

Institut für Informatik, Humboldt-Universität zu Berlin

Dissertation

Entwurf und Verifikation von Petrinetzmodellen verteilter Algorithmen durch Verfeinerung unverteilter Algorithmen

zur Erlangung des akademischen Grades Doktor der Naturwissenschaften
(doctor rerum naturalium) im Fach Informatik

Mathematisch-Naturwissenschaftlichen Fakultät II

MSc. Bixia Wu

geboren am 18. Juli 1966 in Provinz Fujian, China

Dekan: Prof. Dr. Wolfgang Coy

Gutachter: 1. Prof. Dr. Wolfgang Reisig, Humboldt-Universität zu Berlin, Institut für Informatik
2. Prof. Dr. Bodo Hohberg, Humboldt-Universität zu Berlin, Institut für Informatik
3. Prof. Dr. Jörg Desel, Katholische Universität Eichstätt, Math.-Geograf Fakultät

eingereicht: 30. März 2007
Datum der Promotion: 22. Juni 2007

Zusammenfassung

Um Entwurf und Verifikation komplizierter verteilter Algorithmen leichter und verständlicher zu machen, wird oft eine Verfeinerungsmethode verwendet. Dabei wird ein einfacher Algorithmus, der gewünschte Eigenschaften erfüllt, schrittweise zu einem komplizierten Algorithmus verfeinert. In jedem Schritt sollen die gewünschten Eigenschaften erhalten bleiben.

Für nachrichtenbasierte verteilte Algorithmen haben wir eine neue Verfeinerungsmethode entwickelt. Wir beginnen mit einem Anfangsalgorithmus, der Aktionen enthält, die gemeinsame Aufgaben mehrerer Agenten beschreiben. In jedem Schritt verfeinern wir eine dieser Aktionen zu einem Netz, das nur solche Aktionen enthält, die die Aufgaben einzelner Agenten beschreiben. Jeder Schritt ist also eine Verteilung einer unverteiltern Aktion.

Die Analyse solcher Verfeinerungsschritte wird mit Hilfe eines neuen Verfeinerungsbegriffs - der verteilenden Verfeinerung - durchgeführt. Entscheidend dabei ist das Erhaltenbleiben der Halbordnungen des zu verfeinernden Algorithmus. Dies ist durch Kausalitäten der Aktionen der Agenten im lokalen Verfeinerungsnetz zu erreichen.

Die Kausalitäten im lokalen Verfeinerungsnetz lassen sich einerseits beim Entwurf direkt durch Nachrichtenaustausch realisieren. Andererseits kann man bei der Verifikation die Gültigkeit einer Kausalität im lokalen Verfeinerungsnetz direkt vom Netz ablesen. Daher ist diese Methode leicht zu verwenden.

Die Anwendung der Methode wird in der Arbeit an verschiedenen nicht trivialen Beispielen demonstriert.

Schlagwörter:

Petrinetze, Verifikation, Verfeinerung, Verteilte Algorithmen

Abstract

In order to make design and verification of complicated distributed algorithms easier and more understandable, a refinement method is often used. A simple algorithm, which fulfills desired properties, is refined stepwise to a complicated algorithm. In each step the desired properties are preserved.

For messages-based distributed algorithms we have developed a new refinement method. We begin with an initial algorithm, which contains actions, which describe common tasks of several agents. In each step we refine one of these actions to a net, which contains only such actions, which describe the tasks of individual agents. Thus, each step is a distribution of an undistributed action.

The analysis of such refinement steps is accomplished with the help of a new refinement notation - the distributing refinement. **Preservation** of the partial **order** of the refined algorithm is important. This can be achieved by causalities of the actions of the agents in the local refinement net.

Causalities in the local refinement net can be realized on the one hand at design directly by messages passing. On the other hand, at verification one can read the validity of causality in the local refinement net directly from the net. Therefore, this method is easy to use.

The application of the method is demonstrated by several nontrivial examples in this thesis.

Keywords:

Petri Nets, Verification, Refinement, Distributed Algorithms

Inhaltverzeichnis

1. Einführung	7
2. Grundlagen.....	26
2.1 Petrinetze	26
2.2 Verteilte Abläufe	28
2.3 Eigenschaften von verteilten Algorithmen	33
2.4 Kausalitäten im System	39
2.4.1 Definition von Kausalitäten im System.....	40
2.4.2 Ableseregeln für die Kausalitäten im System	42
3. Halbordnung erhaltende Transitionsverfeinerung.....	53
3.1 Verfeinerungsbegriffe, eine Übersicht.....	53
3.1.1 Der erste Verfeinerungsbegriff.....	54
3.1.2 Beobacherverfeinerung.....	56
3.1.3 Vogler's Transitionsverfeinerung.....	57
3.1.4 Vogler's Transitionsverfeinerung und unsere verteilende Verfeinerung.....	59
3.2 Transitionsverfeinerung mit Blockbedingung	62
3.3 Eigenschaften der Blockbedingung	68
3.4 Was bleibt nicht erhalten.....	70
3.5 Erhaltenbleiben der Kausalitäten in einem System	72
3.5.1 Erhaltenbleiben der Kausalordnung	72
3.5.2 Erhaltenbleiben der Kausalitäten im System.....	77
3.5.3 Erhaltenbleiben von Eigenschaften durch Kausalitäten	79
4. Agentensysteme	85
4.1 Warum Agentensysteme	85
4.2 Definition der Agentensysteme.....	92
4.2.1 Was ist ein Agent in einem Petrinetz	92
4.2.2 Was ist ein Kanal in Agentensystemen?	96
4.2.3 Definitionen Agent, Kanal, Agentensystem	99
4.3 Agentensysteme als Anfangsnetze für den Entwurf von	

verteilten Algorithmen - am Beispiel des verteilten <i>Mutex</i>	102
4.3.1 Korrektheit des Anfangsalgorithmus Σ_0	103
4.3.2 Entwurf des Anfangsalgorithmus	109
4.3.3 Nebenläufiges Programm des Anfangsalgorithmus	112
5. Verteilende Verfeinerung.....	114
5.1 Warum verteilende Verfeinerung	114
5.2 Verteilende Verfeinerung	117
5.2.1 Verteilende Systemtransformation.....	117
5.2.2 Synchronisations-Bedingung und verteilende Verfeinerung 127	
5.3 Kausalitäten im lokalen Teilsystem	138
5.4 Erhaltenbleiben von Eigenschaften durch Synchronisationsbedingungen	140
5.4.1 Erhaltenbleiben von Sicherheitseigenschaften	140
5.4.2 Erhaltenbleiben von Lebendigkeitseigenschaften	145
6. Nachweis der Blockbedingung bei der verteilenden Verfeinerung - Grundtypen von Verfeinerungsnetzen	148
6.1 Einführung	148
6.2 Abgesprochene gemeinsame Aktion	150
6.3 Erbetene Mithilfe	162
6.4 Erwartete Mithilfe.....	169
6.5 Vertauschbarkeit von Verfeinerungsschritten	177
6.6 Umgebung, für die es keine korrekte verteilende Verfeinerung gibt 181	
7. Entwurf und Verifikation verteilter Algorithmen durch verteilende Verfeinerung und Anwendungsbeispiele	184
7.1 Verifikation verteilter Algorithmen durch verteilende Verfeinerung.....	185
7.1.1 Verifikation durch verteilende Verfeinerung	185
7.1.2 Anwendungsbeispiel – <i>Token-Passing-Mutex</i> Algorithmus	

186	
7.1.3	Anwendungsbeispiel – <i>asymmetrischer Mutex</i> -Algorithmus
196	
7.2	Entwurf verteilter Algorithmen durch verteilende
Verfeinerung.....	205
7.2.1	Entwurf durch verteilende Verfeinerung
7.2.2	Anwendungsbeispiel <i>crossstalk</i> -Algorithmus.....
	205
Literaturverzeichnis	230
Danksagung	235

1. Einführung

Wenn man eine komplizierte Aufgabe hat, kann man sie meist nicht auf einmal erledigen, man muss *Schritt für Schritt* die Aufgabe lösen, z.B. beim Hausbauen. Der Hausbau ist schon ein grosses Projekt, man muss es schrittweise machen. Am Anfang hat man einen groben Plan, z.B. man weiß ungefähr, Mauern, Dach, Fenster, Türen usw. müssen gebaut werden. Später plant man genauer, z.B. zu der Aufgabe Dach gibt es Teilaufgaben: Dachstuhl, Isolation, Dachziegel usw. Also man *verfeinert* die Aufgaben. Bei einem grossen Projekt kann man meist nicht alles alleine machen sondern man muß mit anderen zusammen arbeiten. Beim Verfeinern der Aufgaben *verteilt* man auch die Aufgaben, z.B. den Dachstuhl soll ein Zimmermann machen und für die Dachziegel ist der Dachdecker zuständig. Also, Verfeinern und Verteilen gehören zusammen.

Schrittweise Verfeinerung verwendet man nicht nur im Alltag sondern auch im wissenschaftlichen Bereich Informatik (vgl. [Wir71], [Hoa99]). Z. B. beim Programmieren schreibt man am Anfang einen Namen für ein Unterprogramm, später schreibt man dann dieses Unterprogramm. Auch bei Petrinetzen verwendet man schrittweise Verfeinerung (vgl. [Val79], [Vog92], [BGV90], [BDE93], [Peu01], [PU03]), um verteilte Algorithmen zu entwerfen und zu verifizieren. Am Anfang hat man ein grobes Modell mit Aktionen, später verfeinert man diese Aktionen zu Teilnetzen. Aktionen nennt man in Petrinetzen *Transitionen*, also Transitionen werden zu *Teilnetzen* verfeinert.

Transitionsverfeinerung

In dieser Arbeit geht es darum, wie wir durch Transitionsverfeinerung komplizierte verteilte Algorithmen entwerfen und verifizieren (Analyse verteilter Algorithmen siehe z.B. [Rei98],[Lyn96],[Tel91]). Wir beginnen mit einem einfachen Algorithmus, der die geforderten Eigenschaften erfüllt. Wir verfeinern dann diesen Algorithmus schrittweise bis zu dem verteilten Zielalgorithmus. Bei jedem Verfeinerungsschritt sollen die geforderten Eigenschaften erhalten bleiben. In dieser Arbeit betrachten wir *nachrichtenbasierte verteilte Algorithmen* (vgl. z.B. [Des97], [LL90]).

Im Folgenden werden wir anhand eines Beispiels den wesentlichen Inhalt der Arbeit vorstellen. Das Beispiel ist *Wassersprung-Training*. Ein *Wasserspringer* trainiert regelmäßig in einer Schwimmhalle. Jedesmal vor dem Training wird der *Hallenwart* das Wasser einlassen, damit das

Schwimmbecken nicht leer ist. Nach dem Training wird er das Wasser wieder ablassen.

In Abb. 1.1 ist ein Petrinetz, das das System *Wasserspringer* modelliert.

