ist dabei das Studium des Fallgraphen des aus der Übersetzung entstandenen Petri-Netzes. Zusammenfassend schreiben Chen und Scheer über die EPK-Methode:

> „Der Ansatz bietet nicht nur die Möglichkeit, bereits in frühen Entwurfsphasen die Prozeßketten mit einer graphenorientierten Beschreibungsmethode zu modellieren. Mit Hilfe der vielfältigen Analyseverfahren, die sich zur Validierung der modellierten Prozeßketten in diesem Ansatz einbinden lassen, kann auch der Übergang von vorwiegend verbalen Systembeschreibungen zu präzisen Modellen wesentlich erleichtert werden. Somit kann der Ansatz mit geringem Aufwand auch zur Validierung einer mit EPK modellierten Prozeßkette eingesetzt werden."

Wir stimmen Chen und Scheer an allen Stellen uneingeschränkt zu, wo sie die Bedeutung der Validierung einer vorliegenden EPK herausstellen. Auch wir betrachten die EPK-Validierung als eine unerläßliche Aufgabe für die praktische Verwendung dieser Methode. Ebenso stimmen wir mit Chen und Scheer überein, daß die Übersetzung in ein Petri-Netz das geeignete Mittel zu diesem Ziel ist. Genau aus diesen Überlegungen heraus haben wir die vorliegende Arbeit verfaßt.

Außerdem fordern wir mit Chen und Scheer, daß *or*-Konnektoren nur paarweise auftreten sollten: Jedem schließenden *or*-Konnektor soll ein analoger öffnender *or*-Konnektor vorausgehen. Dabei muß sichergestellt sein, daß beide Konnektoren einer *or*-Alternative im selben Modus schalten. Unsere Lösung dieser Anforderung (siehe Kapitel 6) unterscheidet sich jedoch von der von Chen und Scheer gewählten.

Insgesamt gesehen gibt es wesentliche Unterschiede zwischen unserem Vorgehen bei der Übersetzung von EPKs in Petri-Netze und dem Ansatz von Chen und Scheer in [CS94]. Die wichtigsten Unterschiede unseres Ansatzes sind:

- Modellierung der logischen Konnektoren durch Boolesche Transitionen
- Unterscheidung von aktivierenden und deaktivierenden Marken zum Zwecke der Netzanalyse
- Netzanalyse in Form eines Algorithmus und nicht als Studium des Fallgraphen (vgl. Kapitel 7).
- Schleifenmodellierung durch verzweigte Stellen.

Auch können wir uns in einzelnen Punkten der Behandlung der gefärbten Petri-Netze durch Chen und Scheer nicht anschließen. So kann u. E. die Synchronisation des schließenden und des öffnenden *or*-Konnektors nicht - wie von den Autoren vorgeschlagen — durch Kantenbedingungen an den Stellen geleistet werden. Stattdessen schlagen wir vor, Guard-Formeln an den Transitionen zu verwenden.

2. In [Brö96], seiner Diplomarbeit aus dem Jahre 1996, entwickelt Bröker Übersetzungsregeln, um EPKs in Pr/T-Netze zu übersetzen. Außerdem implementiert er diese Regeln in einem Programm zur maschinellen Übersetzung von EPKs aus dem ARIS-Toolset in Pr/T-Netze des Tools INCOME.[1]

   Bröker übersetzt

   - Ereignisse der EPK in Stellen des Petri-Netzes
   - Funktionen der EPK in Transitionen des Petri-Netzes
   - logische Konnektoren der EPK in Teilnetze des Petri-Netzes.

   Unproblematisch bei der Übersetzung der logischen Konnektoren sind dabei wieder die *and*-Konnektoren. Sie lassen sich 1:1 in Transitionen ohne Guard-Formel übersetzen.

   Zur Übersetzung der *xor*-Konnektoren werden zusätzliche Stellen und Transitionen eingeführt. Das Schaltverhalten der zusätzlichen Transitionen unterliegt einer Prioritätenregelung, um Eingangsmarkierungen, die nicht zulässig sind, abzufangen. Durch die Einführung der Prioritätenregelung wird die Klasse der S/T-Netze verlassen und zu Pr/T-Netzen übergegangen. Als Alternative zur Prioritätenregelung behandelt der Autor die Übersetzung des *xor*-Konnektors mit einem zusätzlichen Speicher, der Kopien von allen in das System eingespeisten Marken enthält. An dieser Stelle entsteht dann ein Petri-Netz mit individuellen Marken. Für beide Fälle werden die Vor- und Nachteile der jeweiligen Lösung diskutiert.

   Ähnlich wie der *xor*-Konnektor wird auch der Fall des *or*-Konnektors behandelt. In allen Fällen werden die zusätzlichen Transitionen mit Regeln versehen, so daß Pr/T-Netze entstehen.

   Auch der von Bröker gewählte Ansatz zur Übersetzung unterscheidet sich an wesentlichen Punkten von unserem Vorgehen, die Unterschiede betreffen dieselben Stellen wie bei der Arbeit von Chen und Scheer [CS94].

$\diamond$

[1] INCOME ist ein Produkt der PROMATIS Informatik GmbH & Co. KG.

Kapitel 4

Boolesche Algebra

In diesem Kapitel rekapitulieren wir die algebraische Formulierung der Booleschen Logik, d.h. der klassischen zweiwertigen Logik. Wir benutzen die Resultate, um Fusionsregeln von Transitionen und Stellen zu beweisen.
In diesem Kapitel werden wir einige Hilfsmittel aus der Kommutativen Algebra verwenden, die über den in der Informatik gewohnten Rahmen hinausgehen. Der Leser, für den diese Begriffe neu sind, kann dieses Kapitel aber beim ersten Lesen überspringen. Es steht fast parallel zu den nachfolgenden Kapiteln, auf seine Resultate wird nur an wenigen Stellen zurückgegriffen. Als Referenz zur Kommutativen Algebra empfehlen wir z.B.[AM69].

Bemerkung 4.1 [Algebren über $\mathbb{F}_2$])

1. Grundlage der algebraischen Formulierung der Booleschen Aussagenlogik ist die Polynomalgebra über dem Körper

$$\mathbb{F}_2 = (\mathbb{F}_2, +, *).$$

   Der Körper $\mathbb{F}_2$ mit den beiden Elementen 0 und 1 ist der Körper der Restklassen bei Division durch 2. Dieser Körper ist isomorph zur Algebra

$$Boole = (\{0,1\},\ xor\ ,\ and\ ).$$

   Wir bezeichnen mit

$$\mathbb{F}_2[x_1, \ldots, x_n] = (\mathbb{F}_2[x_1, \ldots, x_n], +, *)$$

   die Algebra der Polynome in n Veränderlichen über dem Körper $\mathbb{F}_2$. Wenn die Anzahl der Variablen nicht explizit hervorgehoben werden soll, werden wir zur Abkürzung auch $\mathbb{F}_2[x]$ schreiben,

2. Die Algebra der Booleschen Funktionen

$$g : Boole^n \rightarrow Boole$$

   ist dann isomorph zur Quotientenalgebra

$$B(n) := \mathbb{F}_2[x_1, \ldots, x_n] / < x_1^2 - x_1, \ldots, x_n^2 - x_n > .$$

Die Elemente dieses Quotienten sind Polynome mit Koeffizienten aus $\mathbb{F}_2$ in den Veränderlichen $x_1, \ldots, x_n$, aber ohne quadratische Potenzen x_j^2, $j = 1, \ldots, n$. Man multipliziert zwei Polynome aus $B(n)$ nach der üblichen Produktformel für Polynome und berücksichtigt anschließend die Kongruenz $x_j^2 \equiv x_j$, $j = 1, \ldots, n$.

Die Auswertung

$$\epsilon : B(n) \to \prod_{x \in Boole^n} \mathbb{F}_2$$

$$g \mapsto (g(x))_{x \in Boole^n}$$

ist ein Isomorphismus der Algebra $B(n)$ auf ein Produkt von 2^n Exemplaren des Körpers $\mathbb{F}_2$. Dabei kann man die auf der rechten Seite stehenden Familien

$$(g(x))_{x \in Boole^n}$$

als die Funktionswerte einer Booleschen Funktion

$$Boole^n \to Boole, x \mapsto g(x),$$

ansehen.

Aus der Darstellung vermöge ϵ liest man folgende Eigenschaften von $B(n)$ ab:

- Jede Funktion $f \in B(n)$ ist bereits durch ihre Nullstellen festgelegt.
- Das Element $1 \in B(n)$ ist die einzige Einheit von $B(n)$.
- Jedes Element $f \in B(n)$ ist idempotent, d.h. es gilt $f^2 = f$.

3. Ein symmetrisches Polynom $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ ist ein Polynom, das unter jeder Permutation seiner Variablen invariant ist. Bekanntlich läßt sich f dann als Polynom in den elementar-symmetrischen Polynomen darstellen. Wir bezeichnen mit

$$\sigma_1, \ldots, \sigma_n \in B(n)$$

die Restklassen der elementar-symmetrischen Polynome in n Variablen und mit

$$\sigma := \sum_{j=1,\ldots,n} \sigma_j$$

ihre Summe. Jedes $\sigma_j, j = 1, \ldots, n$, ist ein Polynom vom Grad j:

$$\sigma_j(x_1, \ldots, x_n) = \sum_{1 \leq i_1 < \ldots < i_j \leq n} x_{i_1} * \ldots * x_{i_j},$$

z.B.

$$\sigma_1(x_1, \ldots, x_n) = x_1 + \ldots + x_n$$

$$\sigma_n(x_1, \ldots, x_n) = x_1 * \ldots * x_n.$$

Es gilt $\sigma(x) = 0 \Leftrightarrow x = 0$.

4. Die Aussagenlogik basiert in der algebraischen Darstellung auf den Polynomen für

   - die logischen Grund-Operatoren:

   $$\begin{array}{ll} or: & Boole^2 \to \mathbb{F}_2,\ or\,(x,y) := x+y+xy \\ and: & Boole^2 \to \mathbb{F}_2,\ and\,(x,y) := xy \\ xor: & Boole^2 \to \mathbb{F}_2,\ xor\,(x,y) := x+y \end{array}$$

   - die Negation $\neg x$:

   $$\neg : Boole \to \mathbb{F}_2, \neg(x) := 1+x$$

   - die Implikation $x \Rightarrow y$:

   $$\Rightarrow : Boole^2 \to \mathbb{F}_2, \Rightarrow (x,y) := 1 + x(y+1)$$

   und

   - die Äquivalenz $x \Leftrightarrow y$:

   $$\Leftrightarrow : Boole^2 \to \mathbb{F}_2, \Leftrightarrow (x,y) := 1+x+y.$$

   Unter Verwendung der elementar-symmetrischen Polynome in zwei Veränderlichen

   $$\sigma_1(x,y) = x+y \text{ und } \sigma_2(x,y) = x * y$$

   schreiben sich die symmetrischen unter obigen Operatoren als

   $$\begin{array}{ll} xor & = \sigma_1 \\ and & = \sigma_2 \\ or & = \sigma_1 + \sigma_2 \\ \Leftrightarrow & = 1 + \sigma_1. \end{array}$$

$\diamond$

Bemerkung 4.2 [Guard-Formeln als Polynome] Es sei $BN = (N; x, g)$ ein Boolesches Netz.

1. Die Guard-Formel einer Booleschen Transition t mit insgesamt n Eingangs- und Ausgangskanten, wird in der algebraischen Formulierung zu einem Polynom

   $$g_t \in B(n),$$

   und Variablenzuweisungen, welche die Guard-Formel erfüllen, sind die n-Tupel $b = (b_1, \ldots, b_n)$ aus $Boole^n$ mit

   $$g_t(b) = 1.$$

2. Mit der in Bemerkung 4.1, Teil 2 gegebenen Produktdarstellung ist jede Guard-Funktion aus $B(n)$ durch einen Bit-String der Länge 2^n charakterisiert. Damit lassen sich Boolesche Transitionen performant auf einem Computer implementieren.

3. Die Guard-Formel einer Konnektor-Transition t aus Definition 3.2

$$(x \textit{ or } y \textit{ or } z) \Rightarrow [(x \, op \, y) \textit{ and } z]$$

mit $op \in \{ \textit{ or }, \textit{ and }, \textit{ xor } \}$ ist in der algebraischen Darstellung das Polynom g_t aus $B(3)$ mit

$$g_t(x, y, z) := 1 + x + y + z + xy + yz + xz + xyz + \; op\,(x, y)z.$$

Spezialisiert man in dieser Guard-Formel den logischen Operator „op“ auf die Fälle von Bemerkung 1.8, erhält man

Fall 1, 10: $op = \textit{ xor}$

$$\begin{aligned} g_t(x,y,z) &:= 1 + x + y + z + xy + xyz \\ &= 1 + \sigma(x,y) + z + \sigma_2(x,y) * z. \end{aligned}$$

Fall 2, 11: $op = \textit{ and}$

$$\begin{aligned} g_t(x,y,z) &:= 1 + x + y + z + xy + yz + xz \\ &= 1 + \sigma(x,y) + z + \sigma_1(x,y) * z. \end{aligned}$$

Fall 3, 12: $op = \textit{ or}$

$$\begin{aligned} g_t(x,y,z) &:= 1 + x + y + z + xy \\ &= 1 + \sigma(x,y) + z. \end{aligned}$$

4. Die Guard-Formel einer Funktions-Transition t aus Definition 3.2

$$x \Leftrightarrow z$$

ist in der algebraischen Formulierung das Polynom

$$g_t(x, z) = 1 + x + z.$$

$\diamond$

Definition 4.3 [Fusion von Stellen] Es seien s_1 und s_2 zwei Stellen eines Booleschen Netzes $BN = (N; x, g)$, die zwischen zwei Transitionen t_{pre} und t_{post} liegen, d.h.

$$^\bullet s_1 = {}^\bullet s_2 = \{t_{pre}\} \text{ und } s_1^\bullet = s_2^\bullet = \{t_{post}\}.$$

Die zugehörigen Kanten seien beschriftet als

$$u_{1,pre} := x(t_{pre}, s_1) \text{ und } u_{2,pre} := x(t_{pre}, s_2)$$

und

$$u_{1,post} := x(s_1, t_{post}) \text{ und } u_{2,post} := x(s_2, t_{post}),$$

und in jedem Zuweisungselement

$$(t_j, b) \text{ von } t_j, j = pre, post,$$

möge die Variable $u_{1,j}$ denselben Wahrheitswert haben wie die Variable $u_{2,j}$.
Dann entsteht das Netz $\widehat{BN} = (\widehat{N}; \widehat{x}, \widehat{g})$ aus BN durch Fusion der Stellen s_1 und s_2, indem

- beide Stellen s_1 und s_2 durch eine einzige Stelle $\widehat{s}$ ersetzt werden,
- die beiden Kanten(t_{pre}, s_1) und (t_{pre}, s_2)durch die neue Kante

  $$(t_{pre}, \widehat{s}),$$

  sowie die beiden Kanten (s_1, t_{post}) und (s_2, t_{post}) durch die neue Kante

  $$(\widehat{s}, t_{post})$$

  ersetzt werden,
- die neuen Kanten beschriftet werden als

  $$x(t_{pre}, \widehat{s}) := \widehat{u}_{pre}$$

  und

  $$\widehat{x}(\widehat{s}, t_{post}) := \widehat{u}_{post},$$
- die Transition t_{pre} die neue Guard-Funktion

  $$\widehat{g}_{t_{pre}}(z; \widehat{u}_{pre}) :=$$

  $$g_{t_{pre}}(z; \widehat{u}_{pre}, \widehat{u}_{pre})$$

  und die Transition t_{post} die neue Guard-Funktion erhält

  $$\widehat{g}_{t_{post}}(z; \widehat{u}_{post}) :=$$

  $$g_{t_{post}}(z; \widehat{v}_{post}, \widehat{v}_{post}),$$
- und alle übrigen Netzelemente von BN unverändert nach $\widehat{BN}$ übernommen werden.

□

Bemerkung 4.4 [Stellenfusion]

1. Eine Guard-Funktion $g(z; u, v)$ hat genau dann die Eigenschaft, daß in jedem Zuweisungselement die Variable u denselben Wahrheitswert hat wie die Variable v, wenn im Ring $B(n)$ folgende algebraische Bedingung gilt:

   $$u + v \in < g + 1 > .$$

   Beweis
   Die Bedingung

   $$g + 1 = 0 \Rightarrow u = v$$

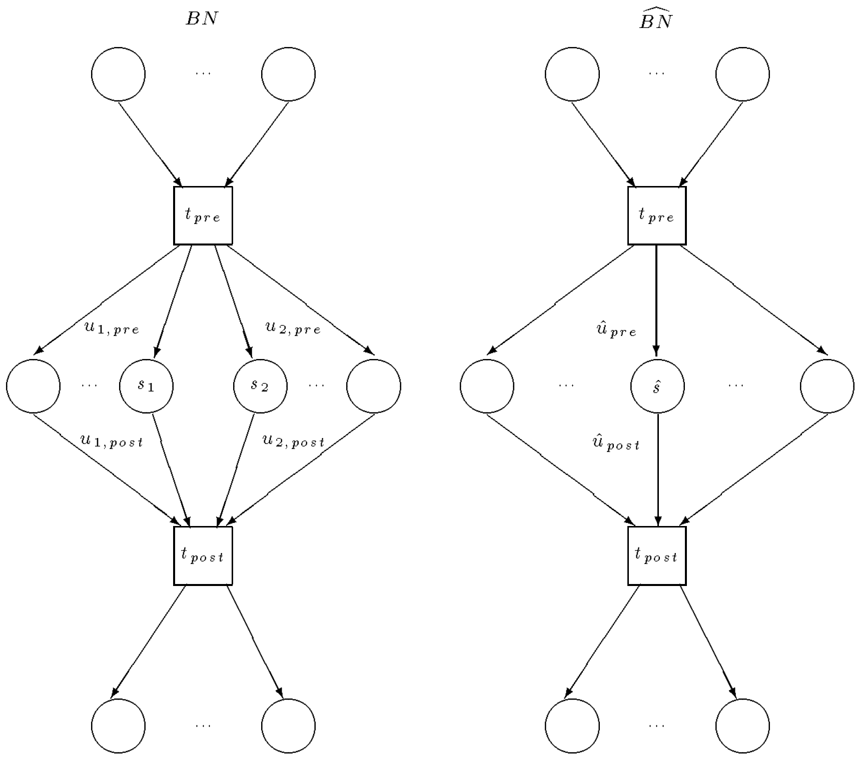


Abbildung 4.1: Fusion von Stellen

bedeutet — wegen $u = v \Leftrightarrow u + v = 0$ — als Aussage über algebraische Varietäten:

$$V(g+1) \subseteq V(u+v),$$

oder als Aussage über Ideale:

$$rad(< u+v >) \subseteq rad(< g+1 >).$$

Da alle Ideale reduziert sind, folgt

$$< u+v > \subseteq < g+1 > .$$

In der umgekehrten Richtung folgt aus

$$< u+v > \subseteq < g+1 >$$

sofort

$$V(g+1) \subseteq V(u+v),$$

so daß jedes Zuweisungselement von g die Bedingung $u + v = 0$ erfüllt. $\diamond$