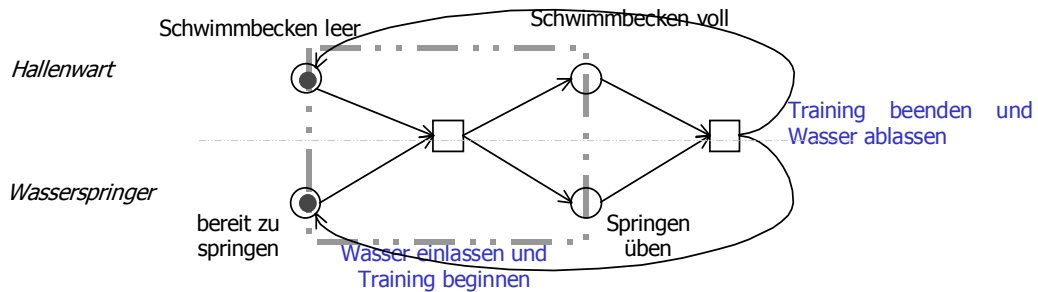


Abb. 1.1 *Wasserspringer*

Am Anfang ist der Wasserspringer bereit zu springen, wir markieren die Stelle *bereit-zu-springen*, indem wir an diese Stelle ein Token legen. Und am Anfang ist das Schwimmbecken leer, also es liegt an der Stelle *Schwimmbecken-leer* auch ein Token. Das ist ein Zustand (*Anfangszustand*). In diesem Zustand kann die Aktion *Wasser-einlassen-und-Training-beginnen* ausgeführt werden. Das nennt man auch: Die Transition *Wasser-einlassen-und-Training-beginnen* kann schalten. Nach dem Schalten dieser Transition wird das Schwimmbecken voll sein und der Sportler springen üben, also an der Stelle *Schwimmbecken-voll* und der Stelle *Springen-üben* liegt jeweils ein Token. Das ist wieder ein Zustand. In diesem Zustand kann die Transition *Training-beenden-und-Wasser-ablassen* schalten. Danach kommt das System zurück zu dem Anfangszustand. So läuft das System. Das beschreiben wir durch einen verteilten Ablauf (Abb. 1.2).

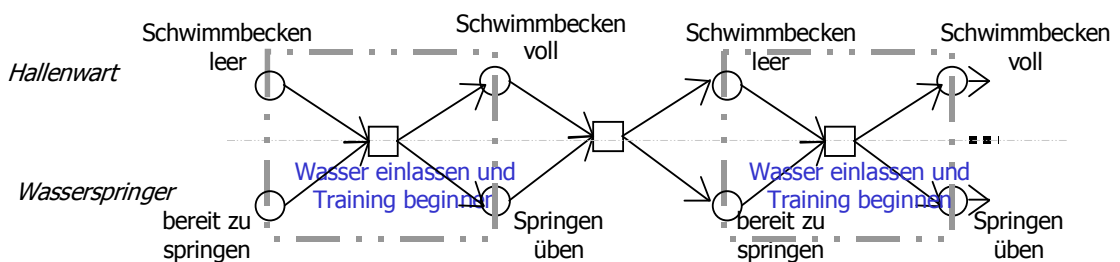


Abb. 1.2 Ablauf des Systems *Wasserspringer*

Im Ablauf sehen wir, die Transition *Wasser-einlassen-und-Training-beginnen* kommt immer wieder vor. Das Vorkommen einer Transition nennen wir das *Auftreten* dieser Transition.

In diesem System gibt es in jedem erreichbaren Zustand an jeder Stelle höchstens ein Token. Wir sagen, das System ist *1-beschränkt*. Wir betrachten in dieser Arbeit 1-beschränkte Systeme.

Das System hat die folgende Eigenschaft: Wenn der Sportler bereit zu springen ist und das Schwimmbecken leer ist, wird irgendwann das Schwimmbecken voll sein und der Sportler Springen üben. Das ist eine *Lebendigkeitseigenschaft*. Wir formulieren diese mit folgender Formel:

$$\text{Schwimmb.-leer} \wedge \text{bereit-zu-sp.} \triangleright \text{Schwimmb.-voll} \wedge \text{Springen-ü.} \quad (1-1)$$

Das System hat noch eine *Sicherheitseigenschaft* (genaue Definitionen siehe Abschnitt 2.3): Wenn der Sportler springt, dann ist das Schwimmbecken sicherlich voll. Wir schreiben:

$$\square (\text{Springen-üben} \rightarrow \text{Schwimmbecken-voll}) \quad (1-2)$$

Für das System sind diese Eigenschaften (1-1), (1-2) gewünschte Eigenschaften.

Was ist Transitionsverfeinerung? Wir können z.B. die Transition *Wasser-einlassen-und-Training-beginnen* durch ein Teilnetz N_{t_1} ersetzen, in dem es zwei Transitionen *Wasser-einlassen* und *Training-beginnen* gibt (Abb. 1.3). Wir sagen, wir verfeinern die Transition *Wasser-einlassen-und-Training-beginnen* zu dem Teilnetz N_{t_1} . In diesem Verfeinerungsschritt sollen die gewünschten Eigenschaften erhalten bleiben.

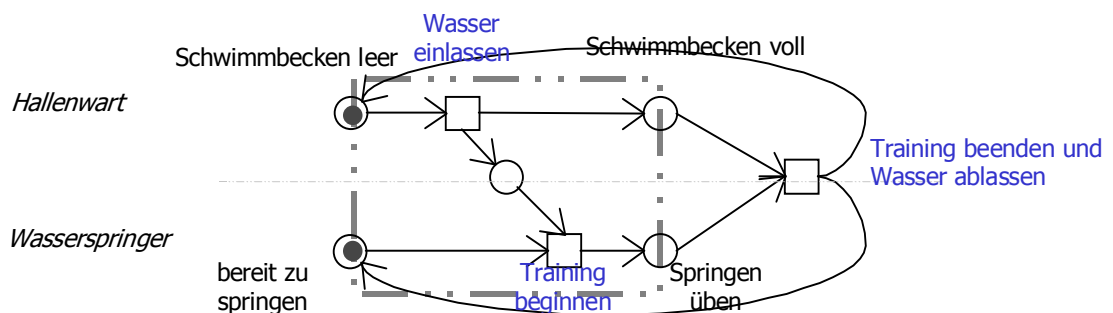


Abb. 1.3 System Sicherer Wasserspringer

Wir können noch weiter die Transition *Training-beenden-und-Wasser-ablassen* zu dem Netz N_{t_2} verfeinern, das eine Transition *Training-beenden* und eine Transition *Wasser-ablassen* enthält (Abb. 1.4). Auch in diesem Verfeinerungsschritt sollen die gewünschten Eigenschaften erhalten bleiben.

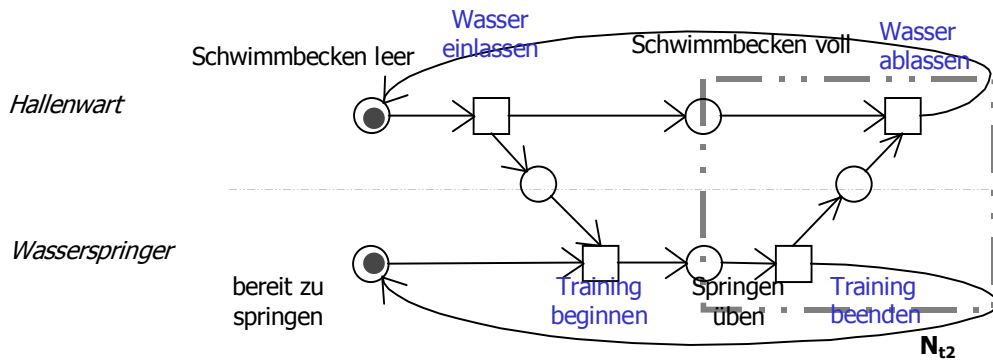


Abb. 1.4 System *vollkommen verteilter Wasserspringer*

Wir haben schon gesehen, wie eine Transitionsverfeinerung aussieht. Wir brauchen einen passenden Verfeinerungsbegriff. Zunächst sehen wir uns ein bisschen genauer an, was für eine Verfeinerung wir haben möchten. Die Transition *Wasser-einlassen-und-Training-beginnen* ist eine gemeinsame Aktion des *Wasserspringers* und des *Hallenwarts*. Wir möchten diese Aktion so verfeinern, dass nur noch Aktionen des *Wasserspringers* oder des *Hallenwarts* modelliert werden und die gewünschten Eigenschaften erhalten bleiben. Also wir suchen nach einem passenden Verfeinerungsbegriff.

Was waren in der Literatur für Transitionsverfeinerungen zu finden? Ein Verfeinerungsbegriff ist Voglers Transitionsverfeinerung (vgl. [Vog87], [Vol92]). Bei dieser Verfeinerung gibt es aber im Verfeinerungsnetz eine Anfangstransition, die von allen Agenten, also hier dem *Wasserspringer* und dem *Hallenwart*, zusammen ausgeführt wird (Abb. 1.5). Und es gibt noch eine Endtransition, die auch wieder zu allen Agenten gehört. Das heißt, nach der Verfeinerung gibt es immer noch gemeinsame Aktionen von beiden Agenten. Wir können also mit diesem Verfeinerungsbegriff die Transition *Wasser-einlassen-und-Training-beginnen* nicht verteilen.

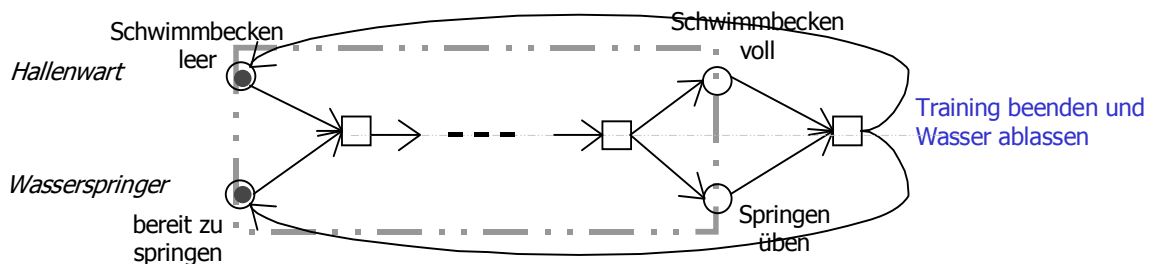


Abb. 1.5 *Verfeinerter Wasserspringer* nach Voglers Transitionsverfeinerung

(Vogler hat seinen Verfeinerungsbegriff auf verteilte Systeme erweitert. Dort fordert er eine Eingabe- und Ausgabesynchronisation für das

Verfeinerungsnetz, was für unser Problem ähnliche Auswirkungen hat.)

Ein weiterer Verfeinerungsbegriff ist Peuker's Transitionsverfeinerung (vgl. [Peu01]). Das ist eine Transitionsverfeinerung mit sogenannter *Blockbedingung*. Nach diesem Verfeinerungsbegriff ist das System *Gefährdeter Wasserspringer* in Abb. 1.6 eine korrekte Transitionsverfeinerung des Systems *Wasserspringer*.