2. Z. B. erfüllt die Guard-Funktion des *and*-Konnektors

$$g(u,v,z) = 1 + u + v + z + uv + vz + uz$$

die Bedingung

$$u + v \in < g+1 >,$$

weil

$$u + v = a * (g+1)$$

mit

$$a(u,v,z) = 1 + z * (1 + u + v + uv) + uv.$$

$\diamond$

Definition 4.5 [Fusion von Transitionen] Es seien $BN = (N; x, g)$ ein Boolesches Netz, sowie t_1 und t_2 zwei Transitionen mit genau einer gemeinsamen Umgebungsstelle s, d.h.

$$\{s\} = t_1^\bullet \cap {}^\bullet t_2$$

und

$${}^\bullet t_1 \cap t_2^\bullet = \emptyset.$$

Die Variablen an den Kanten von t_1 bzw. t_2 seien

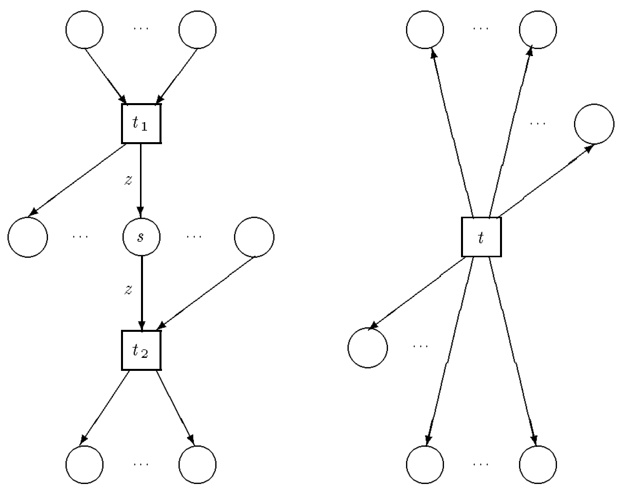


Abbildung 4.2: Fusion von Transitionen

$$(x, z) = (x_1, \dots, x_{n_1}, z) \text{ und } (y, z) = (y_1, \dots, y_{n_2}, z),$$

wobei die ausgezeichnete Variable z die beiden Kanten der gemeinsamen Stelle s beschriftet. Die zugehörigen Guard-Funktionen seien

$$g_1(x; z) \text{ aus } \mathbb{F}_2[x; z] \text{ bzw. } g_2(y; z) \text{ aus } \mathbb{F}_2[y; z].$$

Dann erhält die Boolesche Transition t, die durch Fusion der Transitionen t_1 und t_2 entsteht, die Guard-Formel

$$g(x, y) := \sigma(g_{12}(x, y; 0), g_{12}(x, y; 1)) \text{ aus } \mathbb{F}_2[x, y]$$

mit $g_{12}(x, y; z) := g_1(x; z) * g_2(y; z)$. □

Lemma 4.6 Die Boolesche Transition t, die gemäß Definition 4.5 aus der Fusion der beiden Transitionen t_1 und t_2 entsteht, hat dieselben Zuweisungselemente wie die beiden Transitionen t_1 und t_2 gemeinsam, d.h. $(t, (b_1, b_2))$ ist genau dann ein Zuweisungselement, wenn eine gemeinsame Zuweisung b_s der Variablen an den beiden Kanten(t_1, s) und (s, t_2) existiert, so daß $(t_1, (b_1, b_s))$ und $(t_2, (b_2, b_s))$ Zuweisungselemente sind. □

Beweis
Die Boolesche Transition t hat die Guard-Funktion

$$g_t(x, y) = \sigma(g_{12}(x, y; 0), g_{12}(x, y; 1)),$$

d.h.

$$g_t(x, y) = g_{12}(x, y; 0) + g_{12}(x, y; 1) + g_{12}(x, y; 0) * g_{12}(x, y; 1).$$

Es gilt

$$g_t(x, y) = 1 \Leftrightarrow g_{12}(x, y; 0) = 1 \text{ oder } g_{12}(x, y; 1) = 1$$

Nach Definition von g_{12} gilt

$$g_{12}(x, y; 0) = 1 \Leftrightarrow g_1(x; 0) = 1 \text{ und } g_2(y; 0) = 1$$

und analog für $g_{12}(x, y; 1)$. ◇

Bemerkung 4.7 Definition 4.5 und Lemma 4.6 verallgemeinern sich ohne weiteres auf die Fusion von zwei Transitionen t_1 und t_2 mit genau n gemeinsamen Umgebungsstellen, $n \geq 1$:

$$\{s_1, \ldots, s_n\} = t_1^\bullet \cap {}^\bullet t_2$$

und

$${}^\bullet t_1 \cap t_2^\bullet = \emptyset.$$

Beschriftet $z_1, \ldots, z_n$ die Kanten der gemeinsamen Stellen, so erhält die Transition, die durch Fusion von t_1 und t_2 entsteht, die Guard-Formel:

$$g(x, y) := \sigma(g_1(x; a_1) * g_2(y; a_1), \ldots, g_1(x; a_{2^n}) * g_2(y; a_{2^n}))$$

mit $\{a_1, \ldots, a_{2^n}\} = Boole^n$. ◇

Bemerkung 4.8 Die beiden Fusionsregeln von Definition 4.3 und Definition 4.5 lassen sich auch als Algorithmus formulieren. In Verbindung mit einem mathematischen Programm zur Berechnung von Nullstellen von Polynomen über $\mathbb{F}_2$ und mit den üblichen Traversierungsalgorithmen auf gerichteten Graphen läßt sich hieraus ein Computerprogramm entwickeln, das die Anzahl der Netzelemente eines Booleschen Netzes durch Fusion von Stellen und Transitionen reduziert. ◇

Bemerkung 4.9 [Aktivierungstreue]

1. Nach Bemerkung 3.4 sind alle Booleschen Transitionen, die aus der Übersetzung einer EPK resultieren, aktivierungstreu. Diese Eigenschaft bleibt auch erhalten für Transitionen t, die durch anschließende Fusion gemäß Definition 4.5 entstehen. Bezeichnet g die Guard-Funktion einer aktivierungstreuen Transition und $x = (x_1, \ldots, x_n)$ ihre Eingangs- bzw. $z = (z_1, \ldots, z_m)$ ihre Ausgangsvariablen, so gilt

   $$g(x, z) = 1 + \sigma(x) + \tau(z) + m(x, z)$$

   mit einem Polynom

   $$m(x, z) \in < x >< z > \subseteq B(n + m),$$

   und $\sigma \in \mathbb{F}_2[x], \tau \in \mathbb{F}_2[z]$ als Summe der jeweiligen elementar-symmetrischen Funktionen.

   Beweis
   Nach Voraussetzung gilt

   $$\{z : g(0, z) = 1\} = \{0\} \text{ und } \{x : g(x, 0) = 1\} = \{0\}.$$

   Nach Bemerkung 4.1, Teil 3, haben die eingeschränkten Polynome die Taylorentwicklungen

   $$g(0, z) - 1 = \tau(z) \text{ und } g(x, 0) - 1 = \sigma(x),$$

   weil die Elemente von $B(m)$ bzw. von $B(n)$ bereits durch ihre Nullstellen festgelegt sind. ◇

2. Offensichtlich sind die Booleschen Transitionen, die aus der Übersetzung einer EPK stammen, durch dieses Schaltverhalten bereits charakterisiert: Die Boolesche Transition t habe zwei gleichgerichtete Kanten mit den Beschriftungen x_1, x_2 und eine weitere, andersorientierte Kante mit der Beschriftung z. Dann stammt t genau dann von dem logischen Konnektor einer EPK, wenn die zugehörige Guard-Funktion $g \in B(3)$ die Gestalt hat:

$$g(x, z) = 1 + \sigma(x) + z + m(x) * z, x = (x_1, x_2)$$

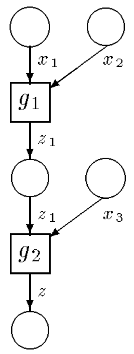
mit einem symmetrischen Polynom

$$m \neq \sigma \in \mathbb{F}_2[x].$$

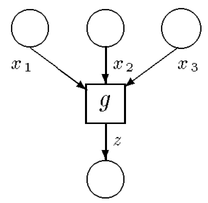
$\diamond$

Beispiel 4.10 Die folgenden Beispiele veranschaulichen die Aussage von Lemma 4.6 über Transitionsfusion. Sie zeigen, daß das ursprüngliche Netz dieselben Zuweisungselemente hat wie das durch Transitionsfusion entstandene.

1. Aus dem Booleschen Netz



entsteht durch Fusion der beiden Transitionen gemäß Definition 4.5 das Boolesche Netz



mit der Guard-Formel:

$$g = op_1 * (c + d) + (1 + \sigma) * (1 + \tau + d) * (1 + [\sigma + op_1] * [\tau + c])$$

wobei die Funktionen

$$c, d \in \mathbb{F}_2[x_3, z]$$

definiert sind als die Koeffizienten des Polynoms

$$op_2(z_1, x_3) * z = c * z_1 + d \in \mathbb{F}_2[x_3, z][z_1],$$

und

$$\sigma = \sigma_1 + \sigma_2 \in \mathbb{F}_2[x_1, x_2]$$

bzw.

$$\tau = \tau_1 + \tau_2 \in \mathbb{F}_2[x_3, z]$$

die Summe der elementar-symmetrischen Funktionen σ_j, bzw. $\tau_j, j = 1, 2$, bezeichnet. Man beachte, daß sich obige Guard-Formel in den Fällen „ $op_2 = \ and$ “ vereinfacht zu

$$g = 1 + (1 + \ op_1 * \ op_2) * (\sigma + \tau + \sigma * \tau).$$

Im folgenden behandeln wir exemplarisch die Fusion von Booleschen Transitionen, die aus der Übersetzung einer EPK resultieren. Wir werden für einige der 9 möglichen Kombinationen

$$(\ op_1, \ op_2) \in \{\ or\ ,\ and\ ,\ xor\ \} \times \{\ or\ ,\ and\ ,\ xor\ \}$$

jeweils die Guard-Formel der resultierenden Transition und ihre Zuweisungselemente berechnen. Nach Bemerkung 4.9, Teil 1 hat die Guard-Funktion in diesen Fällen die Gestalt

$$g(x, z) = 1 + \sigma(x) + z + m(x) * z,$$

$\sigma(x) \in \mathbb{F}_2[x_1, x_2, x_3],\ m(x) \in < x >$

- Fall $(\ op_1, \ op_2) = (\ and\ ,\ and\ )$:

  $$g = 1 + \sigma + z + (\sigma_1 + \sigma_2) * z$$

  Der zugehörige Boolesche Ausdruck heißt

  $$g = [(x_1\ \ or\ x_2\ \ or\ x_3\ \ or\ z) \Rightarrow x_1\ \ and\ x_2\ \ and\ x_3\ \ and\ z)]$$

  mit den Zuweisungselementen

x_1	x_2	x_3	z
1	1	1	1
0	0	0	0

- Fall $(\ op_1, \ op_2) = (\ or\ ,\ and\ )$:

  $$g = 1 + \sigma + z + (\sigma_1 + x_1 * x_2) * z$$

  Der zugehörige Boolesche Ausdruck heißt

  $$(x_1\ \ or\ x_2\ \ or\ x_3\ \ or\ z) \Rightarrow [(x_1\ \ or\ x_2)\ \ and\ x_3\ \ and\ z)].$$

x_1	x_2	x_3	z
1	0	1	1
0	1	1	1
1	1	1	1
0	0	0	0

- Fall $(\ op_1, \ op_2) = (\ and\ ,\ or\ )$:

  $$g = 1 + \sigma + z + (x_1 + x_2) * z$$

x_1	x_2	x_3	z
1	1	1	1
1	1	0	1
0	0	1	1
0	0	0	0

Man beachte, daß es sich hierbei nicht um den Booleschen Ausdruck

$$(x_1 \ or \ x_2 \ or \ x_3 \ or \ z) \Rightarrow [[(x_1 \ and \ x_2) \ or \ x_3] \ and \ z]$$

handelt, sondern um die Formel

$$(x_1 \ or \ x_2 \ or \ x_3 \ or \ z) \Rightarrow [(x_1 \Leftrightarrow x_2) \ and \ (x_1 \ or \ x_3) \ and \ z].$$

- Fall $(op_1, op_2) = (xor, or)$:

$$g = 1 + \sigma + z + x_1 * x_2 * z$$

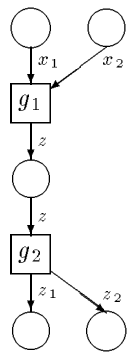
x_1	x_2	x_3	z
1	0	0	1
0	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
0	0	0	0

Man beachte, daß es sich hierbei nicht um den Booleschen Ausdruck

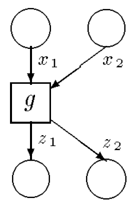
$$(x_1 \ or \ x_2 \ or \ x_3 \ or \ z) \Rightarrow [[(x_1 \ xor \ x_2) \ or \ x_3] \ and \ z]$$

handelt.

2. Aus dem Booleschen Netz



entsteht das Boolesche Netz



mit der Guard-Formel

$$g = 1 + (\sigma + \tau + \sigma * \tau) * (1 + op_1 * op_2).$$

Diese Guard-Formel entspricht dem Booleschen Ausdruck

$$x_1 \ or \ x_2 \ or \ z_1 \ or \ z_2 \Rightarrow [(x_1 \ op_1 x_2) \ and \ (z_1 \ op_2 z_2)].$$

Wieder betrachten wir exemplarisch einige Fälle. Die Guard-Funktion hat in diesen Fällen nach Bemerkung 4.9, Teil 1 die Gestalt

$$g(x, z) = 1 + \sigma(x) + \tau(z) + m(x, z),$$

mit $\sigma(x) \in \mathbb{F}_2[x_1, x_2]$, $\tau(z) \in \mathbb{F}_2[z_1, z_2]$ und $m(x, z) \in <x><z>$.

- Fall $(op_1, op_2) = (and, and)$:

$$g = 1 + \sigma + \tau + \sigma_1 * \tau + \sigma_2 * \tau_1$$

x_1	x_2	z_1	z_2
1	1	1	1
0	0	0	0

- Fall $(op_1, op_2) = (or, and)$:

$$g = 1 + \sigma + \tau + \sigma * \tau_1$$

x_1	x_2	z_1	z_2
1	1	1	1
1	0	1	1
0	1	1	1
0	0	0	0

- Fall $(op_1, op_2) = (xor, and)$:

$$g = 1 + \sigma + \tau + \sigma * \tau_1 + \sigma_2 * \tau_2$$

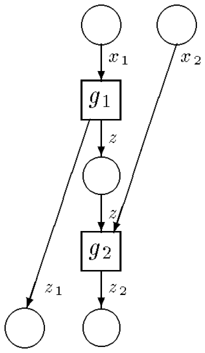
x_1	x_2	z_1	z_2
1	0	1	1
0	1	1	1
0	0	0	0

- Fall $(op_1, op_2) = (xor, or)$:

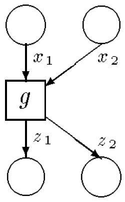
$$g = 1 + \sigma + \tau + \sigma_2 * \tau$$

x_1	x_2	z_1	z_2
1	0	1	0
1	0	0	1
1	0	1	1
0	1	1	0
0	1	0	1
0	1	1	1
0	0	0	0

3. Aus dem Booleschen Netz



entsteht das Boolesche Netz



Wieder betrachten wir exemplarisch einige Fälle. Die Guard-Funktion hat in diesen Fällen nach Bemerkung 4.9, Teil 1 die Gestalt

$$g(x,z) = 1 + \sigma(x) + \tau(z) + m(x,z),$$

mit $\sigma(x) \in \mathbb{F}_2[x_1, x_2], \tau(z) \in \mathbb{F}_2[z_1, z_2]$ und $m(x,z) \in < x >< z >$.

- Fall (op_1, op_2) = (and , or):

$$g = 1 + \sigma + \tau + \sigma * z_1 + x_1 * z_2$$

x_1	x_2	z_1	z_2
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	1

- Fall (op_1, op_2) = (xor , and):

$$g = 1 + \sigma + \tau + \sigma * z_2 + x_2 * z_1 + x_1 * x_2 * z_1 * z_2$$

x_1	x_2	z_1	z_2
0	0	0	0
1	0	1	0
1	1	0	1

$\diamond$

Kapitel 5

Schleifenbäume

In diesem Kapitel führen wir eine spezielle Klasse von Free-Choice Netzen ein: „Schleifenbäume“ treten bei der Übersetzung einer EPK in ein Petri-Netz auf, wenn die EPK innere Schleifen besitzt.

Die Erfinder der EPK haben den Grundsatz aufgestellt, Verzweigungen nur an den logischen Konnektoren einer EPK zuzulassen. Entsprechend modellieren sie Anfang und Ende einer inneren Schleife als schließenden bzw. öffnenden Konnektor. Dadurch treten diese Konnektoren jedoch in zwei ganz verschiedenen Bedeutungen auf und stehen im Modell für unterschiedliche Vorgänge der Realität. Das hat zur Folge, daß ein grundlegender Unterschied der Realität an dieser Stelle der EPK nicht mehr erkennbar ist:

- An einem xor-Konnektor, der eine Branch-Alternative öffnet, verzweigt sich in der Realität der Informationsfluß: Der eine Teilprozeß wird aktiviert, der andere Teilprozeß wird explizit deaktiviert. Diese Aktivierungsinformationen fließen über beide Alternativen weiter und werden an anderer Stelle synchronisiert.

- An einem xor-Konnektor, der eine Schleife öffnet, fließt der Informationsfluß in der Realität entweder in den Schleifenrumpf hinein oder an der Schleife vorbei — ohne sich jedoch zu verzweigen. In jedem Fall wird nur eine der beiden Alternativen aktiviert, die andere bleibt unberücksichtigt. Insbesondere wird die andere Alternative nicht deaktiviert und daher auch nicht an späterer Stelle synchronisiert.

Ein entsprechender Unterschied gilt für schließende Konnektoren.

Daher haben wir bei unseren Übersetzungsregeln in Definition 3.2 Wert auf die Unterscheidung beider Fälle gelegt: Die Konnektoren einer Branch-Alternative werden in eine verzweigte Transition übersetzt, die Konnektoren einer Schleifen-Alternative dagegen in eine verzweigte Stelle. Das Petri-Netz respektiert also die unterschiedlichen Situationen der Realität auch im Modell:

- Im Falle einer Branch-Alternative fließt über beide Ausgänge der verzweigten Transition je eine Marke.

- Im Falle einer Schleifenverzweigung gibt es nur eine einzige Marke, und diese fließt nur über einen der beiden Ausgänge der verzweigten Stelle.

Aus diesem Grund verwenden wir zur Modellierung von EPKs mit inneren Schleifen Boolesche Netze, bei denen sowohl die Transitionen als auch die Stellen verzweigt

sein können. Allerdings sind nicht beliebige Verzweigungen zugelassen, sondern verzweigende Stellen dürfen nur als „Angelpunkt“ einer Schleife auftreten. Darüberhinaus sind keine Ein- oder Aussprünge aus dem Schleifenrumpf zugelassen, und mehrfache Schleifen müssen in wohldefinierter Weise geschachtelt sein.
Im folgenden präzisieren wir diesen Grundgedanken der Schleifenmodellierung im Begriff des „Schleifenbaumes“ und leiten einige Eigenschaften dieser Netze ab. Insbesondere werden wir ein einfaches Kriterium beweisen, mit dem bestimmte Markierungen dieser Netze auf Sicherheit und Lebendigkeit geprüft werden können.

Definition 5.1 [Netzadjunktion]

1. Unter einem *stellenpunktierten Netz* $(N; s)$ verstehen wir ein Paar bestehend aus
   - einem Netz $N = (S, T; A)$ und
   - einer ausgezeichneten Stelle $s \in S$, dem *Basispunkt* von N.

2. Es seien zwei disjunkte, stellenpunktierte Netze

$$(N_i, s_i), \; i = 0, 1,$$

   und eine unverzweigte Stelle $l \neq s_0$ von N_0 vorgegeben. Heftet man das Netz N_1 an das Netz N_0 an, indem man den Basispunkt s_1 von N_1 mit der Stelle l von N_0 identifiziert, so entsteht
   - ein stellenpunktiertes Netz N mit Basispunkt $s := s_0$
   - zusammen mit zwei kanonischen Morphismen von stellenpunktierten Netzen

$$j_i : (N_i; s_i) \longrightarrow (N; s), \; i = 0, 1,$$

$$j_0(x_0) := x_0$$

   und

$$j_1(x_1) := \begin{cases} x_1, & \text{falls } x_1 \neq s_1 \\ l, & \text{falls } x_1 = s_1 \end{cases}$$

   Man nennt

$$((N; s), j_0, j_1)$$

   die *Adjunktion* des Netzes $(N_1; s_1)$ an das Netz $(N_0; s_0)$ an der Stelle l, dem *Angelpunkt* der Adjunktion, und schreibt

$$(N, s) := (N_0, s_0) \coprod_l (N_1, s_1).$$

□

Bemerkung 5.2 [Zur Netzadjunktion]

1. Die kanonischen Morphismen $j_i, \; i = 0, 1,$ einer Netzadjunktion sind offensichtlich Einbettungen auf offene Teilnetze. Mit Hilfe dieser Morphismen kann man jedes der beiden Netze N_i mit seinem Bild $j_i(N_i)$ in N identifizieren und als offenes Teilnetz von N auffassen.

2. Wir werden im folgenden häufig auf die explizite Angabe des Basispunktes verzichten.

3. Die Adjunktion $N := N_0 \coprod_l N_1$ zweier Netze

$$(N_i; s_i),\ i = 0, 1,$$

ist genau dann ein Free-Choice Netz, wenn beide Netze N_i Free-Choice Netze sind und der Angelpunkt l im Netz N_0 die Bedingung erfüllt

$$^\bullet(l^\bullet) = \{l\}.$$

Diese Bedingung bedeutet, daß keine Nachtransition von l im Netz N_0 rückwärts verzweigt.

$\diamond$

Definition 5.3 [Elementares Schleifennetz] Ein *elementares Schleifennetz* $ESN = (TN; s)$ ist ein stellenpunktiertes Netz mit einem stark-zusammenhängenden T-Netz TN, so daß $TN \setminus s$ zyklenfrei ist. $\square$

Definition 5.4 [Schleifenbaum] Ein zusammenhängendes, stellenpunktiertes Netz $SN = (N; s)$ heißt *Schleifenbaum*, wenn

- paarweise verschiedene Stellen von N

$$l_i \neq s, ^\bullet(l_i^\bullet) = \{l_i\},\ i = 1, \ldots, n$$

- und elementare Schleifennetze

$$ESN_i = (N_i; s_i),\ i = 0, \ldots, n$$

existieren, so daß mit der induktiven Definition

$$SN_0 := ESN_0, SN_i := SN_{i-1} \coprod_{l_i} ESN_i,\ i = 1, \ldots, n,$$

gilt: $SN = SN_n$.

Wir nennen die Teilnetze $ESN_i,\ i = 0, \ldots, n$ die *Schleifenkomponenten* von SN, die ausgezeichnete Komponente ESN_0 mit Basispunkt $s_0 = s$ heißt *Basiskomponente*. Die ausgezeichneten Stellen $l_i,\ i = 1, \ldots, n$, heißen *Angelpunkte* von SN. $\square$

Definition 5.5 [Graph der Schleifenkomponenten] Für einen Schleifenbaum SN definieren wir den *gerichteten Graphen der Schleifenkomponenten*

$$BS(SN) = (V, A)$$

durch folgende Festlegung:

- Jede Schleifenkomponente ESN von SN definiert einen Knoten

$$v(ESN) \in V$$

von $BS(SN)$.

- Jeder Angelpunkt l der Adjunktion einer Schleifenkomponente ESN an ein Teilnetz $SN_i,\ i = 0, \ldots, n-1$, definiert eine gerichtete Kante

  $$a(l) \in A$$

  von dem Knoten $v(ESN')$ mit $l \in ESN'$ zum Knoten $v(ESN)$ in $BS(SN)$.

$\square$

Bemerkung 5.6 [Zu Schleifenbäumen]

1. Anschaulich gesprochen entsteht ein Schleifenbaum, indem man an ein elementares Schleifennetz, die Basiskomponente, sukzessive elementare Schleifennetze $ESN = (TN; s)$ als weitere Schleifen anheftet. Dabei erzeugt man bei jeder Adjunktion eine kopfgesteuerte Schleife mit Schleifenrumpf $TN \setminus s$.

2. Bekanntlich kann man jede Schleife in eine kopfgesteuerte Schleife überführen, so daß die Betrachtung ausschließlich kopfgesteuerter Schleifen im Rahmen der Schleifennetze keine Beschränkung der Allgemeinheit bedeutet.

3. Die Bedingung ${}^\bullet\{l_i^\bullet\} = \{l_i\}$ an den Verheftungsstellen stellt sicher, daß nach der Verheftung der elementaren Schleifennetze wieder ein Free-Choice Netz entsteht: Jedes Schleifennetz SN ist ein Free-Choice Netz. Diese Bedingung kann stets durch eine lokale Abänderung erreicht werden, indem man in das Netz SN_{i-1} eine unverzweigte Transition $t = l_i^\bullet$ einfügt.

4. Offensichtlich ist der Graph $BS(SN)$ der Schleifenkomponenten eines Schleifenbaumes SN ein Baum. Das folgt aus dem Zusammenhang von SN und der Forderung, daß der Basispunkt von SN kein Angelpunkt ist. Die Wurzel $v(ESN_0)$ des Baumes $BS(SN)$ stammt von der Basiskomponente ESN_0 von SN. Die Schleifenkomponenten ESN zu den Blättern $v(ESN)$ von $BS(SN)$ enthalten keine Angelpunkte außer dem jeweiligen Basispunkt.

5. Die Adjunktion eines Schleifenbaumes an einen Schleifenbaum an einem weiteren Angelpunkt l mit ${}^\bullet\{l^\bullet\} = \{l\}$ ist wieder ein Schleifenbaum.

6. Die hier vorgestellte Form der Schleifenmodellierung durch Stellenverzweigung erlaubt es noch nicht, Abbruchbedingungen für den Schleifendurchlauf zu modellieren. Semantisch mächtiger wäre es, die gesamte Schleife — einschließlich Schleifenrumpf und Entscheidungsfunktion — als Transitionsverfeinerung zu modellieren und eine weitere Farbe für die Abbruchbedingung einzuführen. Die Ausführung dieses Gedankens soll ggf. an anderer Stelle erfolgen.

$\diamond$

Grundlegend für die Analyse eines Free-Choice Netzsystems sind gewisse strukturelle Eigenschaften des unterliegenden Netzes. Wir rekapitulieren diese Begriffe im folgenden und orientieren uns dabei wieder an [DE95].

Definition 5.7 Es sei $N = (S, T; A)$ ein Netz.

1. Ein *Siphon* von N ist eine Teilmenge von Stellen

   $$R \subseteq S \text{ mit } {}^\bullet R \subseteq R^\bullet.$$

   Ein nichtleerer Siphon R heißt *minimaler Siphon*, wenn er keinen nichtleeren Siphon als echte Teilmenge enthält.

2. Eine *Falle* von N ist eine Teilmenge von Stellen
$$R \subseteq S \text{ mit } R^\bullet \subseteq {}^\bullet R.$$
Eine nicht leere Falle R heißt *minimale Falle*, wenn sie keine nicht leere Falle als echte Teilmenge enthält.

3. Eine *S-Komponente* von N ist ein Teilnetz
$$SK = (S_1, T_1; A_1)$$
mit folgender Eigenschaft:
SK ist ein stark-zusammenhängendes S-Netz und
$${}^\bullet S_1 \cup S_1^\bullet \subseteq T_1.$$

4. Eine *T-Komponente* von N ist ein Teilnetz
$$TK = (S_1, T_1; A_1)$$
mit folgender Eigenschaft:
TK ist ein stark-zusammenhängendes T-Netz
$${}^\bullet T_1 \cup T_1^\bullet \subseteq S_1.$$

5. Ein *Cluster* von N ist eine Teilmenge cl von Knoten von N, die minimal ist bezüglich folgender Eigenschaft:
   - cl enthält eine Stelle.
   - cl enthält mit jeder Stelle s auch $s^\bullet$.
   - cl enthält mit jeder Transition t auch ${}^\bullet t$.

□

Bemerkung 5.8 [Eigenschaften von Schleifenbäumen]

1. Die T-Komponenten eines Schleifenbaumes SN sind genau seine Schleifenkomponenten.

   Zum Beweis beachte man, daß eine T-Komponente TK von SN eine Schleifenkomponente ESN an einem Angelpunkt l nicht verlassen kann: Denn TK ist stark-zusammenhängend, ein Weg in TK von
$$l^\bullet \cap SN \setminus ESN$$
   nach
$${}^\bullet l \cap ESN$$
   und sein Rückweg in TK enthalten beide den Angelpunkt l, aber dieser kann in TK nicht verzweigen. Daher gilt
$$TK \subseteq ESN.$$
   Außerdem stimmt eine T-Komponente eines zusammenhängenden T-Netzes mit dem gesamten Netz überein, so daß
$$TK = ESN.$$

2. Die Zyklen eines Schleifenbaumes SN sind genau die Zyklen seiner Schleifenkomponenten.

   Wir ordnen jeder Schleifenkomponente ESN von SN als Invariante die Tiefe des zugehörigen Knotens $v(ESN)$ im Baum der Schleifenkomponenten $BS(SN)$ zu. Diese Invariante nennen wir die Tiefe der Schleifenkomponente ESN. Analog sprechen wir von der Tiefe eines Zyklus von SN. Offensichtlich haben in einem Schleifenbaum genau die Zyklen der Basiskomponente die Tiefe 0.

3. Inwieweit ist ein Schleifenbaum durch den Baum seiner Schleifenkomponenten bestimmt? Bezüglich eines Homotopiebegriffes, der zwei Wege miteinander identifiziert, wenn sie durch Transitionsverzweigung auseinander hervorgehen (T-Homotopie), bestimmt der Baum der Schleifenkomponenten gerade die Homotopieklasse des Schleifenbaumes.

4. Für eine Teilmenge R von Stellen eines Netzes N bezeichnen wir mit $N(R, {}^\bullet R)$ das von R und der Transitionenmenge ${}^\bullet R$ erzeugte Teilnetz von N.

$\diamond$

Satz 5.9 [Charakterisierung eines minimalen Siphons] Es sei FCN ein zusammenhängendes Free-Choice Netz. Dann ist eine Teilmenge R von Stellen genau dann ein minimaler Siphon von FCN, wenn gilt:

- Jedes Cluster von FCN enthält höchstens eine Stelle von R.
- Das Teilnetz $N(R, {}^\bullet R)$ ist stark-zusammenhängend.

$\square$

Beweis
Siehe [DE95], Chapter 4.5. $\diamond$

Lemma 5.10 Es seien (FCN_i, s_i), $i = 0, 1$, zwei stellenpunktierte Free-Choice Netze und

$$FCN := FCN_0 \coprod_l FCN_1$$

die Adjunktion von FCN_1 an FCN_0 an einer Stelle l mit ${}^\bullet(l^\bullet) = \{l\}$.

1. Für eine Teilmenge R von Stellen in FCN setzen wir $R_i = R \cap FCN_i$, $i = 0, 1$. Dann gilt:

   (a) Wenn R ein minimaler Siphon von FCN ist, so ist jede Menge

   $$R_i, \ i = 0, 1,$$

   entweder leer oder ein minimaler Siphon von FCN_i.

   (b) Wenn jede Menge

   $$R_i, \ i = 0, 1,$$

   ein minimaler Siphon von FCN_i ist und wenn

   $$R_0 \cap R_1 = l,$$

   so ist R ein minimaler Siphon von FCN.

2. Ein Teilnetz S von FCN ist genau dann eine S-Komponente von FCN, wenn die beiden Netze

$$S_i := S \cap FCN_i, \; i = 0, 1,$$

jeweils S-Komponenten von FCN_i sind und

$$S_0 \cap S_1 = \{l\}.$$

□

Beweis
Wir verwenden die Charakterisierung eines minimalen Siphons nach Satz 5.9.

1a) Für $i = 0, 1$ gilt:

1. Die Menge R_i erfüllt in FCN_i ebenfalls die Clusterbedingung, weil die entsprechenden Cluster in FCN_i nicht mehr Stellen enthalten als in SN.
2. Das von R_i und ${}^\bullet R_i$ in FCN_i erzeugte Teilnetz $N(R_i, {}^\bullet R_i)$ ist stark-zusammenhängend: Denn seine Netzelemente lassen sich nach Voraussetzung in N durch einen einfachen Weg verbinden. Da der Weg einfach ist, verläuft er bereits vollständig in FCN_i.

1b) Die Menge der Stellen und Transitionen des Teilnetzes $N(R, {}^\bullet R)$ ist die Vereinigung der Stellen und Transitionen beider Netze $N(R_i, {}^\bullet R_i), \; i = 0, 1$. Wegen der Bedingung

$$R_0 \cap R_1 = \{l\}$$

kommen in $N(R, {}^\bullet R)$ keine neuen Kanten hinzu. Daher gilt auch für die erzeugten Netze:

$$N(R, {}^\bullet R) = N(R_0, {}^\bullet R_0) \cup N(R_1, {}^\bullet R_1).$$

Beide Netze haben genau den Angelpunkt l gemeinsam. Daher folgt der starke Zusammenhang von $N(R, {}^\bullet R)$ aus dem starken Zusammenhang von $N(R_i, {}^\bullet R_i)$, $i = 0, 1$.

Der Angelpunkt l liegt in einem Cluster $cl(l)$, das die Vereinigung zweier Cluster $cl_i(l)$ in FCN_i ist, die jeweils l als einzige Stelle aus R_i enthalten. Also enthält $cl(l)$ auch nur eine einzige Stelle aus R, nämlich den Angelpunkt. Alle anderen Cluster von FCN liegen bereits als Cluster in einem der beiden Teilnetze $FCN_i, \; i = 0, 1$.

2) 1. Das Netz S ist genau dann stark-zusammenhängend, wenn beide Netze $S_i, \; i = 0, 1$, stark zusammenhängen und

$$S_0 \cap S_1 = \{l\}.$$

2. Die Transitionenmenge von S ist die Vereinigung der Transitionenmengen von $S_i, \; i = 0, 1$, denn im Durchschnitt beider Netze liegt keine Transition. Also sind die Transitionen von S genau dann unverzweigt, wenn die Transitionen von S_0 und von S_1 unverzweigt sind.

3. Für eine Stellenmenge R von S setzen wir $R_i := R \cap S_i,\ i = 0, 1$. Um die Äquivalenz der Aussage

$$^\bullet R \cup R^\bullet \subseteq S,$$

mit den beiden Aussagen

$$(^\bullet R_i \cup R_i^\bullet) \subseteq S_i$$

$i = 0, 1$, zu zeigen, genügt es, den Angelpunkt l zu betrachten.
Hier gilt

$$^\bullet l \cup l^\bullet \subseteq S \Leftrightarrow [(^\bullet l \cap S_i) \cup (l^\bullet \cap S_i)] \subseteq S_i,$$

$i = 0, 1$, da

$$S = S_0 \cup S_1$$

und

$$S_0 \cap S_1 = \{l\}.$$

$\diamond$

Lemma 5.11 In einem T-Netz gilt:

1. Jeder Zyklus mit Stellenmenge R ist das Teilnetz $N(R, {}^\bullet R)$.
2. Für ein Teilnetz γ sind äquivalent:
   (a) γ ist ein Zyklus.
   (b) γ ist eine S-Komponente.
   (c) $\gamma = N(R, {}^\bullet R)$ für einen minimalen Siphon R.

$\square$

Beweis

1) In einem T-Netz TN sind alle Stellen s unverzweigt, d.h.

$$|\ {}^\bullet s\ | = |\ s^\bullet\ | = 1.$$

Der Zyklus γ enthält daher mit R auch die Transitionenmengen ${}^\bullet R$ und das Teilnetz

$$N(R, {}^\bullet R) \subseteq \gamma,$$

Da γ ein Zyklus ist und keine Stelle $s \in R$ verzweigt, stimmen beide Transitionenmengen ${}^\bullet R$ und $R^\bullet$ überein. Hieraus folgt auch die umgekehrte Inklusion

$$\gamma \subseteq N(R, {}^\bullet R)$$

2a $\Rightarrow$ 2b) Sei γ ein Zyklus mit Stellenmenge R. Dann ist γ stark-zusammenhängend. Nach Teil 1 ist γ auch ein Teilnetz, als Zyklus insbesondere also ein S-Netz. Bezeichnet TN das umgebende T-Netz, so sind alle Stellen von γ auch in TN unverzweigt. Somit enthält γ auch alle Vortransitionen ${}^\bullet R$ und alle Nachtransitionen $R^\bullet$, d.h.

$${}^\bullet R \cup R^\bullet \subseteq \gamma.$$

Also ist γ eine S-Komponente von TN.