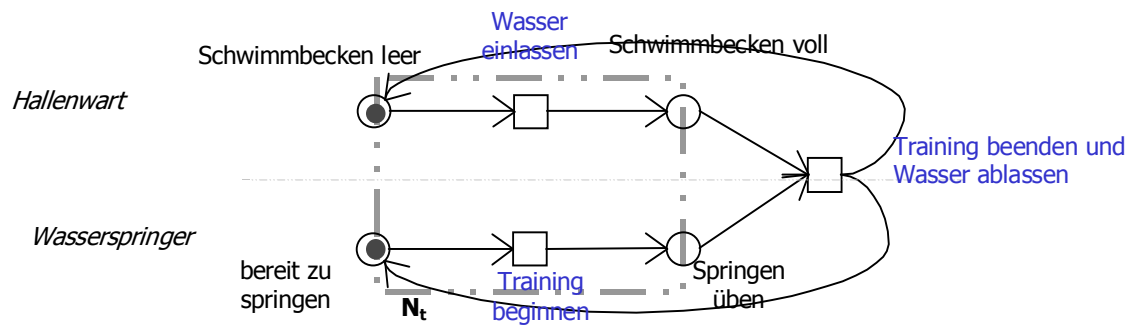
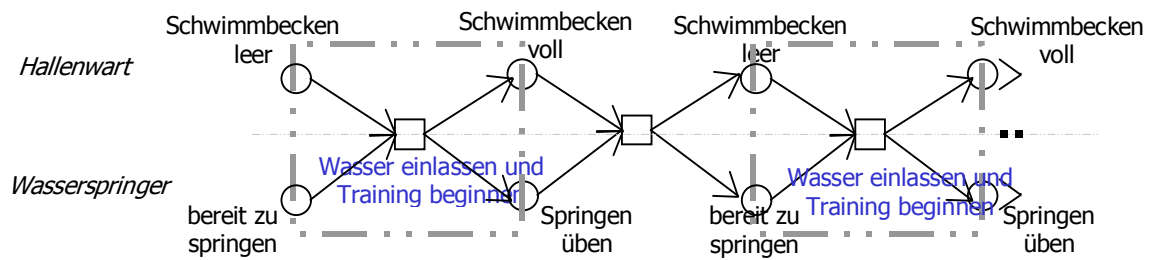
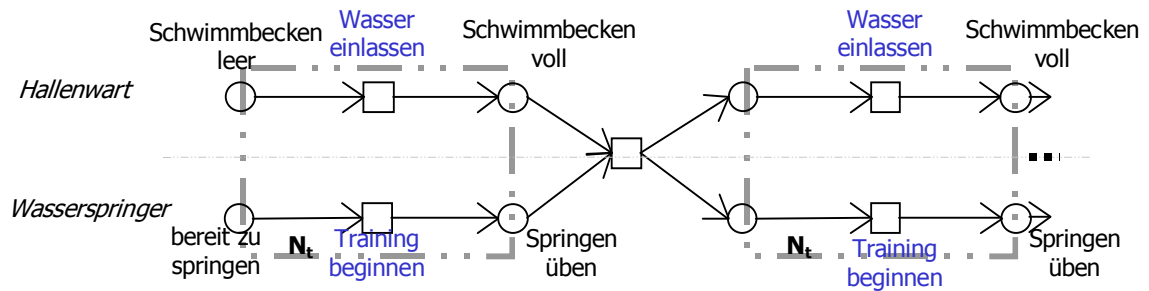


Abb. 1.6 *Gefährdeter Wasserspringer*

Wir erklären zunächst, was die Transitionsverfeinerung mit Blockbedingung ist. In Abb. 1.7 sind der Ablauf vor der Verfeinerung und der Ablauf nach der Verfeinerung dargestellt. Im Ablauf K vor der Verfeinerung (siehe Abb. 1.7(a)) gibt es mehrere Auftreten der Transition *Wasser-einlassen-und-Training-beginnen*. Im Ablauf K' nach der Verfeinerung (siehe Abb. 1.7(b)) ist das Auftreten der Transition durch das Auftreten des Netzes N_t (Verfeinerungsnetz) ersetzt. Wenn das System Σ' nach der Verfeinerung vom System Σ nur solche Abläufe hat, dann sagen wir, das System Σ' erfüllt die Blockbedingung. Wenn das System Σ' die Blockbedingung erfüllt, dann ist Σ' eine Transitionsverfeinerung von Σ .



(a) Ablauf des Systems *Wasserspringer*



(b) Ablauf des Systems *Gefährdeter Wasserspringer*

Abb. 1.7 Ablauf des Systems *Wasserspringer* und des verfeinerten Systems *Gefährdeter Wasserspringer*

Abb. 1.8 veranschaulicht den Begriff Transitionsverfeinerung mit Blockbedingung.

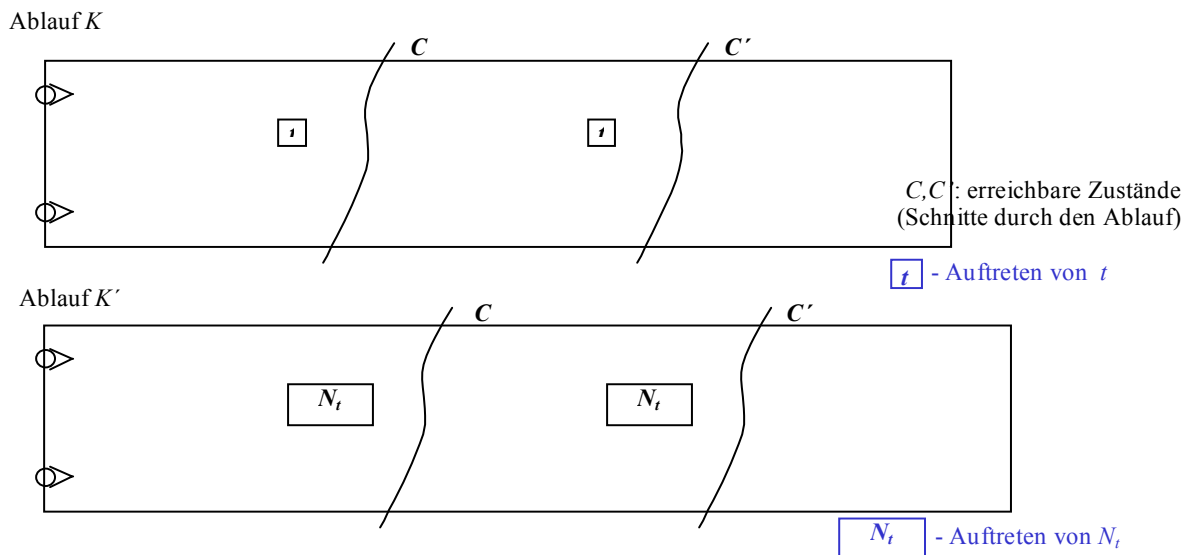


Abb. 1.8 Illustration für die Blockbedingung

Welche Eigenschaften bleiben bei diesem Verfeinerungsbegriff erhalten?

Z.B. hier bleibt die gewünschte Lebendigkeitseigenschaft (1-1) erhalten. Wenn *Schwimmb.-leer* gilt und *bereit-zu-sp.* gilt, wird tatsächlich irgendwann

Schwimmb.-voll und *Springen-ü.* gelten. Allgemein erhält die Blockbedingung die folgende Eigenschaft: Alle erreichbaren Zustände im alten System vor der Verfeinerung bleiben im neuen System nach der Verfeinerung erreichbar, und die Reihenfolge von Zuständen in Abläufen bleibt erhalten (siehe Abb. 1.8). Also die gewünschte Lebendigkeitseigenschaft bleibt bei diesem Verfeinerungsbegriff erhalten.

Reicht das für unser Beispiel? Wir sehen in Abb. 1.9, was im System *Gefährdeter Wasserspringer* passieren kann.

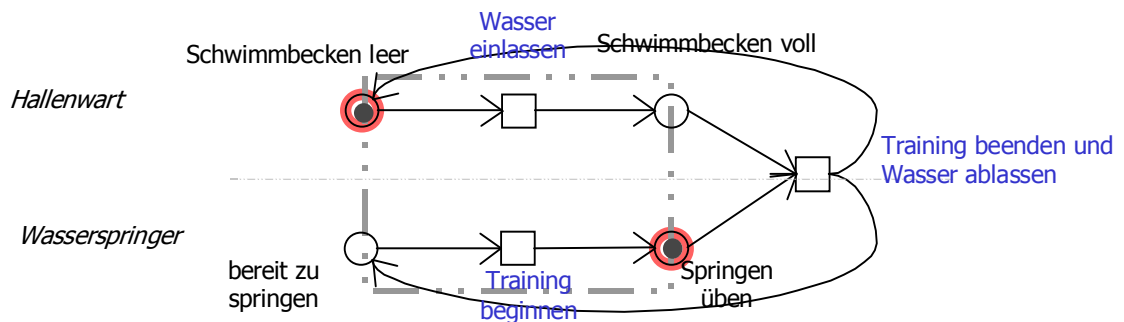


Abb. 1.9 Das System *Gefährdeter Wasserspringer* in einem erreichbaren Zustand

Im System ist der Zustand erreichbar, in dem *Schwimmb.-leer* und *Springen-ü.* markiert sind. D.h. das Schwimmbecken ist leer, aber der Sportler springt. Also die gewünschte Sicherheitseigenschaft bleibt nicht erhalten. Also ist auch dieser Verfeinerungsbegriff nicht ausreichend.

Es gibt in der Literatur sonst keine anderen geeigneten Transitionsverfeinerungen (Eine Übersicht über grundlegende Verfeinerungsbegriffe siehe Abschnitt 3.1). Wir brauchen also einen neuen Verfeinerungsbegriff.

Wie sollen wir die Transition *Wasser-einlassen-und-Training-beginnen* verfeinern, damit alle gewünschten Eigenschaften erhalten bleiben? Im Verfeinerungsnetz muss das Wasser zunächst eingelassen werden, erst danach kann das Training-beginnen zugelassen werden (Abb. 1.10). Nur so kann auch die gewünschte Sicherheitseigenschaft erhalten bleiben. Wir

haben deshalb einen neuen Verfeinerungsbegriff eingeführt. Unsere Idee dabei ist, die Kausalitätsforderungen mit der Blockbedingung zu verbinden. Für eine korrekte Transitionsverfeinerung müssen wir die Blockbedingung erfüllen und die Kausalitäten beachten.