2b $\Rightarrow$ 2c) In jedem Netz bilden die Stellen R einer S-Komponente einen minimalen Siphon, vgl. [DE95], Chapter 5. Als S-Komponente enthält γ auch die Vortransitionen ${}^\bullet R$, also

$$N(R, {}^\bullet R) \subseteq \gamma.$$

Eine vorgegebene Transition t von γ liegt wegen des starken Zusammenhanges auf einem Zyklus in γ, es gibt insbesondere eine Stelle s von γ mit $t \in {}^\bullet s$. Nach Definition enthält $N(R, {}^\bullet R)$ alle Stellen von γ, mit der Stelle s also auch die Transition t. Da die Transition t beliebig vorgegeben war, folgt

$$N(R, {}^\bullet R) = \gamma.$$

2c $\Rightarrow$ 2a) Sei R ein minimaler Siphon. Nach Satz 5.9 ist $N(R, {}^\bullet R)$ stark-zusammenhängend. Wegen $R \neq \emptyset$ gibt es eine Stelle $s \in R$, und wegen des starken Zusammenhanges gibt es einen Zyklus γ_s in $N(R, {}^\bullet R)$, welcher s enthält. Annahme: Es gibt ein Netzelement

$$v \in N(R, {}^\bullet R) \setminus \gamma_s.$$

Dann wählen wir einen einfachen Weg α in $N(R, {}^\bullet R)$ von v nach s. Wegen $v \neq s$ existiert ein Netzelement von N, an dem sich beide Wege α und γ_s rückwärts verzweigen. In einem T-Netz ist jedes verzweigende Netzelement eine Transition. Es gibt also eine Transition

$$t \in (\gamma_s \cap \alpha)$$

mit zwei Stellen

$$s_\alpha \neq s_{\gamma_s} \in {}^\bullet t \cap R.$$

Andererseits liegen beide Stellen in dem von t erzeugten Cluster. Wir erhalten einen Widerspruch zu Satz 5.9. Also haben γ und $N(R, {}^\bullet R)$ dieselben Netzelemente. Da γ_s nach Teil 1) bereits ein Teilnetz ist, folgt

$$\gamma_s = N(R, {}^\bullet R).$$

◇

Satz 5.12 [Darstellungssatz eines minimalen Siphons] Vorgegeben sei ein Schleifenbaum SN mit Schleifenkomponenten $ESN_i,\ i = 0, \ldots, n$, und Angelpunkten $l_i,\ i = 1, \ldots, n$.

1. Für einen minimalen Siphon R von SN gilt:

   (a) Diejenigen Kanten $a(l)$ im Baum $BS(SN)$ der Schleifenkomponenten von SN, welche zu den Angelpunkten $l \in R$ gehören, erzeugen einen Teilbaum B_R von $BS(SN)$. Der Teilbaum B_R hat die gleiche Wurzel wie $BS(SN)$.

   (b) Das Netz $N(R, {}^\bullet R)$ ist ebenfalls ein Schleifenbaum. Setzt man

   $$I_R := \{i \in [1, \ldots, n] : l_i \in R\},$$

   so hat $N(R, {}^\bullet R)$ die Angelpunkte $l_i \in R,\ i \in I_R$ und die Zyklen

   $$N_i := N(R_i, {}^\bullet R_i) \subseteq ESN_i,$$

   mit $R_i := R \cap ESN_i, i \in I_R \cup \{0\}$, als Schleifenkomponenten.

2. Zu einem Teilbaum B von $BS(SN)$ mit gleicher Wurzel existiert ein minimaler Siphon R, dessen Schleifenbaum $N(R, {}^\bullet R)$ den vorgegebenen Baum B als Baum seiner Schleifenkomponenten hat, wenn gilt:

   Ist $v = v(ESN)$ ein Blatt von B, so liegt in der Schleifenkomponente ESN ein Zyklus, der außer dem Basispunkt von ESN keinen weiteren Angelpunkt von SN enthält.

□

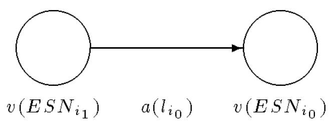
Beweis

1a) Sei B_R das Teilnetz von $BS(SN)$, welches von den Kanten $a = a(l)$ der Angelpunkte $l \in R$ erzeugt wird. Es ist zu zeigen, daß jeder Anfangsknoten einer Kante $a(l)$, der verschieden von der Wurzel ist, zugleich Endknoten einer anderen Kante von B_R ist.

Sei also $i_0 \in I_R$ und $l_{i_0} \in R$ ein Angelpunkt, an dem ein elementares Schleifennetz ESN_{i_0} angeheftet ist. Der Angelpunkt liege in der Schleifenkomponente ESN_{i_1}, d.h.

$$l_{i_0} \in ESN_{i_1} \text{ mit } 0 \leq i_1 \leq n.$$

Im Schleifenbaum $BS(SN)$ gilt dann:



Sei $i_1 \neq 0$. Dann müssen wir zeigen, daß $v(ESN_{i_1})$ Endpunkt einer Kante $a(l_{i_1})$ zu einem weiteren Angelpunkt

$$l_{i_1} \in R$$

ist. Das Netz $N(R, {}^\bullet R)$ enthält mit dem Angelpunkt

$$l_{i_0} \in R$$

auch beide Vortransitionen $t \in {}^\bullet l_{i_0}$. Es sei

$$t_a \in {}^\bullet l_{i_0} \cap ESN_{i_1}$$

die in ESN_{i_1} gelegene Vortransition. Da $N(R, {}^\bullet R)$ stark-zusammenhängt, existiert ein einfacher Weg in $N(R, {}^\bullet R)$ von l_{i_0} nach t_a. Nach Erweiterung um die Kante $(t_{a,l_{i_0}})$ erhält man einen Zyklus γ, der die Schleifenkomponente ESN_{i_1} trifft, also nach Bemerkung 5.8, Teil 2 in ESN_{i_1} verläuft. Nach Definition 5.3 des elementaren Schleifennetzes enthält jeder Zyklus in ENS_{i_1} den Basispunkt s_{i_1}. Also ist

$$l_{i_1} := s_{i_1} \in N(R, {}^\bullet R)$$

der gesuchte Angelpunkt.

1b) Eine Schleifenkomponente ESN mit Basispunkt s trifft $N(R, {}^\bullet R)$ genau dann, wenn $s \in R$. In diesem Fall ist

$$N(R, {}^\bullet R) \cap ESN$$

ein Zyklus.

Denn für jede Schleifenkomponente ESN_i von SN, welche $N(R, {}^\bullet R)$ und damit auch R trifft, ist R_i nach Lemma 5.10 ein minimaler Siphon in ESN_i, also $N(R_i, {}^\bullet R_i)$ nach Lemma 5.11 ein Zyklus. Dieser Zyklus enthält den Basispunkt von ESN_i nach Definition 5.3 eines elementaren Schleifennetzes.

Nach Teil 1a trifft die Basiskomponente das Netz $N(R, {}^\bullet R)$. Die übrigen Schleifenkomponenten treffen $N(R, {}^\bullet R)$ genau dann, wenn der Angelpunkt, an dem sie angeheftet sind, zu R gehört.

2) Es sei ein Teilbaum B von $BS(SN)$ mit der genannten Eigenschaft vorgegeben. Dann wählen wir für

- jedes Blatt v von B einen Zyklus γ_v ohne weitere Angelpunkte in der zu v gehörenden Schleifenkomponente ESN, $v = v(ESN)$, und
- für jeden inneren Knoten v von B mit ausgehender Kante a einen Zyklus γ_v in ESN, welcher den zu a gehörigen Angelpunkt l, $a = a(l)$, von SN enthält.

Nach Lemma 5.11 ist die Stellenmenge der Zyklen γ_v jeweils ein minimaler Siphon R_v in ESN_v, und es gilt

$$\gamma_v = N(R_v, {}^\bullet R_v).$$

Nach Lemma 5.10 bilden die Stellen

$$R := \bigcup_{v \in B} R_v$$

einen minimalen Siphon von SN. Es gilt

$$N(R, {}^\bullet R) = \bigcup_{v \in B} N(R_v, {}^\bullet R_v).$$

Nach Teil 1 ist $N(R, {}^\bullet R)$ ein Schleifenbaum, und nach Konstruktion ist B der Baum der Schleifenkomponenten von $N(R, {}^\bullet R)$

$\diamond$

Bemerkung 5.13 Ein Siphon hat die grundlegende Eigenschaft, daß er durch keine Schaltfolge markiert werden kann, wenn er anfangs unmarkiert ist. Als Konsequenz ergibt sich für einen Schleifenbaum folgender Sachverhalt, der durch den Darstellungssatz 5.12 präzisiert wird:
Um sicherzustellen, daß eine Stelle auf einer Schleife unmarkiert bleibt, dürfen

- die gegebene Schleife
- alle höheren Schleifen auf dem Weg zur Basiskomponente und
- alle auf diesem Weg angehefteten Schleifen

keine Marke tragen. $\diamond$

Korollar 5.14 In einem Schleifenbaum sind die drei Begriffe „Minimaler Siphon", „Minimale Falle" und „S-Komponente" äquivalent, d.h.

1. Eine Teilmenge von Stellen ist genau dann ein minimaler Siphon, wenn sie eine minimale Falle ist.
2. Für jeden minimalen Siphon R ist $N(R, {}^\bullet R)$ eine S-Komponente.
3. Die Stellen einer S-Komponente SK bilden einen minimalen Siphon R mit $SK = N(R, {}^\bullet R)$.

$\square$

Beweis

1) Zum Beweis der Äquivalenz geht man von einem Schleifenbaum $SN = (N, s)$ über zu seinem kanteninversen Netz

$$SN^{-1} := (N^{-1}; s)$$

Das Netz SN^{-1} ist wiederum ein Schleifenbaum — evtl. nach Einfügen von unverzweigten Transitionen vor den Schleifenstellen von SN. Eine Stellenmenge von SN^{-1} ist genau dann ein minimaler Siphon von SN^{-1}, wenn sie als Stellenmenge von SN eine minimale Falle von SN ist. Also gilt der Darstellungsatz 5.12 wörtlich auch für minimale Fallen.

2) Sei R ein minimaler Siphon. Nach dem Darstellungssatz 5.12 ist $N(R, {}^\bullet R)$ ein Schleifenbaum. Für jeden Angelpunkt l dieses Schleifenbaumes betrachten wir die Schleifenkomponente ESN von SN, die an der Stelle l angeheftet ist. Der Durchschnitt

$$N(R \cap ESN, {}^\bullet R \cap ESN)$$

ist jeweils ein Zyklus in ESN, nach Lemma 5.11 also eine S-Komponente in ESN. Nach Lemma 5.10 ist dann auch $N(R, {}^\bullet R)$ eine S-Komponente von SN.

3) Für eine vorgegebene S-Komponente SK und eine Schleifenkomponente ESN_i von SN, die SK trifft, ist

$$SK_i := SN \cap ESN_i$$

nach Lemma 5.10 eine S-Komponente von ESN_i, hat also nach Lemma 5.11 die Gestalt

$$SK_i = N(R_i, {}^\bullet R_i)$$

mit einem minimalen Siphon R_i von ESN_i. Wiederum nach Lemma 5.10 ist dann

$$SK = \bigcup_i SK_i = \bigcup_i N(R_i, {}^\bullet R_i) = N(R, {}^\bullet R)$$

mit einem minimalen Siphon R von SN.

$\diamond$

Satz 5.15 In einem Schleifenbaum gilt:

1. Jeder minimale Siphon ist eine minimale Falle.
2. Jede Falle enthält den Basispunkt.
3. Jede Stelle ist in einer S-Komponente enthalten.
4. Jede S-Komponente enthält den Basispunkt.

$\square$

Beweis
Der Satz ist eine direkte Folgerung aus Korollar 5.14 und Satz 5.12, denn:

- Teil 1 ist die Aussage von Korollar 5.14, Teil 1.
- Teil 2 folgt aus Teil 1 und Satz 5.12, Teil 1b.
- Teil 3 folgt aus Satz 5.12, Teil 2 und Korollar 5.14, Teil 2.
- Teil 4 folgt aus Korollar 5.14, Teil 3 und Satz 5.12, Teil 1.

$\diamond$

Satz 5.15 stellt nun die Verbindung her zwischen Booleschen Netzen mit Booleschen Markierungen auf der einen Seite und Struktureigenschaften des unterliegenden Stellen/Transitionsnetzes auf der anderen Seite. Denn zusammen mit dem Satz von Commoner über lebendige und sichere Markierungen auf Free-Choice Netzen garantiert dieser Satz auf jedem Booleschen Schleifenbaum zumindest zwei Boolesche Markierungen.
Wir geben zunächst die entsprechenden Definitionen und rekapitulieren den Satz von Commoner.

Definition 5.16 [Boolesches Schleifensystem]

1. Wir nennen ein Boolesches Netz $BSN = (SN; x, g)$ einen *Booleschen Schleifenbaum*, wenn das unterliegende Netz SN ein Schleifenbaum ist.
2. Ein Boolesches Netzsystem $BSS = (BSN; BM)$ heißt *Boolesches Schleifensystem*, wenn das Boolesche Netz BSN ein Boolescher Schleifenbaum ist.
3. Wenn wir von
   - Basispunkt
   - Schleifenkomponenten
   - Angelpunkten

   eines Booleschen Netzes sprechen, so meinen wir jeweils den entsprechenden Begriff des unterliegenden Netzes.

   Außerdem sprechen wir von
   - einem elementaren Booleschen Schleifennetz
   - einer Booleschen Schleifenkomponente
   - der Adjunktion von Booleschen Netzen

   und meinen damit jeweils die naheliegende Erweiterung des entsprechenden Netzbegriffes.

$\square$

Satz 5.17 Es sei FCN ein zusammenhängendes Free-Choice Netz.

1. Eine gegebene Markierung M von FCN ist genau dann lebendig, wenn jeder minimale Siphon eine unter M markierte Falle enthält (Commoner).
2. Eine lebendige Markierung von FCN ist genau dann sicher, wenn jede Stelle in einer S-Komponente mit höchstens einer Marke enthalten ist.

$\square$

Beweis
Für Teil 1 siehe [DE95], Chapter 4.3, für Teil 2 siehe [BD90], Corollary 5.6. $\diamond$

Beispiel 5.18 Die Abbildung 5.1 „Siphon, Falle, Komponenten eines Schleifenbaumes“ zeigt einen Schleifenbaum SN mit zwei Schleifenkomponenten ESN_0 und ESN_1. Die Komponente ESN_1 ist am Angelpunkt l_1 an das elementare Schleifennetz ESN_0 angeheftet.

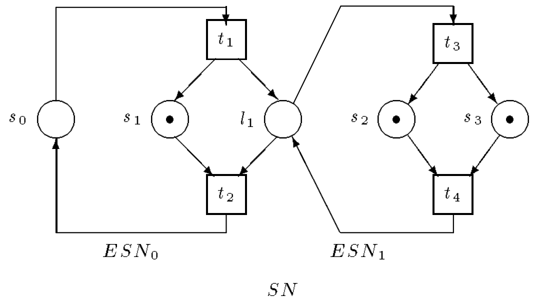


Abbildung 5.1: Siphon, Falle, Komponenten eines Schleifenbaumes

1. Der Schleifenbaum SN hat die beiden T-Komponenten ENS_0 und ESN_1.
2. SN enthält 3 minimale Siphons
   - $R_1 = \{s_0, s_1\}$
   - $R_2 = \{s_0, l_1, s_2\}$
   - $R_3 = \{s_0, l_1, s_3\}$.

   Jeder dieser Siphons ist zugleich eine minimale Falle.

   Die Mengen $\{s_0, l_1\}$ oder $\{l_1, s_2\}$ bilden dagegen keinen Siphon von SN. Allerdings ist jede von ihnen ein Siphon in ihrer jeweiligen T-Komponente.
3. Die erzeugten Teilnetze $N(R, {}^\bullet R)$ bilden die drei S-Komponenten von SN:
   - $N(R_1, {}^\bullet R_1)$ hat die Stellen R_1 und die Transitionen $\{t_1, t_2\}$
   - $N(R_2, {}^\bullet R_2)$ hat die Stellen R_2 und die Transitionen $\{t_1, t_2, t_3, t_4\}$
   - $N(R_3, {}^\bullet R_3)$ hat die Stellen R_3 und die Transitionen $\{t_1, t_2, t_3, t_4\}$.
4. Die angegebene Markierung von NS ist lebendig: Denn jeder der drei minimalen Siphons ist zugleich eine minimale Falle und ist markiert. Die angegebene Markierung ist auch sicher: Denn jede Stelle von N ist in mindestens einer der drei S-Komponenten $N(R_i, {}^\bullet R_i)$, $i = 1, 2, 3$, enthalten, und jede dieser S-Komponenten trägt genau eine Marke.

$\diamond$

Jeder Schleifenbaum — allgemeiner: jedes stellenpunktierte Netz — besitzt die ausgezeichnete Markierung, welche genau den Basispunkt markiert:

Definition 5.19 Für ein stellenpunktiertes Netz $N = ((S, T, A); s)$ nennen wir die ausgezeichnete Markierung des Basispunktes

$$M_s := \chi_s : S \to \mathbb{N}$$

die *Basismarkierung* des punktierten Netzes. Es gilt also

$$M_s(x) = \begin{cases} 1, & x = s \\ 0, & x \neq s \end{cases}$$

$\square$

Satz 5.20 Die Basismarkierung eines Schleifenbaumes ist lebendig und sicher, insbesondere ist jeder Schleifenbaum wohlgeformt. □

Beweis
Die Behauptung folgt sofort aus Satz 5.17 in Verbindung mit Satz 5.15. ◇

Lemma 5.21 [Basismarkierung]

1. Bei einer Netzadjunktion

$$SN = SN_0 \coprod_l SN_1$$

zweier Schleifenbäume gilt für jede lebendige und sichere Markierung M von SN: Wenn M den Angelpunkt l markiert, so ist die Einschränkung $M_{|SN_1}$ die Basismarkierung von SN_1.

2. In einem Schleifenbaum ist die Basismarkierung die einzige lebendige und sichere Markierung, welche den Basispunkt markiert.

□

Beweis

1) Sei M eine lebendige und sichere Markierung von SN mit $M(l) = 1$. Nach Satz 5.17 liegt jede beliebig vorgegebene Stelle $s \in SN_1$ in einer S-Komponente SK von SN mit höchstens einer Marke. Nach Satz 5.15 enthält SK den Basispunkt von SN und damit auch den Angelpunkt l. Es folgt

$$M(s) = 0 \text{ für alle } s \in SN_1, s \neq l.$$

2) Der Beweis folgt analog aus Satz 5.17 und Satz 5.15.

◇

Korollar 5.22 In einem Schleifenbaum ist eine sichere Markierung genau dann lebendig, wenn sie die Basismarkierung als Folgemarkierung besitzt. □

Beweis
Wenn eine sichere Markierung lebendig ist, so hat sie insbesondere eine Folgemarkierung, welche die Vortransition des Basispunktes aktiviert. Nach Schalten dieser Transition erhält man eine Folgemarkierung, welche den Basispunkt markiert, und diese Folgemarkierung ist nach Lemma 5.21 die Basismarkierung. Zur Umkehrung: Die Basismarkierung ist nach Satz 5.20 lebendig. Daher ist jede Markierung mit der Basismarkierung als Folgemarkierung bereits selbst lebendig. ◇

Lemma 5.23 Es sei

$$SN = SN_0 \coprod_l SN_1$$

ein Schleifenbaum, der durch Adjunktion eines Schleifenbaumes SN_1 an einen Schleifenbaum SN_0 am Angelpunkt $l \in SN_0$ entsteht, und es sei M eine lebendige und sichere Markierung von SN. Dann gilt folgende Alternative:

1. Im Falle

$$M_1 := M_{|SN_1} \neq 0$$

ist M_1 eine lebendige und sichere Markierung von SN_1, und

$$M_0 := M_{|SN_0} + \chi_l$$

ist eine lebendige und sichere Markierung von SN_0.

2. Im Falle

$$M_1 := M_{|SN_1} = 0$$

ist

$$M_0 := M_{|SN_0}$$

eine lebendige und sichere Markierung von SN_0.

□

Beweis
Jede Schaltfolge von $SN_i,\ i = 0, 1$, kann auch als Schaltfolge von SN aufgefaßt werden. Somit sind alle betrachteten Einschränkungen der sicheren Markierung M auf $SN_i,\ i = 0, 1$, ebenfalls sichere Markierungen.

1. Im Falle

$$M_1 := M_{|SN_1} \neq 0$$

sei σ eine Schaltfolge von SN minimaler Länge, so daß

$$M_{post} := M[\sigma >$$

den Angelpunkt l markiert. Nach Hilfssatz 5.21 ist dann die Einschränkung

$$M_{post}|SN_1$$

die Basismarkierung von SN_1. Nun ist σ wegen der Minimalität bereits eine Schaltfolge in SN_1. Daher hat M_1 die Basismarkierung von SN_1 als Folgemarkierung und ist nach Korollar 5.22 lebendig.

Offensichtlich ist

$$M_{post}|SN_0 = M_0$$

eine sichere Markierung von SN_0. Als lebendige und sichere Markierung von SN hat M_{post} nach Korollar 5.22 die Basismarkierung als Folgemarkierung. Sei τ eine Schaltfolge in SN minimaler Länge, so daß

$$M_{post}[\tau >$$

die Basismarkierung von SN ist. Wegen der Minimalität ist τ dann bereits eine Schaltfolge in SN_0. Also hat auch M_0 die Basismarkierung von SN_0 als Folgemarkierung und ist daher nach Korollar 5.22 eine lebendige Markierung.

2. Im Falle

$$M_1 := M_{|SN_1} = 0$$

kann man gleich eine Schaltfolge τ von SN minimaler Länge betrachten, so daß

$$M[\tau >$$

die Basismarkierung von SN ist. Man schließt wie in Teil 1, daß τ bereits als Schaltfolge in SN_0 aufgefaßt werden kann, so daß

$$M_0[\tau >$$

die Basismarkierung von SN_0 ist. Nach Korollar 5.22 ist M_0 lebendig.

$\diamond$

Lemma 5.24 Für einen Schleifenbaum

$$SN = SN_0 \coprod_l SN_1,$$

der durch Anheften eines Schleifenbaumes SN_1 an einen Schleifenbaum SN_0 entsteht, gilt die Aussage

> „Jede lebendige und sichere Markierung ist eine Folgemarkierung der Basismarkierung“,

falls die entsprechenden Aussagen für die beiden Komponenten SN_0 und SN_1 gelten. $\square$

Beweis
Es bezeichne jeweils s_i den Basispunkt von SN_i, $i = 0, 1$. Außerdem sei eine lebendige und sichere Markierung M von SN vorgegeben. Wir wenden Lemma 5.23 an. Im Falle

$$M_1 := M_{|SN_1} \neq 0$$

ist M_1 eine lebendige und sichere Markierung von SN_1. Es gibt daher nach Korollar 5.22 eine Schaltfolge σ_1 von SN_1, so daß

$$M_1[\sigma_1 >= \chi_{s_1}$$

die Basismarkierung von SN_1 ist. Außerdem existiert nach Voraussetzung eine Schaltfolge τ_1 von SN_1 mit

$$\chi_{s_1}[\tau_1 >= M_1.$$

Die Markierung

$$M_0 := M_{|SN_0} + \chi_{s_1} = M[\sigma_1 >$$

ist eine lebendige und sichere Markierung von SN_0. Also existiert nach Voraussetzung eine Schaltfolge τ_0 von SN_0 mit

$$\chi_{s_0}[\tau_0 >= M_0.$$

Insgesamt gilt

$$\chi_{s_0}[\tau_0\tau_1 >= M.$$

Also ist M Folgemarkierung der Basismarkierung von SN. $\diamond$

In einem Schleifenbaum bilden alle lebendigen und sicheren Markierungen eine einzige Äquivalenzklasse unter der Erreichbarkeitsrelation. Vorwärts erreichbare Markierungen dieser Äquivalenzklasse sind auch rückwärts erreichbar:

Satz 5.25 In einem Schleifenbaum SN gilt:

1. Jede lebendige und sichere Markierung hat die Basismarkierung als Folgemarkierung.

2. Jede lebendige und sichere Markierung ist eine Folgemarkierung der Basismarkierung.

3. Je zwei lebendige und sichere Markierungen M_1 und M_2 sind Folgemarkierungen voneinander, d.h.

$$M_i \in [M_j > \text{ für } i, j = 1, 2.$$

$\square$

Beweis

1. Siehe Korollar 5.22.

2. Die Behauptung gilt zunächst im Spezialfall eines elementaren Schleifennetzes. Denn in einem T-Netz sind für zwei lebendige und sichere Markierungen M_1 und M_2 äquivalent, vgl. [DE95], Theorem 3.21:

   - $M_2 \in [M_1 >$
   - M_1 und M_2 stimmen auf allen S-Invarianten überein.

   Bezeichnet M_1 eine beliebig vorgegebene lebendige und sichere Markierung und M_2 die Basismarkierung, so gilt nach dem bereits bewiesenen Teil 1

$$M_2 \in [M_1 > .$$

   Also stimmen beide Markierungen auf allen S-Invarianten überein, und es gilt nach demselben Satz auch umgekehrt

$$M_1 \in [M_2 > .$$

   Im allgemeinen Fall eines Schleifennetzes SN folgt die Behauptung durch Traversieren des Baumes der Schleifenkomponenten von SN nach dem postorder-Algorithmus. Dabei wendet man auf den aktuellen Knoten jeweils Lemma 5.24 an.

3. Die Behauptung folgt aus den bereits bewiesenen Teilen 1 und 2.

$\diamond$

Bemerkung 5.26 Es sei $BSN = (SN; x, g)$ ein Boolescher Schleifenbaum mit Basispunkt s.

1. Aufgrund von Satz 5.20 läßt sich die Basismarkierung des unterliegenden Schleifenbaumes SN zu einer Booleschen Markierung BM von BNS liften. Dieser Lift ist eindeutig bestimmt durch die Forderung an die Komponenten von BM

$$BM_0(s) = 0, \; BM_1(s) = 1.$$

Wir nennen diese Boolesche Markierung die Basismarkierung des Booleschen Schleifenbaumes.

2. Die andere ausgezeichnete Boolesche Markierung markiert ebenfalls nur den Basispunkt, und zwar mit der Marke „0“. Wir nennen diese Markierung die Deaktivierung BD des Booleschen Schleifenbaumes.

$\diamond$

Definition 5.27 [Lebendigkeit]

1. Es sei $BNS = (BN; BM)$ ein Boolesches Netzsystem.
   (a) Eine Transition t vonBNS heißt *lebendig*, wenn zu jeder erreichbaren Markierung BM_{pre} von BNS eine Folgemarkierung

   $$BM_{post} \in [BM_{pre} >$$

   existiert, die ein Zuweisungselement (t, b) dieser Transition aktiviert.
   (b) Das Boolesche Netzsystem BNS heißt *lebendig*, wenn jede Transition von BNS lebendig ist.
2. Ein Boolesches Netzsystem $BNS = (BN; BM)$ mit einem aktivierungstreuen Booleschen Netz BN heißt von ordentlichem Verhalten (well behaved), wenn jede Transition t von BN lebendig ist und dabei jeweils ein Zuweisungselement von t

$$(t, b) \text{ mit } b \neq 0$$

aktivierbar ist.

$\square$

Definition 5.28 [Wohlgeformtheit] Ein aktivierungstreuer Boolescher Schleifenbaum heißt *wohlgeformt*, wenn seine Basismarkierung von ordentlichem Verhalten ist. $\square$

Kapitel 6

Modellierung der *or*-Alternative

Eine *or*-Alternative hat wesentlich mehr Entscheidungsmöglichkeiten als die *and*- bzw. die *xor*-Alternative. Denn die *or*-Alternative enthält sowohl die Möglichkeiten der *and*-Alternative als auch die Möglichkeiten der *xor*-Alternative: Bei zwei Variablen gilt

$$x \ or \ y \Leftrightarrow (x \ xor \ y) \ \ xor \ (x \ and \ y).$$

Die Schwierigkeit der *or*-Modellierung wird beim schließenden *or*-Konnektor deutlich. Wir stellen zwei Ansichten über die Modellierung von *or*-Alternativen dar:

1. Die Guard-Formel eines schließenden *or*-Konnektors synchronisiert sowohl die beiden Teilprozese einer *and*-Alternative, als auch die beiden Teilprozesse einer *xor*-Alternative. Bei jedem konkreten Ablauf hat man sich nun bei dem öffnenden *or*-Konnektor für eine der beiden Verzweigungsarten entschieden — entweder verzweigt der Prozeß im Fork-Modus, oder er verzweigt im Branch-Modus. Daher sollte auch der schließende *or*-Konnektor im ersten Fall nur im Join-Modus und im zweiten Fall nur im Merge-Modus schalten können.

2. Die schließende *or*-Transition schaltet undifferenziert, d.h. ohne Unterscheidung von Join- und Merge-Modus. Bei diesem Schaltverhalten können keine Deadlocks auftreten. Sind im Extremfall sogar alle Transitionen eines Booleschen Netzes *or*-Transitionen, so ist jede Boolesche Markierung BM des Netzes bereits von ordentlichem Verhalten. Die einzige Einschränkung an die Markierung BM ergibt sich aus der induzierten Markierung M des unterliegenden Stellen/Transitionsnetzes.

Wir meinen, daß die zweite Sichtweise einige Nachteile enthält:

- Die schließende *or*-Transition läßt eine Reihe von Modellierungsfehlern unerkannt passieren.

- Die schließende *or*-Transition erlaubt zu viele Fälle der Modellierung — mehr Fälle als in der Realität vorkommen.

Es ist bei der Modellierung von *or*-Alternativen nötig, zwischen Branch/Merge-Modus auf der einen Seite und Fork/Join-Modus auf der anderen Seite zu unterscheiden. Solange die schließende *or*-Transition aber nur eine lokale Information auswertet, kann sie diese Unterscheidung nicht treffen.

Als Konsequenz fordern wir zu jedem schließenden *or*-Konnektor einen eindeutig bestimmten öffnenden *or*-Konnektor und v.v. Daher lassen wir nur solche *or*-Alternativen zu, die als Stellenverfeinerung aus einer „elementaren“ *or*-Alternative hervorgehen (Definition 6.2). Durch diese Regel wird insbesondere sichergestellt, daß eine schließende *or*-Transition in demselben Modus schaltet — entweder als Join- oder als Merge-Transition — wie ihre zugehörige öffnende *or*-Transition. Eine weitere Folgerung aus dieser Regel ist es, daß alle zugelassenen *or*-Alternativen gemäß obiger Formel

$$x \ or \ y \Leftrightarrow (x \ \ xor \ y) \ \ xor \ (x \ and \ y)$$

äquivalent sind zu einer analogen *xor*-Schachtelung von *xor*- und *and*-Alternativen.

Definition 6.1 [Stellenverfeinerung]

1. Es seien N_0 ein Netz mit einer unverzweigten Stelle l und N_1 ein Schleifenbaum. Das Netz

$$N := N_0 \widetilde{\coprod}_l N_1$$

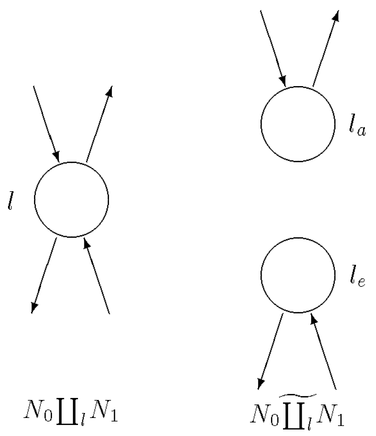
heißt die *Stellenverfeinerung* von N_0 durch N_1 an der Stelle l, wenn man den Angelpunkt l des Netzes

$$N_0 \coprod_l N_1$$

durch zwei neue Stellen l_a und l_e ersetzt (*point splitting*), so daß

$$^\bullet l_a = {}^\bullet l \cap N_0, \ \ l_a^\bullet = l^\bullet \cap N_1$$

$$^\bullet l_e = {}^\bullet l \cap N_1, \ \ l_e^\bullet = l^\bullet \cap N_0$$



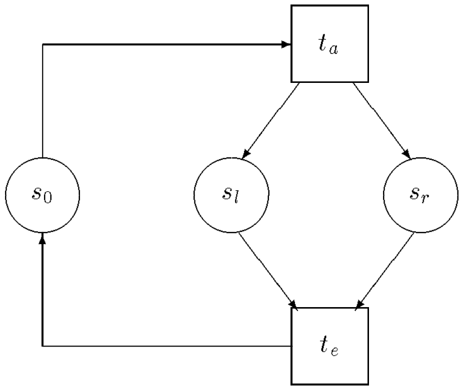
Umgekehrt sagt man, daß N_0 aus N durch *Reduktion* des Teilnetzes N_1 entsteht.

2. Analog definiert man die Stellenverfeinerung eines Booleschen Netzes durch einen Booleschen Schleifenbaum, sowie den Begriff der Reduktion.

□

Definition 6.2 [Logische Alternative]

1. Ein elementares Boolesches Schleifennetz der Form



mit Basispunkt s_0, ausgezeichneten Stellen s_l und s_r und ausgezeichneten Booleschen Transitionen t_a und t_e heißt *elementare logische Alternative*, und zwar:

- elementare *or*-Alternative im Falle von *or*-Transitionen t_a und t_e,
- elementare *xor*-Alternative im Falle von *xor*-Transitionen t_a und t_e,
- elementare *and*-Alternative im Falle von *and*-Transitionen t_a und t_e.

Die Transitionen t_a und t_e heißen (vgl. Bemerkung 3.4)

- im ersten Fall Branch/Fork- bzw. Merge/Join-Transition,
- im zweiten Fall Branch- bzw. Merge-Transition
- und im dritten Fall Fork- bzw. Join-Transition.

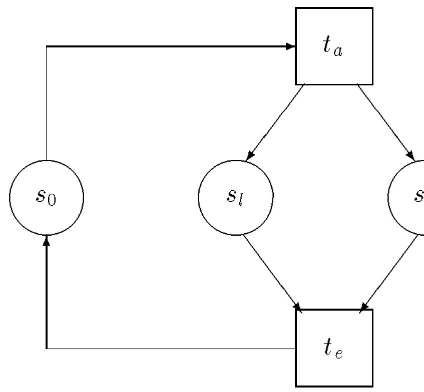
2. Ein Boolesches Netz BNS heißt *logische Alternative*, wenn es die Stellenverfeinerung einer elementaren logischen Alternative BNS_0 durch einen Booleschen Schleifenbaum BNS_1 ist. Im Einzelnen spricht man von *or*-Alternative, *xor*-Alternative und *and*-Alternative.

□

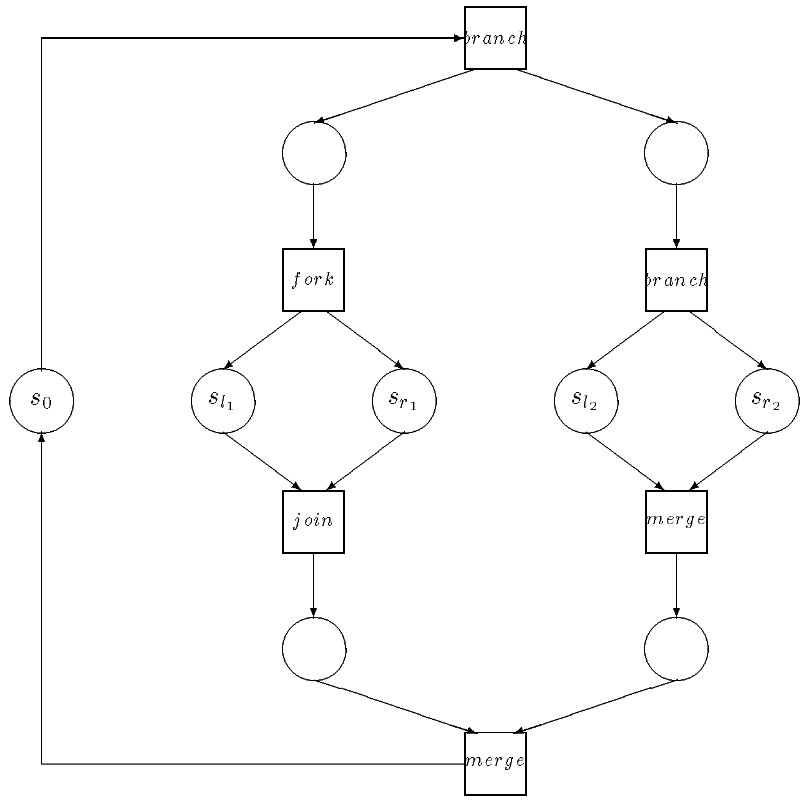
Bemerkung 6.3 Eine öffnende *or*-Transition kann also nur dann eine Branch/Fork-Transition sein, wenn sie eine korrespondierende schließende *or*-Transition (Merge/Join-Transition) besitzt, d.h. beide *or*-Transitionen zusammen bilden eine *or*-Alternative im Sinne obiger Definition. ◇

Definition 6.4 [Branch/Fork-Auflösung]

1. Es sei OA eine elementare *or*-Alternative.



Dann heißt das elementare Schleifennetz $(OA_{bf}; s_{l_1}, s_{l_2}, s_{r_1}, s_{r_2})$



die *Branch/Fork-Auflösung* der elementaren *or*-Alternative OA.

2. Die *or*-Alternative OA sei die Stellenverfeinerung einer elementaren *or*-Alternative OA_{red} an den Stellen s_l und s_r durch jeweils einen Booleschen Schleifenbaum BNS_l bzw. BSN_r. Dann definiert man die Branch/Fork-Auflösung OA_{bf} von OA als die Stellenverfeinerung der Branch/Fork-Auflösung $OA_{red_{fb}}$ von OA_{red} durch

   - das Netz BSN_l an den beiden Stellen s_{l_1} und s_{l_2} und
   - das Netz BSN_r an den beiden Stellen s_{r_1} und s_{r_2}.

□

Bemerkung 6.5 [Wohlgeformtheit]

1. Die Stellenverfeinerung eines aktivierungstreuen Booleschen Schleifenbaumes BNS_0 durch einen aktivierungstreuen Booleschen Schleifenbaum BNS_1 an der Stelle l ist genau dann wohlgeformt, wenn die Adjunktion

$$BSN_0 \coprod_l BSN_1$$

wohlgeformt ist.