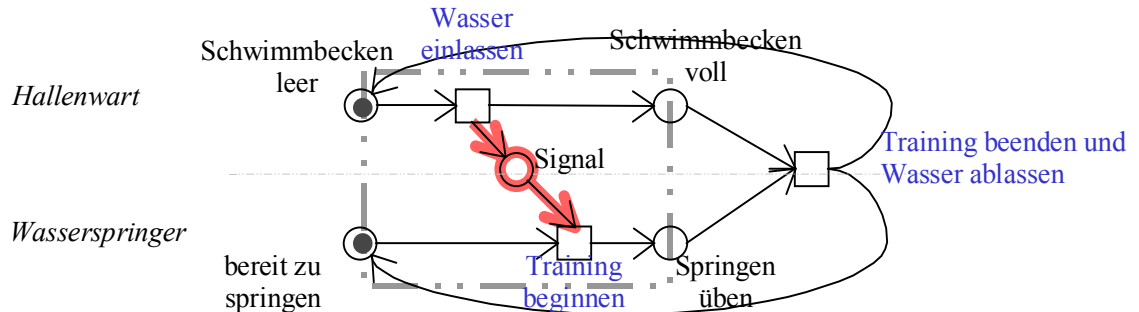


Abb. 1.10 Sicherer Wasserspringer

Agentensysteme

Im System *Wasserspringer* gibt es zwei Agenten: den *Wasserspringer* und den *Hallenwart* (Abb. 1.11).

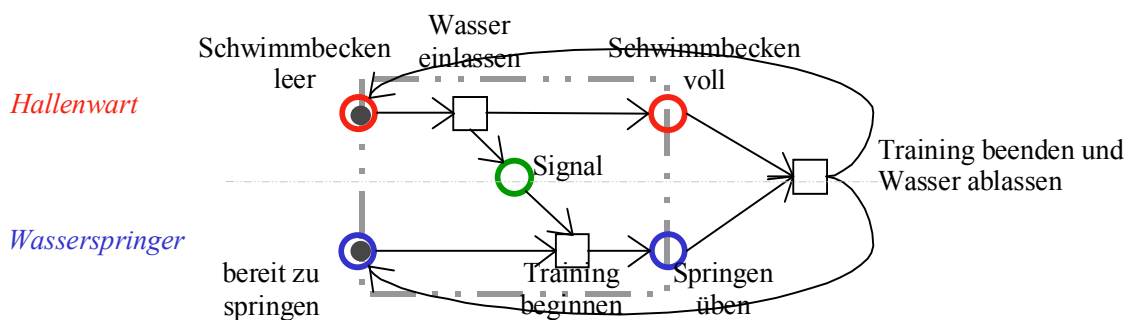


Abb. 1.11 Agentensystem *Sicherer Wasserspringer*

Der *Wasserspringer* kann im Zustand *bereit-zu-springen* sein, er kann auch gerade im Zustand *Springen-üben* sein. Also die Stellen *bereit-zu-springen* und *Springen-üben* gehören zu dem *Wasserspringer*. Analog, die Stellen *Schwimmbecken-leer* und *Schwimmbecken-voll* gehören zu dem *Hallenwart*. D.h. ein Agent wird durch Stellen und nicht durch Transitionen charakterisiert.

Die Stelle *Signal* steht für eine Nachricht. Wenn der *Hallenwart* das Wasser schon eingelassen hat, sagt er dem *Wasserspringer* Bescheid, indem er ihm diese Nachricht schickt. Die Stelle *Signal* nennen wir eine Kanalstelle. Eine Nachricht ist immer *von* einem Agenten *für* einen

anderen Agenten. Deshalb gilt: $\text{Sender} \neq \text{Empfänger}$. In diesem Beispiel also: *Hallenwart* \neq *Wasserspringer*.

Ein Agentensystem ist ein Petrinetz mit Agenten, Kanälen und der Bedingung, dass das System 1-beschränkt ist.

In Kapitel 4 werden wir die Begriffe Agenten, Agentensysteme, verteilte (lokale) Transition, unverteilte Transition einführen.

Kausalitäten im System

Wir haben schon gesehen, das System *Sicherer Wasserspringer* (Abb. 1.10) nach der Verfeinerung muss die folgende Bedingung erfüllen: In jedem verteilten Ablauf muss jedesmal zuerst das Wasser eingelassen werden, bevor das Training-beginnen zugelassen werden kann. Das nennen wir eine *Kausalität im System*. Im Folgenden werden wir genauer erklären, was dieser Begriff Kausalität im System in dieser Arbeit bedeutet. Wir fangen mit dem Begriff *Kausalität im Ablauf* an.

Wir sagen, in einem Ablauf gilt *Wasser-einlassen* bereitet

Training-beginnen vor (Notation: $\text{Wasser-einlassen} \blacktriangleleft \text{Training-beginnen}$), gdw. es in diesem Ablauf zu jedem Auftreten von *Training-beginnen* ein Auftreten von *Wasser-einlassen* gibt, mit: a) Dieses Auftreten von *Wasser-einlassen* liegt vor dem Auftreten von *Training-beginnen*; b) Zwischen den beiden Auftreten gibt es kein anderes Auftreten von *Training-beginnen*.

In Abb. 1.12 ist ein Ablauf des Systems *Sicherer Wasserspringer* dargestellt.

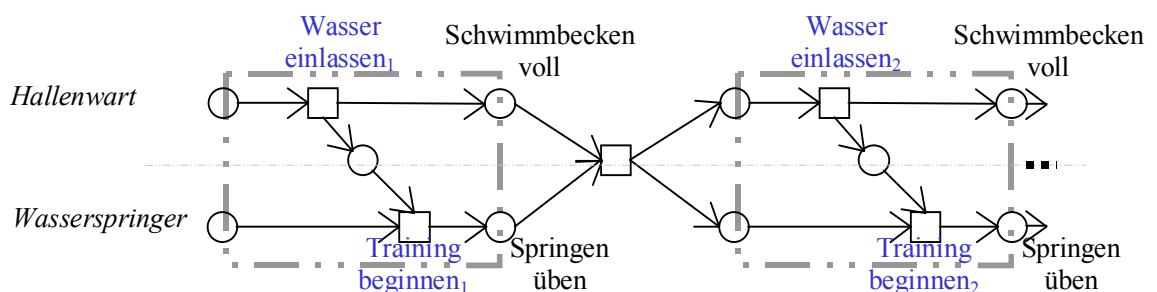


Abb. 1.12 Ablauf des Systems *Sicherer Wasserspringer* (Abb. 1.10)

In diesem Ablauf ist *Training-beginnen₂* ein Auftreten der Transition

Training-beginnen. Vor *Training-beginnen*₂ liegt ein Auftreten *Wasser-einlassen*₂ der Transition *Wasser-einlassen*. Zwischen beiden gibt es kein Auftreten der Transition *Training-beginnen*. Deshalb ist *Wasser-einlassen*₂ für *Training-beginnen*₂ das richtige Auftreten von *Wasser-einlassen*, das die oben genannten Bedingungen a) und b) erfüllt.

Dagegen ist *Wasser-einlassen*₁ zwar auch ein Auftreten von *Wasser-einlassen* und vor dem *Training-beginnen*₂, aber nicht das richtige Auftreten von *Wasser-einlassen* für *Training-beginnen*₂, weil zwischen *Wasser-einlassen*₁ und *Training-beginnen*₂ ein Auftreten *Training-beginnen*₁ der Transition *Training-beginnen* liegt.

In einem System gilt *Wasser-einlassen* \blacktriangleleft *Training-beginnen*, gdw. in jedem Ablauf *Wasser-einlassen* \blacktriangleleft *Training-beginnen* gilt.

In Kapitel 2 (Abschnitt 2.4) werden wir den Begriff Kausalität im System als eine wichtige Grundlage dieser Arbeit einführen und einige Ableseregeln angeben, damit wir direkt vom Netz die Gültigkeit der Kausalitäten im System ablesen können.

Verteilte Abläufe spielen in dieser Arbeit eine wichtige Rolle. Es gibt genau einen verteilten Ablauf (Abb. 1.7(b)) für das System *Gefährdeter Wasserspringer*. *Sequentielle Abläufe* gibt es unendlich viele, in denen Kausalitäten nicht direkt ablesbar sind, z.B. den folgenden sequentiellen Ablauf (Abb. 1.13).

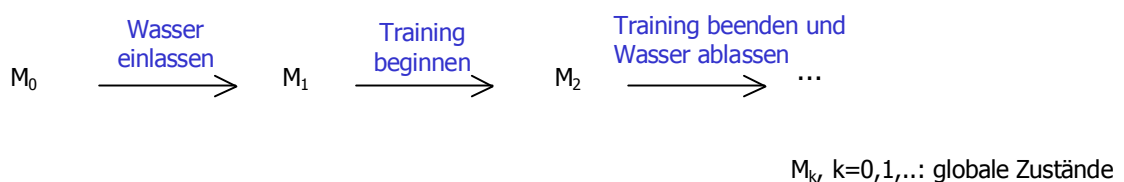


Abb. 1.13 Ein sequentieller Ablauf des Systems *Gefährdeter Wasserspringer*

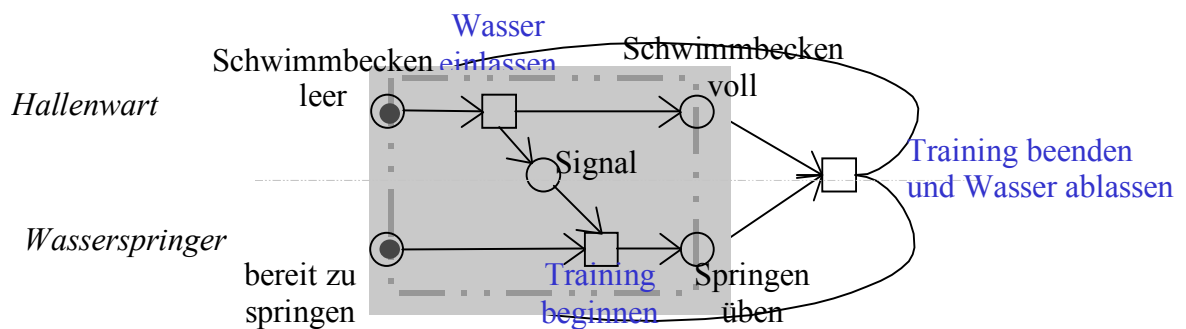
Sequentielle Abläufe verschleiern fehlende Kausalitäten im Netz, weil der direkte Zusammenhang zur Netzstruktur verloren geht.

In Kapitel 3 werden wir die Halbordnung erhaltende Transitionsverfeinerung betrachten. Zunächst werden wir die Definition für die Blockbedingung angeben und diskutieren, welche Eigenschaften die Blockbedingung garantieren kann und welche nicht. Danach werden wir Transitionsverfeinerung mit Kausalitäten angeben, bei der die Halbordnungen des zu verfeinernden Systems vollständig erhalten

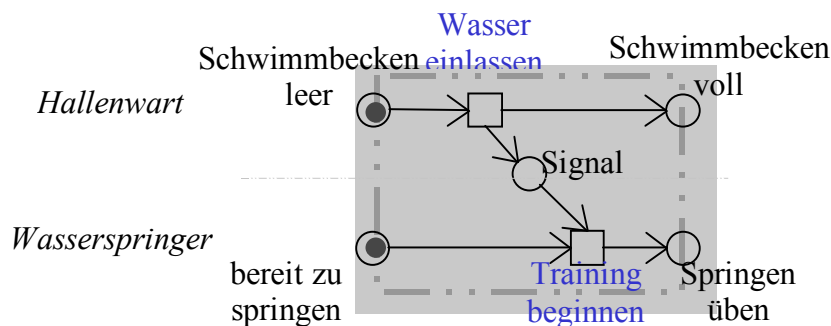
bleiben, und diskutieren, welche Eigenschaften diese Verfeinerung garantiert.