2. Jede elementare logische Alternative ist wohlgeformt.

3. Die Branch/Fork-Auflösung einer elementaren *or*-Alternative ist wohlgeformt.

◇

Löst man in einem Booleschen T-Netz, das nur Transitionen der folgenden Art enthält:

- Identische Transition
- Fork- oder Join-Transition
- Branch- oder Merge-Transition
- Fork/Branch- oder Join/Merge-Transition,

alle *or*-Alternativen auf, so erhält man einen „bipolaren Synchronisationsgraphen", wie er von Genrich und Thiagarajan in [GT84] erfunden wurde. Die Autoren führen diese Netzklasse als eine Erweiterung der Klasse der markierten Graphen ein. Ein bipolarer Synchronisationsgraph zusammen mit einer Booleschen Anfangsmarkierung heißt bipolares Synchronisationsschema. Die Autoren zeigen, daß bipolare Synchronisationsschemata von ordentlichem Verhalten eine echte Teilklasse der lebendigen und sicheren Free-Choice Netze bilden. Sie charakterisieren bipolare Synchronisationsschemata mit ordentlichem Verhalten durch einen Satz von Konstruktionsregeln und geben einen Algorithmus an, der in Form einer statischen Netzanalyse prüft, ob ein vorgelegtes bipolares Synchronisationsschema von ordentlichem Verhalten ist.

Definition 6.6 [Bipolares Synchronisationsschema] Ein *bipolarer Synchronisationsgraph BSG* ist ein Boolesches T-Netz, das nur Transitionen der folgenden Art enthält:

- Identische Transition
- Fork- oder Join-Transition
- Branch- oder Merge-Transition.

Ein *bipolares Synchronisationsschema* $BSS = (BSG; BM)$ ist ein Boolesches Netzsystem mit einem bipolaren Synchronisationsgraphen BSG. □

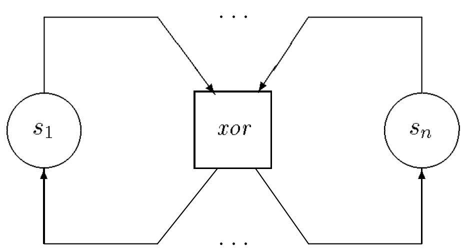
Bemerkung 6.7 Ein bipolares Synchronisationsschema $BSS = (BSG; BM)$ ist genau dann von ordentlichem Verhalten, wenn

- seine Anfangsmarkierung $BM = (BM_0, BM_1)$ wenigstens eine Stelle mit einer „1"-Marke belegt, d.h. $BM_1 \neq 0$, und
- keine Transition unter einer erreichbaren Markierung blockiert.

Die zweite Bedingung ist eine starke Abschwächung des Lebendigkeitsbegriffes. Eine *xor*-Transition blockiert, wenn mindestens zwei Eingänge mit „1"-Marken belegt sind, eine *and*-Transition blockiert, wenn mindestens zwei Eingänge mit verschiedenen Marken belegt sind. ◇

Beweis
[GT84], Theorem 2.12. ◇

Bemerkung 6.8 [Synthesis] Ein bipolares Synchronisationsschema ist genau dann von ordentlichem Verhalten, wenn es durch endliche Anwendung einer Reihe von Konstruktionsregeln $T_1, \ldots, T_8$ aus einem *elementaren bipolaren Synchronisationsschema* entsteht. Dabei hat ein elementares bipolares Synchronisationsschema entweder die Form

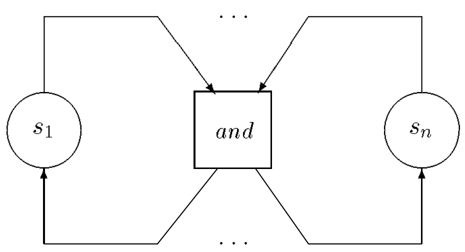


mit n Zyklen, $n \geq 1$, und Anfangsmarkierung

$$BM = (BM_0, BM_1),$$

$$supp(BM_0) = \{s_2, \ldots, s_n\},\ supp(BM_1) = \{s_1\},$$

oder die Form



mit n Zyklen, $n \geq 1$, und Anfangsmarkierung

$$BM = (BM_0, BM_1),$$

$$supp(BM_0) = \emptyset,\ supp(BM_1) = \{s_1, \ldots, s_n\}.$$

◇

Beweis
Siehe [GT84], Theorem 6.19. ◇

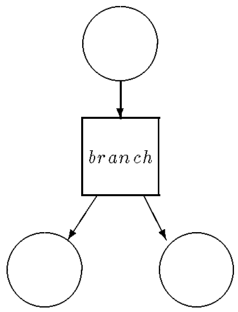
Bemerkung 6.9 [Genrich-Thiagarajan Reduktion] Genrich und Thiagarajan geben in [GT84], Chapter 6.5, einen Reduktionsalgorithmus an, der für ein bipolares Synchronisationsschema von ordentlichem Verhalten nach endlich vielen Schritten terminiert und das Schema auf ein elementares bipolares Synchronisationsschema reduziert. ◇

Bemerkung 6.10 In bipolaren Synchronisationsschemata von ordentlichem Verhalten sind die „0“-Marken überflüssig: Man kann ein gegebenes bipolares Synchronisationsschema BNS von ordentlichem Verhalten in ein äquivalentes Free-Choice Netzsystem FCS übersetzen — wir nennen FCS die *Free-Choice Auflösung* von BPS —, indem man

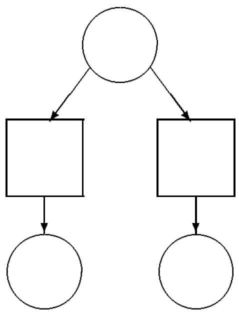
- die Umgebung einer jeden Booleschen Transition von BNS durch ein stellenberandetes Teilnetz ersetzt,
- alle Kantenbeschriftungen von BNS wegläßt,
- alle "1“-Marken von BNS übernimmt und
- alle „0“-Marken von BNS wegläßt.

Für die Ersetzung binärer Boolescher Transitionen gilt im einzelnen:

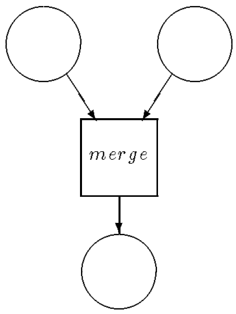
- Eine Branch-Transition



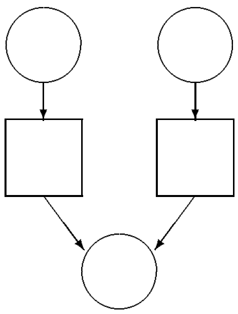
  wird ersetzt durch



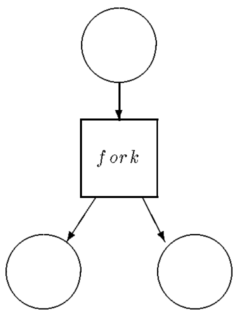
- Eine Merge-Transition



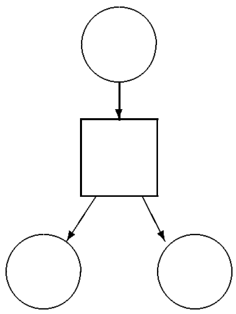
wird ersetzt durch



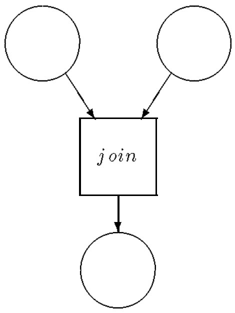
- Eine Fork-Transition



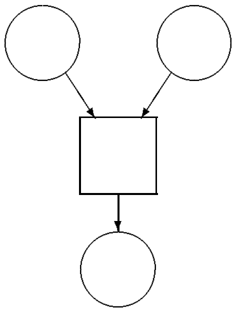
wird ersetzt durch



- Eine Join-Transition



wird ersetzt durch



◇

Satz 6.11 Die Free-Choice Auflösung eines bipolaren Synchronisationsschemas von ordentlichem Verhalten ist lebendig und sicher. □

Beweis
Siehe [GT84], Theorem 3.13. ◇

Kapitel 7

Netzanalyse

Ein Blick auf die betriebliche Praxis zeigt, daß gegenwärtig in vielen kommerziellen Projekten Fachkonzepte verfaßt werden, bei denen die Spezifikation mit EPKs an erster Stelle steht — noch vor dem Datenmodell, ja teilweise sogar ohne explizites Datenmodell. Es gibt Fachkonzepte, bei denen die Modellierung der Geschäftsprozesse mehr als 100 Seiten mit EPKs umfaßt. Aber: Diese EPKs sind lediglich gezeichnet und frei kommentiert — nicht maschinell überprüft.
Projektmanager würden zurecht ein gewisses Unbehagen empfinden, wenn eine vergleichbar große Anzahl von Programmablaufplänen oder Struktogrammen „auf Halde“ produziert würden, ohne daß diese von den Programmierern auch codiert, debuggt und in Komponenten getestet werden.
Netzanalyse und Simulation sind hier die Mittel der Wahl. Sie initiieren folgende Lernschleife zum richtigen Entwurf von EPKs:

- Kundenanforderung
- Modellierung der EPK
- Analyse der EPK
- Korrektur der EPK
- Simulation der EPK zusammen mit dem Kunden.

Nicht zu unterschätzen ist bei diesem iterativen Prozeß auch der Knowhow-Erwerb bei den Software Beratern, wenn diese alle drei Tätigkeiten — Modellierung, Analyse und Simulation — in engem Zusammenhang durchführen.
Ein Tool zum Entwickeln von EPKs sollte daher unbedingt eine Simulationskomponente enthalten, um eine interaktive Analyse und Korrektur von Entwurfsfehlern zum frühestmöglichen Zeitpunkt zu unterstützen. Die vorausgehende Netzanalyse wird die EPK dabei stärker strukturieren, so daß anschließend weniger Fälle für die Simulation übrigbleiben. Beispielsweise ist die Ersetzung von or- Konnektoren durch xor-Konnektoren ein typisches Resultat der Netzanalyse.
In diesem Kapitel präzisieren wir, welche Eigenschaften einer EPK bei der Netzanalyse geprüft werden sollen. Denn nicht jede syntaktisch korrekte EPK modelliert deswegen auch schon einen sinnvollen Prozeß. Zahlreiche Beispiele aus der Praxis machen vielmehr deutlich, wie leicht die syntaktische Freiheit der EPKs bei der Modellierung mißbraucht werden kann. Dabei entstehen dann EPKs, die keinen realen Prozeß mehr darstellen.
Wir gehen davon aus, daß alle EPKs in diesem Kapitel in der Syntax von Definition 1.4 vorliegen und gemäß der Übersetzungsregel von Definition 3.2 normalisiert und in ein Boolesches Netz $BN = (N; x, g)$ auf einem unterliegenden Stellen/Transitionsnetz $N = (S, T; A)$ übersetzt wurden.

Definition 7.1 [Statische Netzanalyse] Die statische Analyse eines Booleschen Netzes $BN = (N; x, g)$ prüft, ob das unterliegende Netz N ein Schleifenbaum ist. □

Bemerkung 7.2 [Eigenschaften von Schleifenbäumen]

1. Ein Schleifenbaum hat insbesondere folgende Eigenschaften:

   - Free-Choice Struktur
   - Zusammenhang
   - Starker-Zusammenhang
   - Zerlegbarkeit in elementare Schleifennetze durch Auftrennen an den Angelpunkten.

   Die Standardalgorithmen der Graphentheorie, vgl. z. B. [Meh84], Chapter IV, lassen sich leicht zu Prüfalgorithmen für die statische Netzanalyse erweitern.

2. Wir haben in Definition 1.4 den unterliegenden Graphen einer EPK als zusammenhängend vorausgesetzt, zur Begründung vgl. auch Bemerkung 1.5, Teil 2.

3. Starker Zusammenhang der normalisierten EPK bedeutet, daß zu jedem Knoten ein gerichteter Pfad führt, der von dem Ereignis „Start/Ziel“ ausgeht, und daß von jedem Knoten ein gerichteter Pfad zu diesem Ereignis hinführt.

   Wenn ein Knoten von dem Ereignis „Start/Ziel“ aus nicht erreicht werden kann, so heißt das, daß die entsprechende Funktion in der Realität nie ausgeführt werden kann, oder daß das entsprechende Ereignis nie eintreten kann. Solche Netzelemente sind für die Modellierung überflüssig.

   Analog bedeutet ein Knoten, von dem aus es keinen Weg zu dem Ereignis „Start/Ziel“ gibt, eine Sackgasse. Solche Funktionen oder Ereignisse verhindern in der Realität, daß der Prozeß sein Ziel erreicht.

4. EPKs ohne innere Schleifen werden in T-Netze übersetzt und haben damit insbesondere die Free-Choice Eigenschaft. Zur Modellierung von inneren Schleifen werden T-Netze miteinander verheftet. Nach eventuellem Einfügen einer Booleschen Transition bleibt dabei die Free-Choice Eigenschaft bei der Verheftung erhalten. Unter allen Petri-Netzen bilden die Free-Choice Netze zur Zeit die am besten untersuchte, nicht-triviale Netzklasse.

   Free-Choice Netze tragen ihren Namen aus folgendem Grund: Im Konfliktfall, bei dem mehrere konkurrierende Transitionen aktiviert sind, hat das Schalten einer dritten Transition keinen Einfluß auf die Entscheidung, welche der am Konflikt beteiligten Transitionen zum Schalten ausgewählt wird. Die Entscheidung (free choice) kann lokal, unabhängig vom übrigen Netzverhalten getroffen werden. Nach dem Schalten dieser Transition bleibt keine der anderen Transitionen teilaktiviert zurück.

◇

Beispiel 7.3 [Beschaffungslogistik] Das Boolesche Netz der Beschaffungslogistik in Abbildung 3.1 ist ein Boolescher Schleifenbaum, ja sogar ein elementares Boolesches Schleifennetz. Denn das unterliegende Netz N enthält keine verzweigten Stellen, es ist ein stark-zusammenhängendes T-Netz. Bezeichnet die Stelle s den Basispunkt von N, d.h. das ausgezeichnete Ereignis „Start/Ziel“, so ist $N \setminus s$ zyklenfrei.
Wir erinnern an die Aussage von Bemerkung 5.26, nach der jeder Boolesche Schleifenbaum $BSN = (SN; x, g)$ zumindest zwei Boolesche Markierungen besitzt, die

Basismarkierung BM, welche genau den Basispunkt s aktiviert, und die Deaktivierung BD, welche genau den Basispunkt deaktiviert, d.h.

$$BM = (BM_0, BM_1) \text{ mit } supp(BM) = \{s\},\ BM(s) = (0,1),$$

und

$$BD = (BD_0, BD_1) \text{ mit } supp(BD) = \{s\},\ BD(s) = (1,0).$$

◇

Lemma 7.4 Ein aktivierungstreuer Boolescher Schleifenbaum BSN ist genau dann wohlgeformt, wenn für seine Basismarkierung BM gilt:

1. BM ist eine Heimatmarkierung von $BSS = (BNS; BM)$

   und

2. Für jede Transition t von BSN existiert eine Folgemarkierung

   $$BM_{pre} \in [BM>,$$

   die ein Zuweisungselement $(t,b),\ b \neq 0$, aktiviert.

□

Beweis

ad 1) Sei BNS wohlgeformt. Dann ist BSS nach Definition 5.28 von ordentlichem Verhalten. Somit existiert für jede erreichbare Markierung

$$BM_{pre} \in [BM>$$

eine Folgemarkierung, welche ein Zuweisungselement $(t,b),\ b \neq 0$, der Vortransition $t = {}^\bullet s$ des Basispunktes s aktiviert. Nach Schalten von t erhält man eine Folgemarkierung

$$BM_{post} \in [BM>,$$

welche insbesondere den Basispunkt markiert. Nach Lemma 5.21 ist BM_{post} entweder die Basismarkierung oder die Deaktivierung BD. Da BNS als aktivierungstreu vorausgesetzt wurde, gilt

$$BM_{post} = BM.$$

Der zweite Teil der Behauptung ist ein Spezialfall der Voraussetzung, daß die Basismarkierung BM von ordentlichem Verhalten ist.

ad 2) Zum Beweis der Umkehrung seien eine erreichbare Markierung

$$BM_{pre} \in [BM>$$

und eine Boolesche Transition t vorgegeben. Dann gilt nach Voraussetzung

$$BM \in [BM_{pre} >,$$

und es existiert eine Folgemarkierung

$$BM_{post} \in [BM >,$$

welche ein Zuweisungselement $(t, b),\ b \neq 0$, aktiviert. Insgesamt folgt

$$BM_{post} \in [BM_{pre} > .$$

◇

Wurde bei der statischen Netzanalyse eines Booleschen Netzes die Struktur des unterliegenden Netzes geprüft, so setzen wir für die anschließende dynamische Netzanalyse voraus, daß das unterliegende Netz ein Schleifenbaum ist. Wir versehen diesen Schleifenbaum mit der Basismarkierung und erhalten damit aus der ursprünglichen EPK ein Boolesches Schleifensystem. Im weiteren Teil dieses Kapitels geht es um das Verhalten dieses Booleschen Schleifensystems.
Die dynamische Netzanalyse hat nun alle Schaltfolgen zu überprüfen, die von der Basismarkierung ausgehen. Durch den grundlegenden Algorithmus von Genrich und Thiagarajan über bipolare Synchronisationsschemata reicht für diese Untersuchung jedoch bereits eine statische Analyse aus. Die Untersuchung des Fallgraphen ist nicht nötig!
Zur Vorbereitung der dynamischen Analyse bringen wir einige Beispiele aus der Praxis, die auf die Notwendigkeit einer konsequenten Durchführung der Verhaltensanalyse des Netzsystems hinweisen.

Bemerkung 7.5 [Modellierungsfehler mit EPK]

1. Die Abbildung 7.1 „Modellierungsfehler mit EPKs" zeigt eine Auswahl möglicher Fehlersituationen bei der Modellierung mit EPKs. Ein elementares Boolesches Schleifennetz ist nicht wohlgeformt, wenn es eines der gezeigten Netze als Teilnetz enthält.

   a) Abschluß einer Branch-Alternative mit einem Merge/Join-Konnektor.
   Der Merge/Join-Konnektor t hat das tote, d.h. nicht erreichbare Zuweisungselement (t, b) mit $b = (1, 1, 1)$.

   b) Abschluß einer Branch-Alternative mit einem Join-Konnektor.
   Der Join-Konnektor mit den Eingangsstellen s_1 und s_2 blockiert unter den erreichbaren Markierungen

   $$BM(s_1) = (0, 1),\ BM(s_2) = (1, 0),$$

   bzw.

   $$BM(s_1) = (1, 0),\ BM(s_2) = (0, 1).$$

   c) Abschluß einer Branch/Fork-Alternative mit einem Merge-Konnektor.
   Der Merge-Konnektor mit den Eingangsstellen s_1 und s_2 blockiert unter der erreichbaren Markierung

   $$BM(s_1) = (0, 1),\ BM(s_2) = (0, 1).$$

d) Falsche Schachtelung von Branch- und Merge-Konnektoren. Jeder der beiden schließenden Merge-Konnektoren kann blockieren, z.B. blockiert der Merge-Konnektor mit den Eingangsstellen s_1 und s_2 unter der erreichbaren Markierung

$$BM(s_1) = (0,1),\ BM(s_2) = (0,1).$$

e) Aussprung aus einer Fork-Alternative mit einem Branch/Fork-Konnektor.

Der Join-Konnektor mit den Eingangsstellen s_1 und s_2 blockiert bei der erreichbaren Markierung

$$BM(s_1) = (1,0),\ BM(s_2) = (0,1).$$

f) Einsprung in eine Branch-Alternative mit einem Merge/Join-Konnektor.

Der Merge-Konnektor mit den Eingangsstellen s_1 und s_2 blockiert bei der erreichbaren Markierung

$$BM(s_1) = (0,1),\ BM(s_2) = (0,1).$$

Darüberhinaus verhindern bei der Schleifenmodellierung einige typische Fehlersituationen, daß ein Boolesches Netz ein Schleifenbaum ist.

g) Schleife mit Einsprung verschieden von Aussprung.

Der Konnektor K_2 für den Einsprung in die Schleife ist verschieden vom Konnektor K_1 des Aussprungs.

Hier soll eine Situation modelliert werden, bei der in jedem Fall einmal vorweg die Funktion F_1 ausgeführt wird und anschließend ggf. in einer Schleife die Sequenz der Funktionen F_2 und F_1.

h) Zusätzlicher Aussprung bzw. Einsprung an beliebiger Stelle des Schleifenrumpfes.

Der Aussprung aus der Schleife am Konnektor K geschieht nicht am Ende des Schleifenrumpfes.

2. Neben diesen formalen Fehlern können bei der Schleifenmodellierung einige typische inhaltliche Fehler vorkommen. Man sollte bei der Schleifenmodellierung zunächst immer den Schleifenrumpf identifizieren. Der Schleifenrumpf ist derjenige Teil, welcher iteriert werden kann.

   Dabei unterscheidet man zwischen einer fußgesteuerten Schleife und einer kopfgesteuerten Schleife: Wenn der Prozeß den Schleifenrumpf mindestens einmal durchläuft und man erst danach die Abbruchbedingung testet, so handelt es sich um eine fußgesteuerte Schleife. Wenn die Abbruchbedingung dagegen schon vor dem ersten Betreten des Schleifenrumpfes getestet wird, so spricht man von einer kopfgesteuerten Schleife.

   Die Modellierung mit EPKs unterstützt kopfgesteuerte Schleifen. Jedoch kann jede fußgesteuerte Schleife in eine kopfgesteuerte Schleife überführt werden, indem man den Schleifenrumpf verdoppelt und vorweg einmal außerhalb der Schleife durchläuft. (Vgl. Abbildung 7.2 „Korrekte Schleifenmodellierung“.) In dieser Abbildung haben wir die Schleife von Abbildung 7.1 Teil g) in ein Boolesches Netz mit einer kopfgesteuerten Schleife übersetzt: Die Funktion F_1 soll in jedem Fall einmal vorweg ausgeführt werden. Daher haben wir sie verdoppelt und zusätzlich vor die Schleife gestellt.

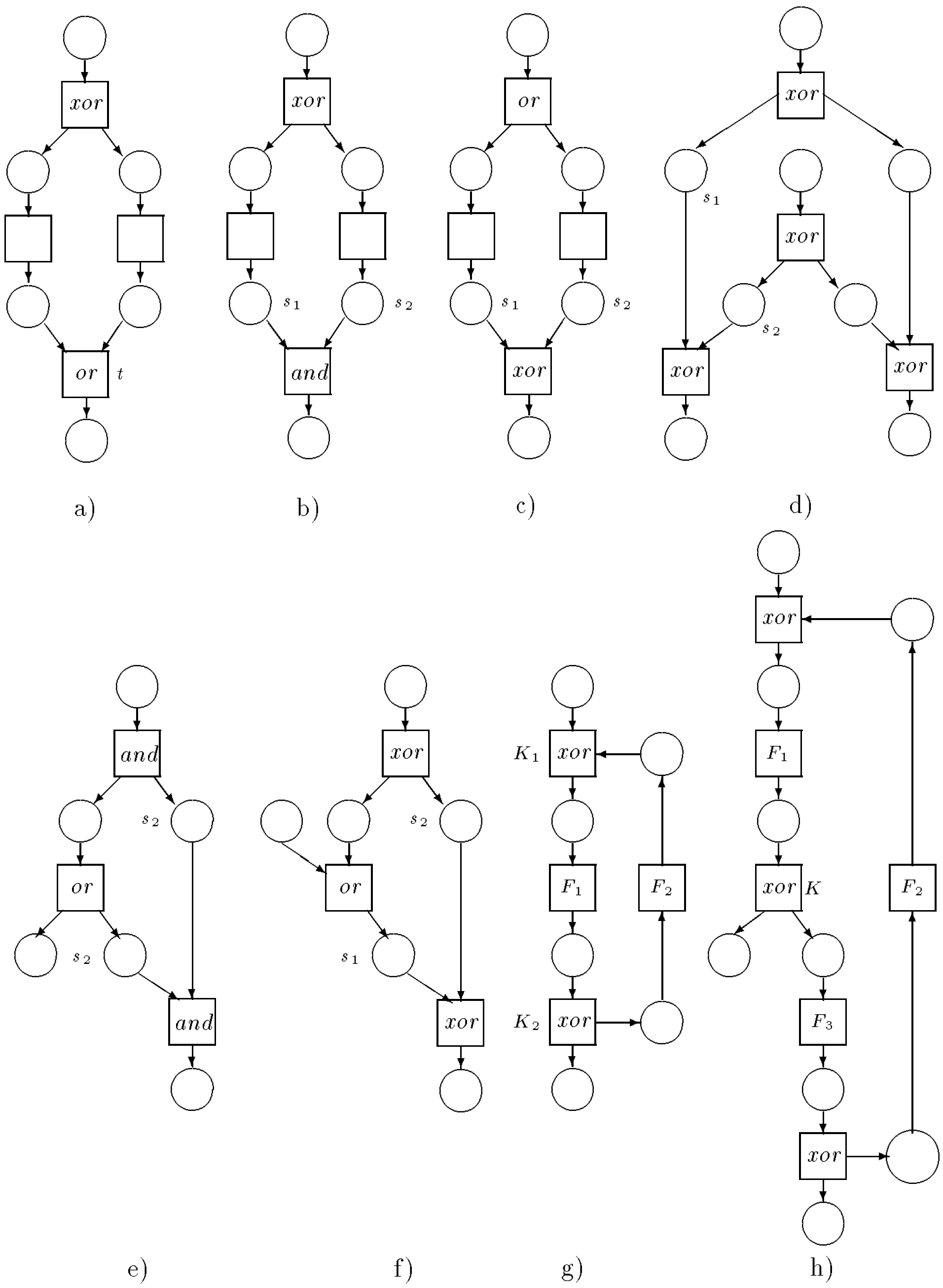


Abbildung 7.1: Modellierungsfehler mit EPK

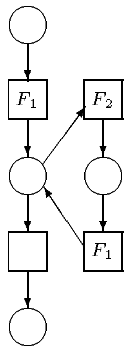


Abbildung 7.2: Korrekte Schleifenmodellierung

Darüberhinaus sind bei der Schleifenmodellierung Aus- und Einsprünge in den Schleifenrumpf an anderer Stelle als nach der Abbruchbedingung überflüssig und können durch eine sorgfältige Modellierung vermieden werden. Die Problematik der Schleifenmodellierung ist jedem vertraut, der z.B. mit Nassi-Shneiderman-Diagrammen arbeitet.

3. Außerdem raten wir dem Modellierer, seine EPKs auch auf folgende Situationen zu überprüfen:

   - Nach einem öffnenden Konnektor müssen die Folgeknoten entweder alle Funktionen oder alle Ereignisse sein. Eine Alternative zwischen einem Ereignis und einer Funktion ist nicht zulässig, vgl. Definition 1.4, Teil 2.
   - Bei einer Alternative zwischen verschiedenen Ereignissen sollten diese Ereignisse auf derselben Abstraktionsstufe stehen.
   - Alle gezeichneten EPKs müssen stellenberandet sein, Randfunktionen sind nicht zugelassen.
   - Viele EPKs enthalten Prozeßwegweiser, die einen Prozeß an anderer Stelle referenzieren. In diesem Falle ist darauf zu achten, daß die Umgebungsereignisse des Prozeßwegweisers mit den Randereignissen des referenzierten Prozesses übereinstimmen.

4. Wir empfehlen den Herstellern von Tools zur EPK-Modellierung, unterschiedliche Netzelemente für die Modellierung von Alternativen und Schleifen vorzusehen. Wir haben zu Beginn von Kapitel 5 auf diesen grundsätzlichen Unterschied hingewiesen.

$\diamond$

Satz 7.6 Die Adjunktion

$$BSN = BSN_0 \coprod_l BSN_1$$

zweier aktivierungstreuer Boolescher Schleifenbäume ist genau dann wohlgeformt, wenn beide Komponenten $BSN_i,\ i = 0, 1$, wohlgeformt sind. $\square$

Beweis
Der Beweis ist elementar. Er beruht auf folgendem Sachverhalt (Lemma 5.21): Solange das Teilnetz BSN_1 markiert ist, kann der Angelpunkt l durch das Schalten

einer Transition von BSN_0 nicht markiert werden. Außerdem verwenden wir die Charakterisierung der Wohlgeformtheit in Lemma 7.4.
Wir bezeichnen mit BM_i, $i = 0, 1$, die Basismarkierungen von BSN_i, und mit BM die Basismarkierung von BSN.

1. Sei BSN wohlgeformt. Wir zeigen, daß BSN_0 wohlgeformt ist.

   - Eine vorgegebene Folgemarkierung der Basismarkierung von BSN_0

     $$BM_{0,pre} \in [BM_0 >$$

     läßt sich erweitern zu einer Booleschen Markierung

     $$BM_{pre} = (BM_{0,pre}, BM_{1,pre}) \in [BM >$$

     von BSN mit

     $$supp(BM_{1,pre}) \subseteq \{l\},\ BM_{1,pre}(l) := BM_{0,pre}(l)$$

     Nach Voraussetzung ist BM in $(BSN; BM)$ eine Heimatmarkierung. Es gibt also eine Schaltfolge σ in BSN minimaler Länge mit

     $$BM_{pre}[\sigma > BM.$$

     Wegen der Minimalität liegen alle Transitionen von σ in BSN_0. Also gilt bereits in BSN_0

     $$BM_{0,pre}[\sigma > BM_0.$$

   - Sei t eine Boolesche Transition in BSN_0. Dann existiert eine Schaltfolge σ in BSN, so daß die Folgemarkierung

     $$BM[\sigma > BM_{post}$$

     ein Zuweisungselement (t, b), $t \neq 0$ aktiviert. Es sei

     $$BM_{post} = (BM_{0,post}, BM_{1,post}).$$

     Man kann annehmen, daß alle Transitionen von σ in BSN_0 liegen, so daß

     $$BM_0[\sigma >$$

     eine gesuchte Folgemarkierung in BSN_0 ist, welche das Zuweisungselement (t, b) aktiviert.

2. Sei BNS wohlgeformt. Wir zeigen, daß BNS_1 wohlgeformt ist.

   Da BSN wohlgeformt ist, existiert eine Schaltfolge σ in BNS, so daß

   $$BM_{pre} := BM[\sigma > BM_{0,post}$$

   die Nachtransition

   $$t = l^\bullet \cap BNS_1$$

des Angelpunktes mit einem Zuweisungselement $b \neq 0$ aktiviert. Da diese Transition aktivierungstreu ist, folgt aus Lemma 5.21, daß die Einschränkung

$$BM_{1,pre} := BM_{pre_{|BSN_1}} = BM_1$$

die Basismarkierung von BSN_1 ist. Man kann annehmen, daß σ eine Schaltfolge in BNS_0 ist. Wir setzen

$$BM_{0,pre} := BM_{pre_{|BSN_0}}.$$

- Seien in BSN_1 eine Schaltfolge σ_1 und eine Markierung $BM_{1,post}$ vorgegeben mit

  $$BM_1[\sigma_1 > BM_{1,post}.$$

  Dann ist

  $$BM_{post} := (BM_{0,pre}, BM_{1,post})$$

  eine Folgemarkierung der Basismarkierung BM von BSN. Nach Voraussetzung und Hilfssatz 7.4 ist die Basismarkeirung BM eine Heimatmarkierung in BSN. Es gibt also eine Schaltfolge τ, so daß

  $$BM_{post}[\tau > BM.$$

  Man kann annehmen, daß mit der Einschränkung

  $$\tau_1 := \tau_{|BSN_1}$$

  bereits

  $$BM_{1,post}[\tau_1 > BM_1$$

  die Basismarkierung von BSN_1 erzeugt, dabei wird die Aktivierungstreue von BSN_1 ausgenützt. Also ist BM_1 eine Heimatmarkierung von BSN_1.

- Sei t eine Boolesche Transition von BSN_1. Dann existiert eine Schaltfolge σ in BSN mit $BM[\sigma > BM_{post}$, so daß

  $$BM_{post} = (BM_{0,post}, BM_{1,post})$$

  ein Zuweisungselement (t, b), $b \neq 0$, von t aktiviert. O. E. kann man annehmen, daß σ eine Zerlegung besitzt

  $$\sigma = \sigma_1 \circ \sigma_0,$$

  mit Schaltfolgen σ_i in BSN_i, $i = 0, 1$, so daß

  $$BM[\sigma_0 > (BM_{0,post}, BM_1)$$

  und

  $$BM_1[\sigma_1 > BM_{1,post}.$$

3. Es seien die beiden Komponenten BSN_i, $i = 0, 1$, wohlgeformt. Wir zeigen, daß BSN wohlgeformt ist.

   - Vorgegeben sei eine Schaltfolge σ in BSN und eine Boolesche Markierung von BSN mit

     $$BM[\sigma > BM_{post}, BM_{post} = (BM_{0,post}, BM_{1,post}).$$

     Falls die Komponente $BM_{1,post}$ keine Stelle von $BNS_1 \setminus \{l\}$ markiert, folgt bereits aus der Wohlgeformtheit von BSN_0, daß $BM = BM_0$ eine Folgemarkierung von BM_{post} ist.
     Andernfalls kann man annehmen, daß eine Zerlegung

     $$\sigma = \sigma_{0,2} \circ \sigma_{1,1} \circ \sigma_{0,1}$$

     existiert mit Schaltfolgen $\sigma_{0,2}$ und $\sigma_{0,1}$ in BSN_0 und $\sigma_{1,1}$ in BSN_1, so daß für

     $$BM_{pre} = (BM_{0,pre}, BM_{1,pre}) \text{ mit } BM[\sigma_{0,1} > BM_{pre}$$

     die Komponente $BM_{1,pre}$ entweder die Basismarkierung BM_1 oder die Deaktivierung BD_1 ist,

     $$BM_{1,pre}[\sigma_{1,1} > BM_{1,post}$$

     und

     $$BM_{0,pre}[\sigma_{0,2} > BM_{0,post}.$$

     Nun existiert eine Schaltfolge $\sigma_{1,2}$ in BSN_1 mit

     $$BM_{1,post}[\sigma_{1,2} > BM_{1,pre};$$

     im Falle $BM_{1,pre} = BD_1$ verwendet man die Aktivierungstreue von BSN_1 zusammen mit Satz 5.20, und im Falle $BM_{1,pre} = BM_1$ verwendet man die Wohlgeformtheit von BSN_1. Dann gilt

     $$BM[\sigma_{1,2} \circ \sigma_{0,2} \circ \sigma_{1,1} \circ \sigma_{0,1} > BM_{0,post}$$

     $$BM[\sigma_{0,2} \circ \sigma_{0,1} > BM_{0,post}$$

     $$BM_0[\sigma_{0,2} \circ \sigma_{0,1} > BM_{0,post}$$

     mit der Schaltfolge $\sigma_{0,2} \circ \sigma_{0,1}$ in BSN_0. Nach Voraussetzung ist BSN_0 wohlgeformt, also existiert eine Schaltfolge $\sigma_{0,3}$ in BSN_0, so daß

     $$BM_{0,post}[\sigma_{0,3} > BM_0.$$

     Insgesamt haben wir die Basismarkierung BM in BSN vermöge

     $$BM_{post}[\sigma_{0,3} \circ \sigma_{1,2} > BM$$

     als Folgemarkierung von BM_{post} dargestellt.

- Sei t eine Boolesche Transition von BNS. Falls t in BNS_0 liegt, existiert in BNS_0 eine Schaltfolge σ, so daß

$$BM_0[\sigma > BM_{0,post}$$

ein Zuweisungselement (t, b), $b \neq 0$, aktiviert. Da σ auch eine Schaltfolge in BSN ist, aktiviert die Folgemarkierung von BM

$$BM[\sigma > BM_{post}$$

ebenfalls das Zuweisungselement (t, b). Falls die Transition t in BSN_1 liegt, wählen wir in BSN_0 eine Schaltfolge σ_0, so daß

$$BM_0[\sigma_0 > (BM_{0,pre}, BM_1),$$

und dann in BSN_1 eine Schaltfolge σ_1, so daß

$$BM_1[\sigma_1 > BM_{1,post}$$

ein Zuweisungselement (t, b), $b \neq 0$, aktiviert. Dann ist $\sigma_1 \circ \sigma_0$ eine Schaltfolge in BSN, so daß

$$BM[\sigma_1 \circ \sigma_0 > BM_{post}$$

das Zuweisungselement (t, b) aktiviert.

$\diamond$

Korollar 7.7 Ein aktivierungstreuer Boolescher Schleifenbaum ist genau dann wohlgeformt, wenn alle seine Schleifenkomponenten wohlgeformt sind. $\square$

Beweis
Zu jedem Knoten v des Baumes $BS(BSN)$ der Schleifenkomponenten des aktivierungstreuen Booleschen Schleifenbaumes BSN bezeichne $BSN(v) \subseteq BS(BSN)$ den maximalen Booleschen Schleifenbaum, dessen Basiskomponente $EBSN$ zu v gehört, d.h. $v = v(ESBN)$.
Aus Satz 7.6 folgt als Vorüberlegung: $BSN(v)$ ist genau dann wohlgeformt, wenn das elementare Boolesche Schleifennetz $EBSN_v$ und die Booleschen Schleifenbäume $BSN(w)$ aller Söhne w von v wohlgeformt sind.

1. Der Schleifenbaum BSN sei wohlgeformt. Man traversiert den Baum $BS(BSN)$ preorder und wendet auf den aktuellen Knoten v jeweils die Vorüberlegung an.

2. Alle Booleschen Schleifenkomponenten von BSN seien wohlgeformt. Dann traversiert man den Baum $BS(SN)$ postorder und wendet auf den aktuellen Knoten v jeweils die Vorüberlegung an.

$\diamond$

Korollar 7.8 Für einen Boolescher Schleifenbaum BSN, der nur Boolesche Transitionen der folgenden Art enthält:

- Identische Transition

- Branch- oder Merge-Transition
- Fork- oder Join-Transition
- Branch/Fork- oder Merge/Join-Transition,

sind äquivalent:

- BSN ist wohlgeformt.
- Jede Boolesche Schleifenkomponente von BSN hat als Branch/Fork-Auflösung ein bipolares Synchronisationsschema, das unter der Basismarkierung von ordentlichem Verhalten ist.

□

Beweis
Der Beweis folgt aus Korollar 7.7 und Bemerkung 6.5. ◇

Definition 7.9 [Dynamische Netzanalyse] Die dynamische Analyse prüft einen Booleschen Schleifenbaum,

1. ob er nur Transitionen der folgenden Art enthält:
   - Identische Transition
   - Branch- oder Merge-Transition
   - Fork- oder Join-Transition
   - Branch/Fork- oder Merge/Join-Transition,
2. und ob er wohlgeformt ist.