Kausalitäten im lokalen Teilsystem

In Abb. 1.14 (a) ist das System *Sicherer Wasserspringer* nochmals angegeben. Wir können den grau unterlegten Teil – das Ersatznetz für die zu verfeinernde Transition – isoliert betrachten. Dann ist dieser Teil auch ein System (siehe Abb. 1.14 (b)). Wir nennen es ein *lokales Teilsystem* (ein Teil des ganzen Systems).



(a) System



(b) Teilsystem

Abb. 1.14 Lokales Teilsystem

In einem lokalen Teilsystem kann man vom Netz schon ablesen, ob eine Kausalität in diesem Teilsystem gilt. Z.B. gilt hier

Wasser-einlassen \leftarrow *Training-beginnen* im lokalen System, weil es einen Weg von der Transition *Wasser-einlassen* zu der Transition *Training-beginnen* gibt.

In einem lokalen Teilsystem kann man durch Nachrichtenaustausch eine Kausalität in diesem Teilsystem realisieren. Z.B. wird hier die Kausalität

Wasser-einlassen \leftarrow *Training-beginnen* im lokalen System durch die Kanalstelle *Signal* realisiert.

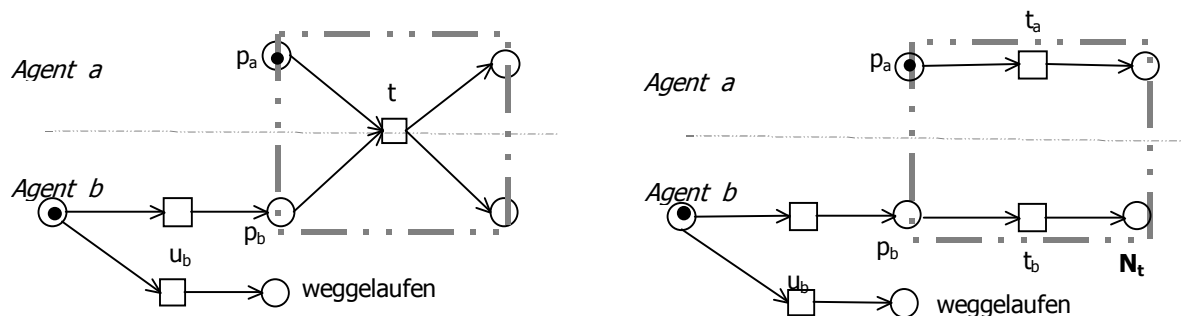
In Kapitel 5 werden wir die Transitionsverfeinerung einer unverteilter Transition in einem Agentensystem zu einem verteilten Netz betrachten, die wir *verteilende Verfeinerung* nennen. Dabei sollen *Synchronisationsbedingungen* erfüllt werden.

Synchronisationsbedingungen sind Kausalitäten im lokalen Teilsystem (Ersetzungsnetz) zwischen unterschiedlichen Agenten. Weiter diskutieren wir in diesem Kapitel, welche Eigenschaften die verteilende Verfeinerung garantiert.

Drei Grundtypen von Verfeinerungsnetzen und Umgebungen, die die Blockbedingung garantieren

Wie erwähnt, die Blockbedingung ist eine notwendige Voraussetzung für eine korrekte verteilende Verfeinerung. In Kapitel 6 beweisen wir *Kriterien*, mit deren Hilfe die Blockbedingung nachgewiesen werden kann.

Wann könnte die Blockbedingung verletzt werden? Dafür ein kleines Beispiel (Abb. 1.15).



(a) Vor der Verfeinerung

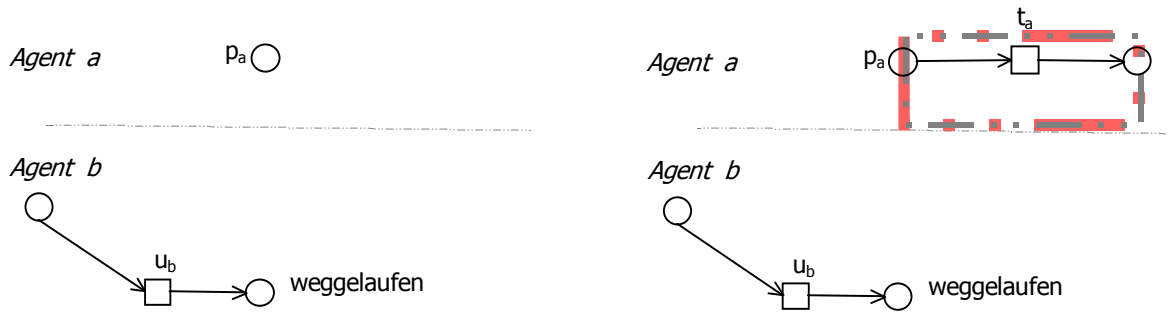
(b) Nach der Verfeinerung

Abb. 1.15 Gegenbeispiel zur Blockbedingung

Es gibt zwei Agenten a und b . Die unverteilte Transition t (siehe Abb. 1.15(a)) wird zu dem Netz N_t (siehe Abb. 1.15(b)) verfeinert. Vor der Verfeinerung kann die Transition t nur schalten, wenn Agent b die Stelle p_b erreicht hat. Wenn b statt der Aufgabe für t eine andere Aufgabe macht und weggelaufen ist, dann kann die Transition t nicht schalten (siehe Abb. 1.16(a)).

Nach der Verfeinerung weiß Agent a nicht von alleine, ob Agent b die Stelle p_b überhaupt erreicht. Wenn b die andere Aufgabe macht und keine

Nachricht an a schickt, dann kann es passieren, dass nur der Agent a die Aufgabe für t macht (siehe Abb. 1.16(b)). Dann ist die Blockbedingung verletzt.

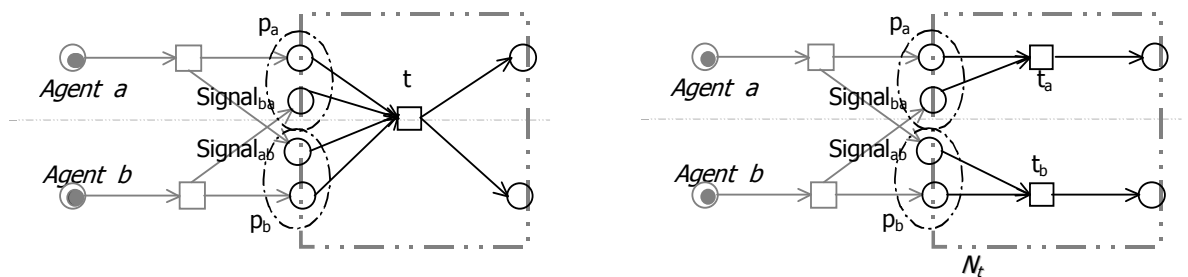


(a) Vor der Verfeinerung (b) Nach der Verfeinerung
 Abb. 1.16 Ein Ablauf, in dem die Blockbedingung verletzt ist

Wir haben drei Grundtypen von Verfeinerungsnetzen und Umgebungen gefunden, die die Blockbedingung garantieren.

Der erste Grundtyp: Abgesprochene Zusammenarbeit

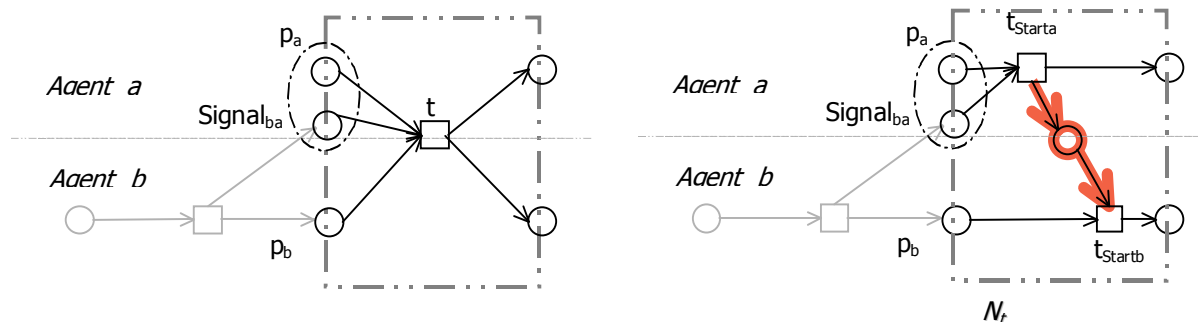
Dieser Grundtyp passt zu dem folgenden Fall: Beide Agenten haben vorher (vor der Transition t) gegenseitig Nachrichten geschickt. D.h. jeder Agent muss warten, bis er ein Signal von seinem Partner – dem anderen Agenten – bekommen hat. Erst danach kann er die Aufgabe für t machen. In diesem Fall brauchen die beiden im Ersetzungsnetz (also Verfeinerungsnetz) keine Kommunikation mehr, um die Blockbedingung zu erfüllen.



(a) Vor der Verfeinerung von t (b) Nach der Verfeinerung von t
 Abb. 1.17 Der erste Grundtyp: *Abgesprochene Zusammenarbeit*

Der zweite Grundtyp: erbetene Mithilfe

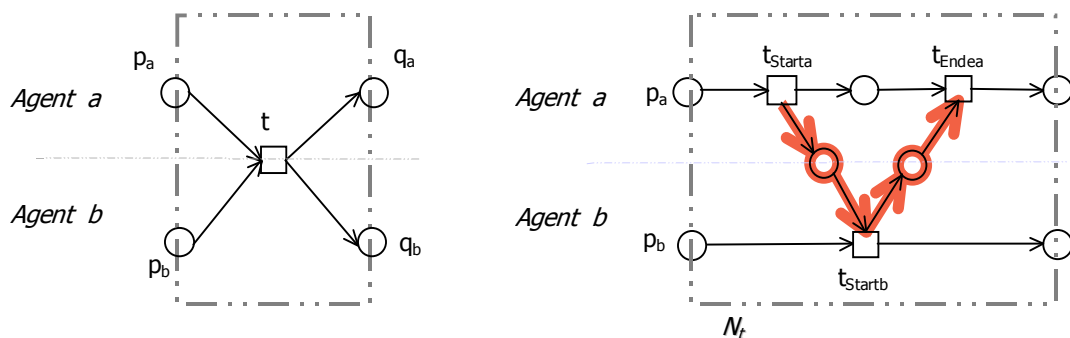
Dieser Grundtyp passt zu dem folgenden Fall: Einer (z.B. Agent b) der beiden Agenten hat vorher (vor der Transition t) eine Nachricht geschickt. D.h. der andere Agent (z.B. Agent a) muss warten, bis er ein Signal von seinem Partner bekommen hat. Erst danach kann er die Aufgabe für t machen. In diesem Fall muss dieser Agent (also Agent a) im Ersetzungsnetz eine Nachricht schicken, damit die Blockbedingung erfüllt ist.



(a) Vor der Verfeinerung von t (b) Nach der Verfeinerung von t
 Abb. 1.18 Der zweite Grundtyp: *Erbetene Mithilfe*

Der dritte Grundtyp: Erwartete Mithilfe

Dieser Grundtyp passt zu dem folgenden Fall: Keiner von den beiden Agenten hat vorher (vor der Transition t) eine Nachricht geschickt. In diesem Fall müssen beide Agenten im Ersetzungsnetz noch mehr kommunizieren, damit die Blockbedingung erfüllt ist.



(a) Vor der Verfeinerung von t (b) Nach der Verfeinerung von t
 Abb. 1.19 Der dritte Grundtyp: *Erwartete Mithilfe*

Also, beim ersten Grundtyp ist die Anforderung an die Umgebung sehr

streng. Dafür ist die Anforderung an das Ersetzungsnetz sehr schwach: Es benötigt gar keine Kausalität für die Blockbedingung.

Beim zweiten Grundtyp ist die Anforderung an die Umgebung etwas schwächer. Dann muss aber das Ersetzungsnetz eine Kausalität erfüllen, also die Anforderung an das Ersetzungsnetz ist etwas strenger.

Beim dritten Grundtyp ist die Anforderung an die Umgebung sehr schwach. Dann muss das Ersetzungsnetz noch mehr Kausalitäten erfüllen, also die Anforderung an das Ersetzungsnetz ist sehr streng.

Methode der verteilenden Verfeinerung: Entwurf und Verifikation an einem nicht trivialen Beispiel

In Kapitel 7 demonstrieren wir an verschiedenen Beispielen, wie sich verteilte Algorithmen mit Hilfe der verteilenden Verfeinerung entwerfen und verifizieren lassen. Hier werden wir kurz an einem dieser Beispiele unsere Grundideen vorstellen.

Beispiel: *Fahrradausleihe*

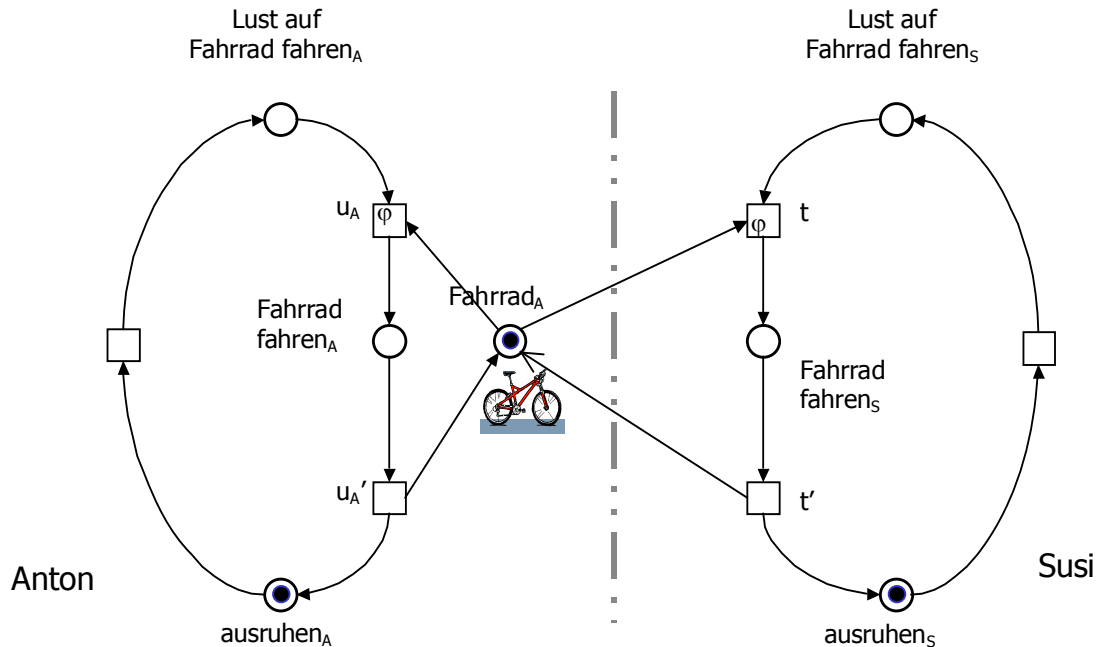


Abb. 1.20 *Fahrradausleihe*

Anton und *Susi* sind Geschwister. *Anton* hat ein Fahrrad (Stelle *Fahrrad_A*). Wenn er Lust hat (Stelle *Lust-auf-Fahrrad-fahren_A*), kann er Fahrrad

fahren (Stelle $Fahrrad-fahren_A$). Danach wird er sich ausruhen (Stelle $ausruhen_A$). *Susi* hat kein Fahrrad. Sie kann das Fahrrad aber von *Anton* ausleihen, wenn sie auch mal Fahrrad fahren möchte. Wenn *Anton* aber zufällig auch selber das Fahrrad braucht, kann er es benutzen. Aber *Anton* soll nicht immer wieder nur selbst das Fahrrad benutzen, er soll *Susi* irgendwann einmal das Fahrrad ausleihen (das nennt man *fairness*). Wenn *Susi* das Fahrrad benutzt, soll sie danach sofort das Fahrrad zurückgeben, weil das Fahrrad *Anton* gehört.

Das System hat folgende Eigenschaften:

Lebendigkeitseigenschaften:

$$Lust-auf-Fahrrad-fahren_A \triangleright Fahrrad-fahren_A \quad (L1)$$

$$Lust-auf-Fahrrad-fahren_S \triangleright Fahrrad-fahren_S \quad (L2)$$

D.h. jeder wird irgendwann das Fahrrad benutzen, wenn er Bedarf hat.

Sicherheitseigenschaft:

$$\square \neg (Fahrrad-fahren_A \wedge Fahrrad-fahren_S) \quad (S)$$

D.h. die beiden werden nie gleichzeitig das Fahrrad benutzen.

In Abb. 1.21 ist das Agentensystem dargestellt.

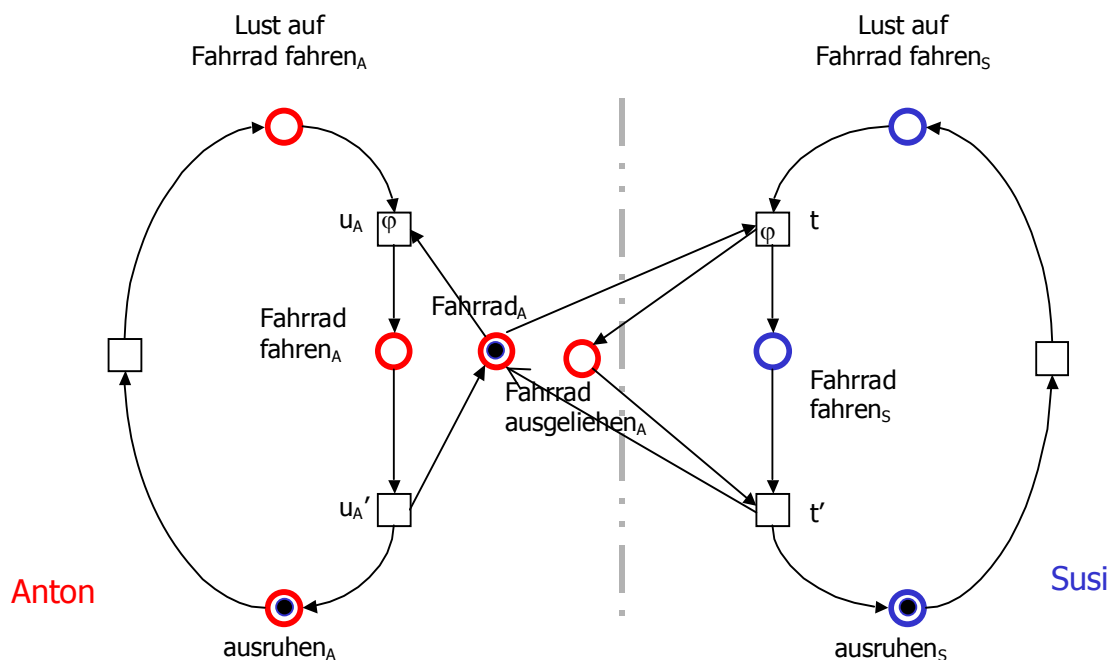


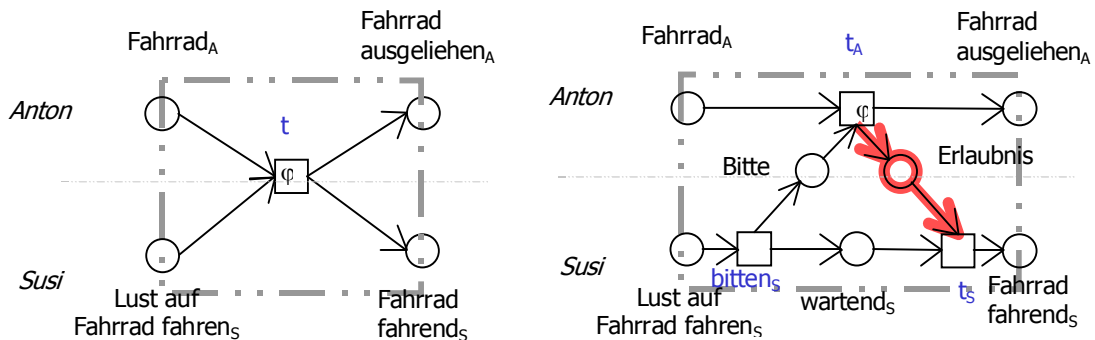
Abb. 1.21 Agentensystem *Fahrradausleihe*

Im System gibt es zwei Agenten: den Agenten *Anton* und die Agentin *Susi*. Die roten Stellen sind Stellen von *Anton*, die blauen Stellen sind Stellen von *Susi*. Die Stelle *Fahrrad* gehört zum Agenten *Anton*, weil das Fahrrad *Anton* gehört. Die Stelle *Fahrrad-ausgeliehen_A* modelliert, ob das Fahrrad ausgeliehen ist.

Die Transition u_A ist lokal von Anton, weil das Fahrrad von *Anton* ist. Analog ist die Transition u_A' lokal. Die Transition t ist von *Anton* und *Susi* zusammen auszuführen, sie ist also unverteilt. Analog ist die Transition t' auch unverteilt. Das System ist also unverteilt.

Wir möchten von diesem unverteilt System (Algorithmus) ausgehend einen verteilten Algorithmus entwerfen und seine Korrektheit verifizieren. Wir werden die unverteilt Transitionen t, t' nacheinander verteilen.