□

Als Zusammenfassung unserer Resultate erhalten wir:

Satz 7.10 [Dynamische Analyse einer EPK] Die dynamische Analyse des Booleschen Schleifensystems

$$BSS = (BSN; BM),$$

das aus der Übersetzung einer EPK resultiert, kann auf folgende statische Analyse des unterliegenden Booleschen Schleifenbaumes BSN zurückgeführt werden:

- Man zerlegt BSN durch Auftrennen seiner Angelpunkte in endlich viele elementare Boolesche Schleifenbäume.
- Man prüft jeden elementaren Booleschen Schleifenbaum, ob seine or-Transitionen Branch/Fork- bzw. Merge/Join-Transitionen sind.
- Für jeden elementaren Booleschen Schleifenbaum mit or-Alternativen bildet man die Branch/Fork-Auflösung und erhält einen bipolaren Synchronisationsgraphen.
- Man prüft jeden dieser bipolaren Synchronisationsgraphen mit der Genrich-Thiagarajan Reduktion, ob er wohlgeformt ist.

Der Boolesche Schleifenbaum BSN ist genau dann wohlgeformt, wenn alle resultierenden bipolaren Synchronisationsgraphen wohlgeformt, d.h. unter der Basismarkierung von ordentlichem Verhalten sind. □

Beispiel 7.11 [Beschaffungslogistik]

1. Das Boolesche Netz der Beschaffungslogistik in Abbildung 3.1 ist nicht wohlgeformt. Die Prüfung ergibt zunächst zwei schließende *or*-Transitionen K_{121} und K_7, die keine Merge/Join-Transitionen sind. Was wollte der Modellierer ausdrücken, welcher Vorgang der Realität soll abgebildet werden?

   - Transition K_{121} ist vermutlich das Gegenstück zur Branch-Transition K_8. Bei K_8 verzweigt der Prozeß der Qualitätsprüfung in zwei Alternativen, die sich gegenseitig ausschließen und die bei K_{121} wieder synchronisiert werden. Daher ändern wir die *or*-Transition K_{121} in eine *xor*-Transition K_{123}.
   - Die *or*-Transition K_7 ist schwieriger zu interpretieren. Vermutlich soll hier eine Korrektur der Wareneingangsdaten modelliert werden für den Fall, daß die Qualitätsprüfung negativ verläuft.

     Es gibt hier in der Realität mehrere Vorgehensweisen:

     Alternative 1:

     Vor der Buchung des Wareneingangs soll in jedem Fall der Ausgang der Qualitätsprüfung abgewartet werden. Die Funktion F_9, „Wareneingangsbuchung", wird nur ein einziges Mal durchgeführt, ggf. nach vorheriger Korrektur der Werte durch die Funktion F_{10} „Reklamation.erstellen, Datum.korrigieren".

     Alternative 2:

     Unabhängig von dem Ergebnis der Qualitätsprüfung wird zunächst der Wareneingang gebucht. Sollte die Qualitätsprüfung danach eine Änderung der Daten erfordern, wird zusätzlich eine Korrekturbuchung durchgeführt.

     Wir meinen, daß Alternative 2 das Vorgehen in der Praxis besser wiedergibt und werden nur sie weiterverfolgen. Die vorliegende EPK von Abbildung 1.1 bzw. Abbildung 1.2 modelliert diesen Fall nicht, weil hier Transition K_7 in jedem Fall das Ergebnis der Qualitätsprüfung abwartet. Wir führen daher eine weitere Funktion F_{95} ein, „Korrekturbuchen", und diese Funktion ersetzt zugleich die Booleschen Transitionen K_{110} und K_7. Als Konsequenz werden außerdem die beiden *and*-Konnektoren K_{91} und K_{92} durch einen einzigen *and*-Konnektor K_{97} ersetzt.

     Einfacher zu behandeln sind dagegen die Branch/Fork-Transitionen K_{101}, K_{102} und ihre zugehörigen Merge/Join-Transitionen K_{104} und K_{103}. Wir bilden zunächst für die äußere *or*-Alternative mit den Transitionen K_{101} und K_{104} die Branch/Fork-Auflösung. Anschließend bilden wir die Branch/Fork-Auflösund der inneren *or*-Alternative.

     Als Resultat entstehen die Konnektoren K_{150} – K_{167} sowie durch Stellenverdopplung

     - von E_1 die Stellen E_{100}, E_{101},
     - von E_2 die Stellen $E_{200}, E_{201}, E_{202}, E_{203}$,
     - von E_3 die Stellen $E_{300}, E_{301}, E_{302}, E_{303}$.

     Die Branch/Auflösung kann natürlich als Algorithmus implementiert werden. Sie bereitet das Netz auf als bipolaren Synchronisationsgraphen für die anschließende Analyse mit dem Algorithmus von Genrich und Thiagarajan. Falls diese Analyse ein positives Ergebnis liefert, wird die Branch/Fork-Auflösung wieder reduziert.

     Das Resultat unserer Änderung zeigt Abbildung 7.3. Das Boolesche Netz enthält jetzt keine *or*-Alternativen mehr und ist ein bipolarer Synchronisationsgraph. Nach Aktivierung seines Basispunktes „Start/Ziel" erhalten wir ein bipolares Synchronisationsschema.

Man kann dieses bipolare Synchronisationsschema nun mit dem Algorithmus von Genrich und Thiagarajan reduzieren, vgl. Bemerkung 6.9. Bei dem vorgelegten System von Abbildung 7.3 terminiert der Algorithmus mit einem elementaren bipolaren Synchronisationsschema und zeigt damit, daß das Ausgangsnetz von Abbildung 7.3 wohlgeformt ist. Die dynamische Netzanalyse ist beendet.

Warum liefert der Algorithmus dieses Resultat? Man überzeugt sich leicht, daß für die Frage nach der Wohlgeformtheit des bipolaren Synchronisationsgraphen aus Abbildung 7.3 allein die Booleschen Transitionen K_{100}, K_{122}, K_{97} und K_{200} maßgeblich sind. Die dazwischen liegenden Netzteile lassen sich auf jeweils eine Zwischenstelle reduzieren. Den reduzierten bipolaren Synchronisationsgraphen zeigt Abbildung 7.4. Hier kann man die Wohlgeformtheit direkt nachrechnen, oder auch die Reduktionsregel „R2" (node reduction) des Algorithmus von Genrich und Thiagarajan anwenden.

2. Die Netzanalyse hat ergeben, daß der Boolesche Schleifenbaum „Beschaffungslogistik" von Abbildung 7.3 wohlgeformt ist. Man kann nun die Branch/Fork-Auflösungen wieder reduzieren und außerdem die binären Booleschen Transitionen mit der Fusionsregel von Definition 4.5 zu komplexeren Booleschen Transitionen zusammenfassen:

   - Aus der Reduktion der Branch/Fork-Auflösung und der anschließenden Fusion der *or*-Konnektoren resultieren die *or*-Transitionen K_{105} und K_{106}.
   - Die *xor*-Transition K_{123} und die *and*-Transition K_{122} fusionieren zu einer Booleschen Transition K_{125}.

   Das Resultat zeigt Abbildung 7.5 „Bipolares Synchronisationsschema Beschaffungslogistik nach Transitionsfusion".

◇

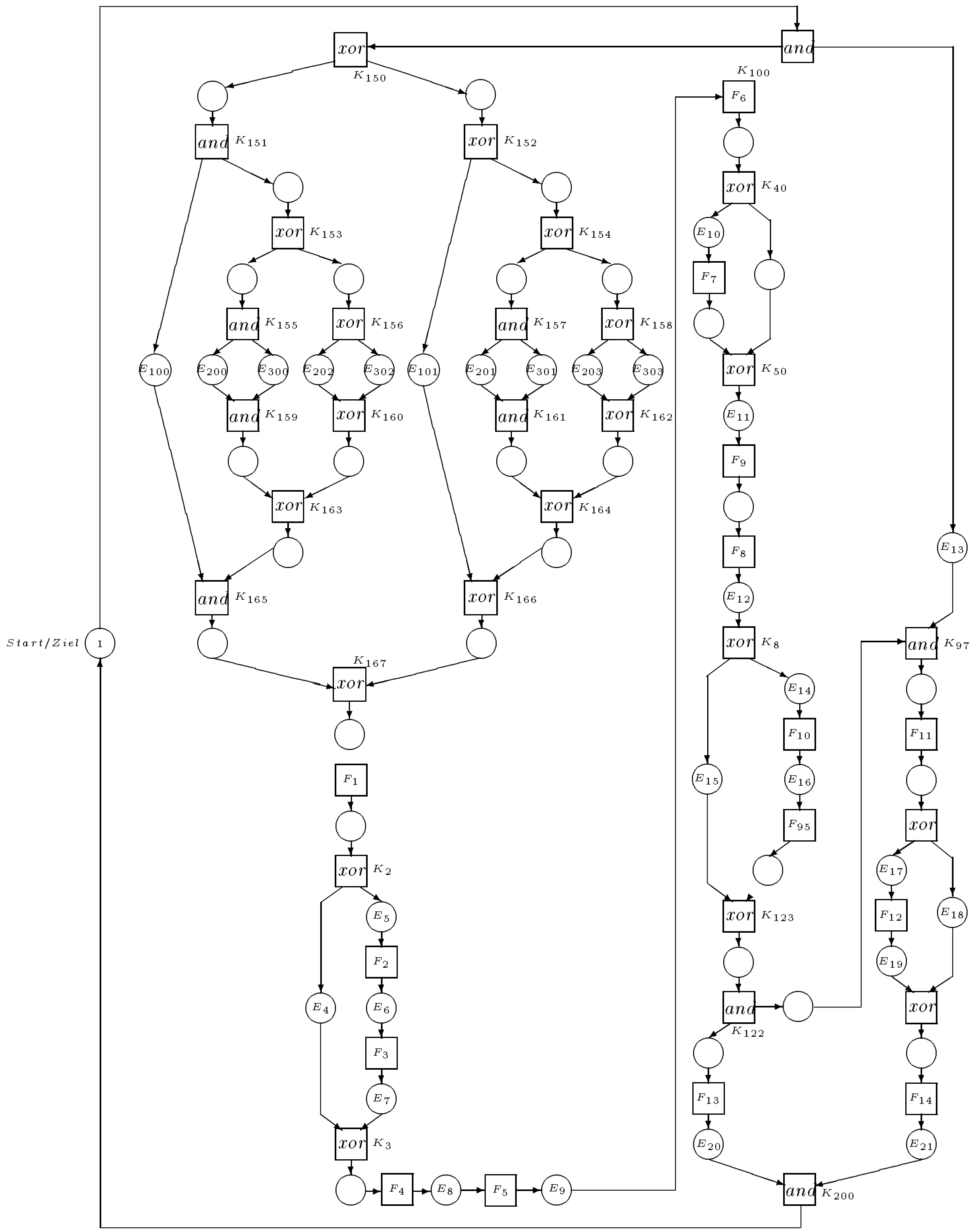


Abbildung 7.3: Bipolares Synchronisationsschema Beschaffungslogistik

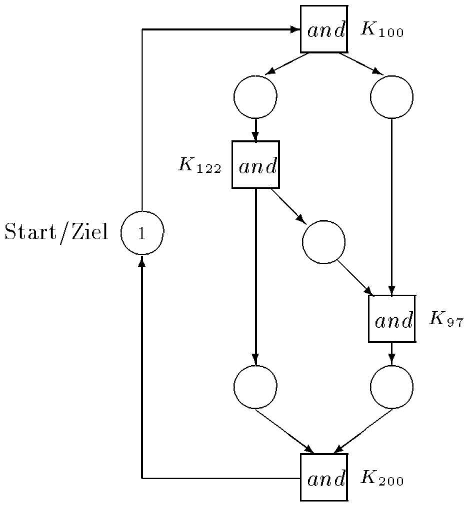


Abbildung 7.4: Netzanalyse Beschaffungslogistik

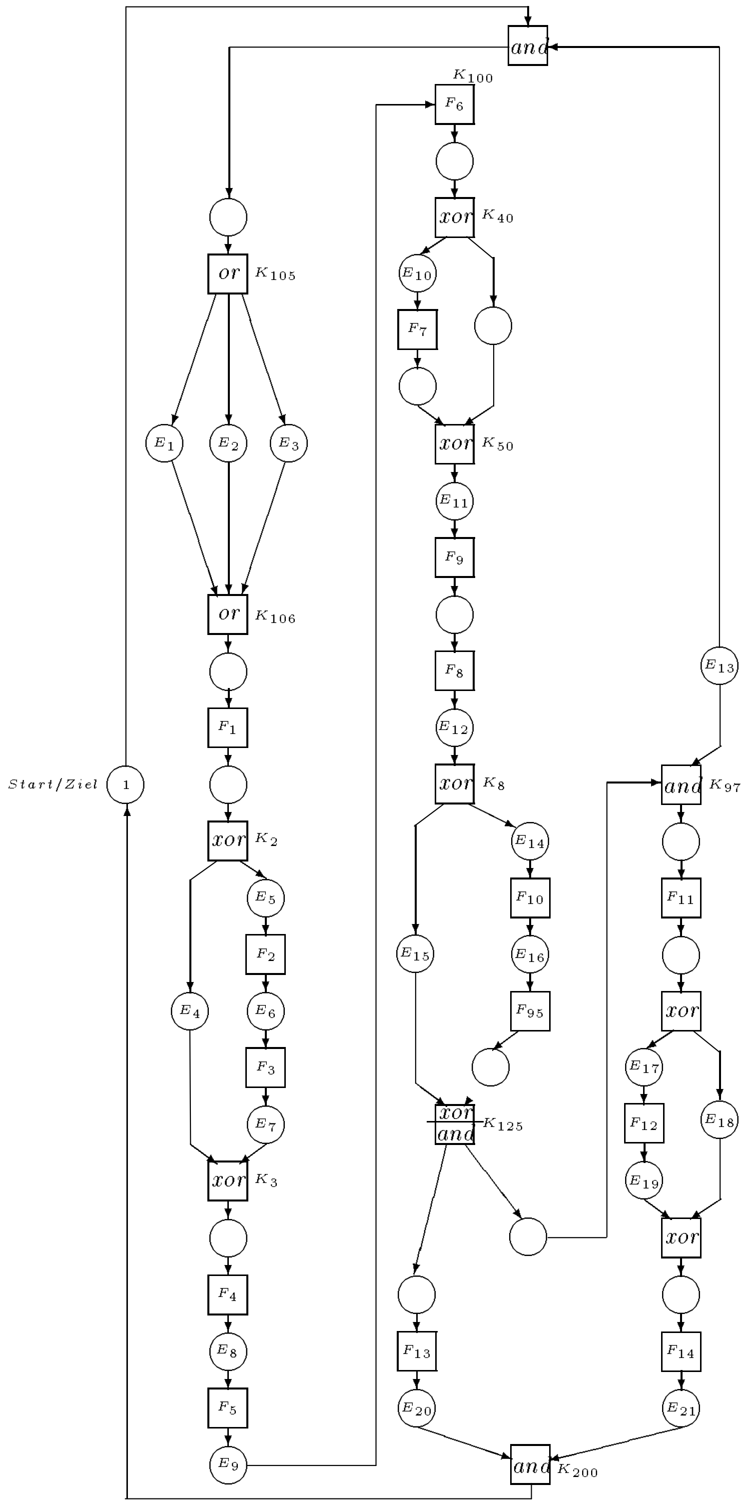


Abbildung 7.5: Bipolares Synchronisationsschema Beschaffungslogistik nach Transitionsfusion

Abbildungsverzeichnis

Literaturverzeichnis

[AM69] M.I. Atiyah und I.B. Macdonald. *Introduction to Commutative Algebra.* Addison Wesley, 1969.

[Bau90] B. Baumgarten. *Petri-Netze. Grundlagen und Anwendungen.* BI-Wissenschaftsverlag, 1990.

[BD90] E. Best und J. Desel. Partial order behavior and structure of Petri nets. *Formal aspects of computing*, (2):123–138, 1990.

[Brö96] A. Bröker. Transformation ereignisgesteuerter Prozeßketten in Prädikat-Transitions-Netze sowie Vergleich der Modellierungstools ARIS-Toolset und INCOME. Diplomarbeit, Hochschule für Gestaltung, Technik und Wirtschaft, 1996.

[CS94] R. Chen und A.-W. Scheer. Modellierung von Prozeßketten mittels Petri-Netz-Theorie. Heft 107, Institut für Wirtschaftsinformatik, Saarbrücken, 1994.

[DE95] J. Desel und J. Esparza. *Free Choice Petri Nets.* Cambridge University Press, 1995.

[GT84] H. Genrich und P. Thiagarajan. A Theory of Bipolar Synchronization Schemes. *Theoretical Computer Science*, (30):241–318, 1984.

[Jen92] K. Jensen. *Coloured Petri Nets. Basic Concepts*, Band 1. Springer Verlag, Berlin Heidelberg New York, 1992.

[Jen95] K. Jensen. *Coloured Petri Nets. Analysis Methods*, Band 2. Springer Verlag, Berlin Heidelberg New York, 1995.

[KM94] G. Keller und S. Meinhard. *SAP R/3-Analyser, Optimierung von Geschäftsprozessen auf Basis des R/3-Referenzmodells.* SAP AG, Walldorf, 1994.

[KNS91] G. Keller, M. Nüttgens und A.-W. Scheer. Semantische Prozessmodellierung auf der Grundlage „Ereignisgesteuerter Prozessketten (EPK)“. Heft 89, Institut für Wirtschaftsinformatik, Saarbrücken, 1991.

[Meh84] K. Mehlhorn. *Data structures and Algorithms 2: Graph Algorithms and NP-Completeness.* Springer Verlag, Berlin Heidelberg New York, 1984.

[Obe96] A. Oberweis. *Modellierung und Ausführung von Workflows mit Petri-Netzen.* Teubner, 1996.

[Rei86] Wolfgang Reisig. *Petrinetze – Eine Einführung.* Springer Verlag, Berlin Heidelberg New York, 2. Auflage, 1986.

[Reu95] B. Reuter. *Direkte und indirekte Wirkungen rechnerunterstützter Fertigungssysteme. Formalisierte Netzstrukturen zur Darstellung und Analyse der Unternehmung.* Physica-Verlag, 1995.

[Sch94] A.-W. Scheer. *Wirtschaftsinformatik.* Springer-Verlag, Berlin Heidelberg New York, 5. Auflage, 1994.

[Sch96] A.-W. Scheer. *Geschäftsprozeßmodellierung innerhalb einer Unternehmensarchitektur.* In: G. Vossen and J. Becker (Hrsg.), Geschäftsprozeßmodellierung und Workflow-Management. Modelle. Methoden, Werkzeuge, International Thomson Publishing, 1996.

[SNZ95] A.-W. Scheer, M. Nüttgens und V. Zimmermann. Rahmenkonzept für ein integriertes Geschäftsprozeßmanagement. *Wirtschaftsinformatik*, (37):426–434, 1995.

Index