Wir verteilen die Transition t zu dem Netz N_t (Abb. 1.22).



(a) Transition t (b) Ersetzungsnetz N_t für t
Abb. 1.22 Verteilung von t

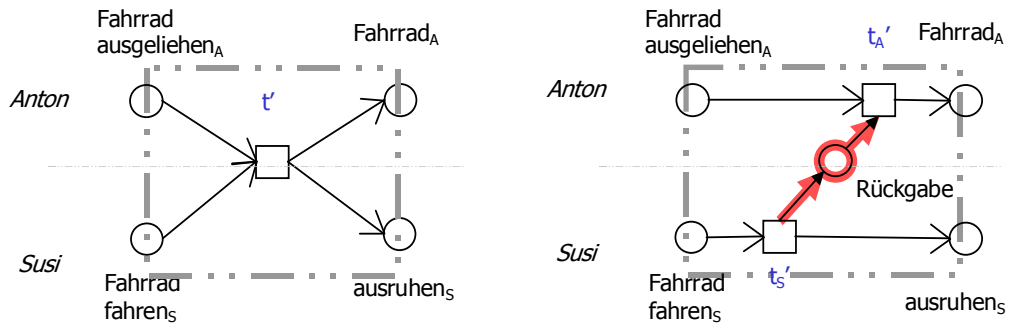
Wir können unseren dritten Grundtyp verwenden und zeigen, dass die Blockbedingung erfüllt ist.

Im lokalen Teilsystem (N_b, t^-) gilt die folgende Kausalität:

$$\bullet t \blacktriangleleft t \bullet.$$

Diese Kausalität sichert, dass die Halbordnungen des zu verfeinernden Systems vollständig erhalten bleiben. Dadurch wird das Erhaltenbleiben aller gewünschten Eigenschaften (S), (L1) und (L2) garantiert.

Wir verteilen die Transition t' zu dem Netz $N_{t'}$ (Abb. 1.23).



(a) Transition t' (b) Ersetzungsnetz $N_{t'}$ für t'
 Abb. 1.23 Verteilung von t'

Wir können unseren zweiten Grundtypen verwenden und zeigen, dass die Blockbedingung erfüllt ist.

Im lokalen Teilsystem (N_b, t^-) gilt die folgende Kausalität:

$$\text{Fahrrad-fahren}_S \blacktriangleleft \text{Fahrrad}_A.$$

Diese Kausalität zusammen mit der Blockbedingung garantiert das Erhaltenbleiben aller gewünschten Eigenschaften (S), (L1) und (L2).

In Abb. 1.24 ist das erhaltene System *Verteilte Fahrradausleihe* nach der Verteilung von t und t' angegeben.

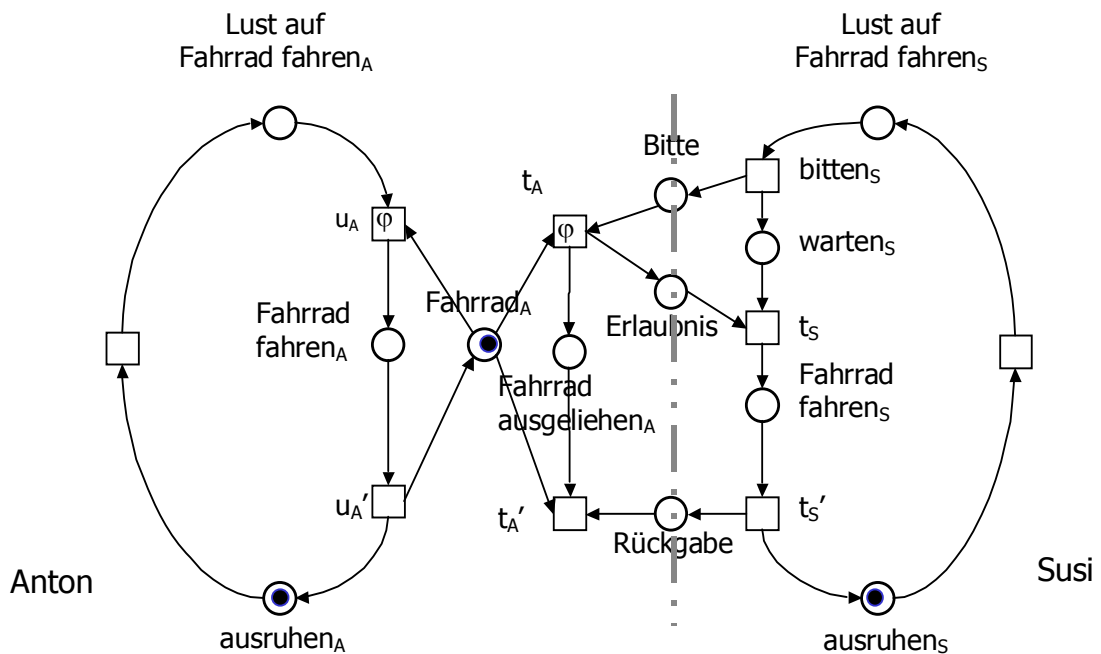


Abb. 1.24 Verteilte Fahrradausleihe

Das System *Verteilte Fahrradausleihe* ist schon verteilt, es ist unser Zielalgorithmus. In diesem System gelten alle gewünschten Eigenschaften.

Wir sehen, durch verteilende Verfeinerung kann man einen verteilten Algorithmus leicht entwerfen und verifizieren.

In Kapitel 7 geben wir genauer an, welche von uns bewiesenen Sätze die Korrektheit der beiden Verfeinerungsschritte garantieren.

2. Grundlagen

In diesem Kapitel werden wir die für diese Arbeit notwendigen Grundlagen vorstellen – Petrinetzmodell verteilter Algorithmen, ihre Halbordnungssemantik und ihre temporal-logischen Eigenschaften und Kausalitäten im Netz

2.1 Petrinetze

In diesem Abschnitt definieren wir elementare Petrinetze (S/T-Netze ohne Kantengewichtung) (vgl. [Peu01]). Der Leser, der mit dieser Petrinetzklasse vertraut ist (z.B. aus [Rei98]), kann diesen Abschnitt überspringen.

Definition 2.1.1(Petrinetz)

Ein *Petrinetz* (kurz: *Netz*) $N=(P,T,F)$ besteht aus zwei nicht-leeren, disjunkten Mengen P und T und der Flussrelation $F \subseteq (P \times T) \cup (T \times P)$, so dass für jedes $t \in T$ die Mengen $\{p \in P \mid (p,t) \in F\}$ und $\{p \in P \mid (t,p) \in F\}$ nicht leer und endlich sind. Die Elemente von P , T und F nennen wir Stellen, Transitionen bzw. Kanten des Netzes.

Für jedes $x \in P \cup T$ nennen wir die Menge $\bullet x = \{y \in P \cup T \mid (y,x) \in F\}$ den Vorbereich von x und die Menge $x \bullet = \{y \in P \cup T \mid (x,y) \in F\}$ den Nachbereich von x .

Für eine technisch angenehme Präsentation halten wir in dieser Arbeit eine universelle Stellenmenge \wp fest und nehmen an, dass alle in dieser Arbeit betrachteten Petrinetze nur Stellen aus dieser Menge haben, d.h. $P \subseteq \wp$.

Definition 2.1.2 (Markierung)

Eine Markierung von \wp ist eine Funktion $M: \wp \rightarrow \mathbb{N}$, die höchstens endlich vielen Stellen eine von Null verschiedene Zahl zuordnet. Eine Markierung des Netzes N ist eine Markierung, für die $M(p)=0$ für alle $p \in \wp \setminus P$ gilt.

Eine Markierung eines Petrinetzes beschreibt einen globalen Zustand des

Petrinetzes. Wir visualisieren eine Markierung durch Marken: Jede Stelle trägt die durch die Markierung angegebene Anzahl von Marken.

Jeder Transition t ordnen wir die beiden Markierungen t^- und t^+ zu, die folgendermaßen definiert sind:

$$t^-(p) = \begin{cases} 1, & p \in t \\ 0, & \text{sonst} \end{cases} \quad \text{und} \quad t^+(p) = \begin{cases} 1, & p \in t \\ 0, & \text{sonst} \end{cases}$$

Wir definieren die Addition von Markierungen punktweise, d.h. für zwei Markierungen M_1 und M_2 des Netzes N definieren wir die Summe $M_1 + M_2: \wp \rightarrow \mathbb{N}$ durch $(M_1 + M_2)(p) = M_1(p) + M_2(p)$ für alle $p \in \wp$. Analog vergleichen wir Markierungen punktweise und schreiben $M_1 \leq M_2$, falls $M_1(p) \leq M_2(p)$ für alle $p \in \wp$.

Eine Markierung wird durch das Schalten einer Transition verändert. Eine Transition t ist aktiviert in der Markierung M , wenn auf jeder Stelle im Vorbereich von t mindestens eine Marke liegt, d.h. $t^- \leq M$. Eine aktivierte Transition kann schalten. Das führt zur Markierung M' , die durch $M' + t^- = M + t^+$ gegeben ist (Es wird vermieden, eine partielle Subtraktion von Markierungen zu definieren). Den gerade beschriebenen Vorgang nennen wir *Schritt* und bezeichnen ihn durch $M \xrightarrow{t} M'$.

Eine Folge von Schritten $M_0 \xrightarrow{t_1} M_1 \dots \xrightarrow{t_n} M_n$ heißt ein *sequentieller Ablauf in N* .

Eine Markierung M' ist von der Markierung M aus *erreichbar*, wenn es eine endliche, ggf. leere Folge von Schritten $M_0 \xrightarrow{t_1} M_1 \dots \xrightarrow{t_n} M_n$ mit $M = M_0$ und $M' = M_n$ gibt.

Zwei verschiedene Transitionen t_1 und t_2 sind *konkurrent* zueinander, wenn ihre Vorbereiche nicht disjunkt sind. Eine Transition t_1 ist eine *aktivierte konkurrente* Transition zu einer anderen Transition t_2 in einer Markierung M , wenn sie konkurrent zu t_2 ist und beide in M aktiviert sind.

2.2 Verteilte Abläufe

Wir definieren eine Halbordnungssemantik für Petrinetze mit Hilfe verteilter Abläufe. Ein verteilter Ablauf wird durch ein Kausalnetz dargestellt. Ein Kausalnetz ist ein Petrinetz mit besonderen Eigenschaften. Es ist azyklisch, d.h. die Flußrelation enthält keinen Kreis. Außerdem ist ein Kausalnetz an Stellen unverzweigt, d.h. jede Stelle hat höchstens eine eingehende und eine ausgehende Kante. Die Stellen eines Kausalnetzes nennen wir Bedingungen und die Transitionen nennen wir Ereignisse.

Definition 2.2.1 (Kausalnetz)

Ein Netz $K = (B, E, \prec)$ ist ein Kausalnetz, wenn folgende Bedingungen gelten:

- i. K ist an den Stellen unverzweigt, d.h. $|\bullet b| \leq 1$ und $|b\bullet| \leq 1$ für jede Bedingung $b \in B$.
- ii. Die Menge ${}^\circ K = \{ b \in B \mid \bullet b = \emptyset \}$ ist nicht leer und endlich.
- iii. Die Relation \prec ist azyklisch. Dadurch ist die reflexive transitive Hülle von \prec eine Halbordnung, die wir mit \leq bezeichnen.
- iv. Jedes Element von $B \cup E$ hat nur endlich viele Vorgänger bzgl. der Halbordnung \leq .

Wir definieren nun einige Begriffe, um Aussagen über Kausalnetze zu treffen. Ein Schnitt ist eine maximale aber endliche Menge von paarweise unabhängigen Bedingungen bzgl. der Kausalordnung \leq eines Kausalnetzes. Zwei von einander unabhängige Ereignisse bzgl. \leq nennen wir auch zueinander nebenläufig.

Die transitive Hülle von \prec bezeichnen wir mit $<$.

Definition 2.2.2 (co-Menge, Schnitt, nebenläufige Ereignisse)

Sei $K = (B, E, \prec)$ ein Kausalnetz. Weiter seien $x, y \in B \cup E$. x co y dgw. $\neg(x < y)$ und $\neg(y < x)$.

Eine Menge $C \subseteq B$ ist eine *co-Menge* von K , gdw. für je zwei Bedingungen $b_1, b_2 \in C$ gilt b_1 co b_2 .

Eine maximale, endliche *co-Menge* C ist ein *Schnitt*.

Zwei verschiedene Ereignisse $e_1, e_2 \in E$ mit $e_1 \underline{co} e_2$ heißt zueinander *nebenläufig* (*parallel*).

Kausalität wird also auch benötigt, um über nebenläufige (parallele) und nicht nebenläufige Ereignisse Aussagen machen zu können.

Die Menge ${}^\circ K = \{ b \in B \mid \bullet b = \emptyset \}$ ist ein Schnitt. Diesen Schnitt nennen wir den Anfangsschnitt von K . Die Menge $K^\circ = \{ b \in B \mid b^\bullet = \emptyset \}$ ist eine *co-Menge*. Wir nennen K° das Ende des Kausalnetzes. Das Ende eines Kausalnetzes K ist genau dann ein Schnitt, wenn K endlich ist. Für jedes Ereignis $e \in E$ sind die Menge $\bullet e$ und e^\bullet *co-Mengen*. Für zwei *co-Mengen* C_1 und C_2 von K definieren wir: $C_1 < C_2$ genau dann, wenn $b_1 < b_2$ für alle $b_1 \in C_1$ und $b_2 \in C_2$ gilt.

Ein Schnitt C' eines Kausalnetzes K ist durch Schalten des Ereignisses $e \in E$ von einem Schnitt C erreichbar, wenn $\bullet e \subseteq C$ und $e^\bullet \cap C = \emptyset$ und $C' = C \setminus \bullet e \cup e^\bullet$ gilt. Wir schreiben $C \xrightarrow{e} C'$. Wenn ein Ereignis e mit $C \xrightarrow{e} C'$ existiert, schreiben wir $C \rightarrow C'$ und für die reflexive transitive Hülle von \rightarrow schreiben wir \rightarrow^* . Ein Schnitt C' ist von einem Schnitt C eines Kausalnetzes erreichbar, wenn $C \rightarrow^* C'$ gilt. (Erreichbarkeit wird benutzt, um über temporal-logische Eigenschaften reden zu können.)

Um einen verteilten Ablauf eines Systems zu modellieren, brauchen wir noch eine Funktion, die jedes Ereignis des Kausalnetzes als Schalten einer Transition des Systems identifiziert und jede Bedingung mit einer Stelle des Systems.

Definition 2.2.3 (*N-Beschriftung*)

Sei $N = (P, T, F)$ ein Netz und $K = (B, E, \triangleleft)$ ein Kausalnetz. Eine Funktion $r: B \cup E \rightarrow P \cup T$ ist eine *N-Beschriftung* von K , wenn die Funktion r Bedingungen auf Stellen und Ereignisse auf Transitionen abbildet.

Durch eine *N-Beschriftung* können wir jeder endlichen *co-Menge* C von K kanonisch eine Markierung von Σ zuordnen. Wir bezeichnen diese Markierung mit $r(C)$ und definieren $r(C): \wp \rightarrow \mathbb{N}$ mit $r(C)(p) = |\{ b \in C \mid r(b) = p \}|$.

Definition 2.2.4 (*Verteilter Ablauf in einem Netz*)

Seien $N=(P,T,F)$ ein Netz, $K = (B, E, \leq)$ ein Kausalnetz, $r: B \cup E \rightarrow P \cup T$ mit $r: B \rightarrow P, r: E \rightarrow T$ eine N -Beschriftung von K .

Das Paar $\rho=(K, r)$ heißt *ein verteilter Ablauf in N* , gdw. für jedes Ereignis $e \in E$ gilt: Wenn $r(e) = t$, dann ist $r(\bullet e) = t^-$ und $r(e \bullet) = t^+$.

Wir verlangen hier, dass jedes Ereignis e dem Schalten einer Transition t des Petrinetzes entspricht, d.h. e wird auf t abgebildet und der Vor- und Nachbereich von e auf den Vor- bzw. Nachbereich von t . Wir nennen e ein *Auftreten* der Transition t in diesem verteilten Ablauf in N .

Definition 2.2.5 (*Sequentialisierung eines verteilten Ablaufs im Netz N*)

Seien $N=(P,T,F)$ ein Netz, (K, r) ein verteilter Ablauf in N , wobei $K = (B,$

$E, \leq)$. Sei weiter $C_0 \xrightarrow{e_1} C_1 \xrightarrow{e_2} C_2 \dots$ ein sequentieller Ablauf in K mit $\{e_1, e_2, \dots\} = E$. Dann sagen wir, $r(C_0) \xrightarrow{r(e_1)} r(C_1) \xrightarrow{r(e_2)} r(C_2) \dots$ ist eine *Sequentialisierung* von (K, r) .

Bemerkung 2.2.6

Seien N ein Netz, (K, r) ein verteilter Ablauf in N .

Dann gilt: Jede Sequentialisierung von (K, r) ist ein sequentieller Ablauf in N .

Definition 2.2.7 (*Fairness im Ablauf*)

Seien $N=(P,T,F)$ ein Netz, $t \in T$.

- i. Sei $w = M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \dots$ ein unendlicher sequentieller Ablauf in N .
 w verletzt *Fairness* von t (w ist *unfair* zu t), falls t in unendlich vielen Markierungen M_i aktiviert ist und $t = t_i$ für nur endlich viele i .
- ii. Ein sequentieller Ablauf w in N *respektiert Fairness* von t (w ist *fair* zu t), falls w nicht unfair zu t ist.
- iii. Ein verteilter Ablauf (K, r) in N *respektiert Fairness* von t , falls jede Sequentialisierung von (K, r) *Fairness* von t respektiert.

Grafisch wird eine *faire* Transition mit „ φ “ beschriftet.

Definition 2.2.8 (System)

Seien $N=(P,T,F)$ ein Netz, m_0 eine Markierung von N , $T_f \subseteq T$.

$\Sigma=(N, m_0)$ mit T_f nennen wir ein *System*.

Wir nennen m_0 die *Anfangsmarkierung* von Σ und T_f die *Menge der fairen Transitionen* von Σ .

Eine Markierung M ist *erreichbar in Σ* , wenn M von m_0 aus erreichbar ist.

Für alle Systeme nehmen wir an, dass jede aktivierte Transition irgendwann schaltet oder eine zu der Transition in Konflikt stehende Transition schaltet (d.h. keine Transition von Σ ist in K° aktiviert). Das ist die sogenannte *progress-Annahme*.

Definition 2.2.9 (Verteilter Ablauf von einem System)

Sei $\Sigma=(N, m_0)$ mit T_f ein System, wobei $N=(P,T,F)$.

Ein verteilter Ablauf (K, r) in N ist ein *verteilter Ablauf* von Σ , gdw.

- i. $r(\circ K) = m_0$;
- ii. Für alle Transitionen t aus T und alle endlichen *co-Mengen* $C \subseteq K^\circ$ gilt: $t^- \not\subseteq r(C)$.
- iii. (K, r) respektiert die *Fairness* aller Transitionen aus T_f .

Mit Bedingung *i* fordern wir, dass der Anfangsschnitt auf die Anfangsmarkierung des Petrinetzes abgebildet wird. In Bedingung *ii* verlangen wir, dass die *progress-Annahme* erfüllt wird. Mit Bedingung *iii* fordern wir, dass die *Fairness-Annahme* jeder fairen Transition erfüllt wird.

Sei $\rho=(K, r)$ ein verteilter Ablauf von Σ und sei C eine *co-Menge* des Kausalnetzes, das ρ zugrunde liegt. Wenn wir alle Elemente des Kausalnetzes weglassen, die grösser als C sind, dann erhalten wir ein *Anfangsstück* von ρ .

Bemerkung: Eine Markierung M von Σ ist genau dann erreichbar, wenn es einen verteilten Ablauf von Σ mit einem Schnitt C gibt, so dass $r(C)=M$.

Definition 2.2.10 (*Progress im sequentiellen Ablauf*)

Seien $N=(P,T,F)$ ein Netz, $t \in T$.

Sei $w = M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \dots$ ein sequentieller Ablauf in N .

- i. w verletzt Progress von t , falls t in einer Markierung M_i aktiviert ist, und es keine Markierung M_j gibt, mit: $j > i$ und $t_j \in (\bullet t) \bullet$.
- ii. w respektiert Progress von t , falls w Progress von t nicht verletzt.

Definition 2.2.11 (*Sequentieller Ablauf von einem System*)

Sei $\Sigma=(N, m_0)$ mit T_f ein System, wobei $N=(P,T,F)$.

Seien $N=(P,T,F)$ ein Netz, $t \in T$.

Ein sequentieller Ablauf $w = M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \dots$ in N ist ein sequentieller Ablauf von Σ , gdw.

- i. $M_0 = m_0$;
- ii. w respektiert Progress aller Transitionen aus T ;
- iii. w respektiert Fairness aller Transitionen aus T_f .

Eine Stelle p eines Systems Σ ist *1-beschränkt*, wenn in jeder erreichbaren Markierung M auf dieser Stelle nicht mehr als eine Marke liegt, d.h. $M(p) \leq 1$.

Definition 2.2.12 (*1-beschränktes System*)

Ein System ist *1-beschränkt*, wenn auf keiner Stelle jemals mehr als eine Marke liegt, d.h. für jede erreichbare Markierung M und für jede Stelle $p \in \wp$ gilt $M(p) \leq 1$.

Wie in der Einleitung schon erwähnt, verwenden wir in dieser Arbeit verteilte Abläufe. Ein sequentieller Ablauf zeigt nicht Kausalitäten. Aus allen sequentiellen Abläufen lassen sich nicht alle Kausalitäten feststellen. Wir werden sehen, Kausalitäten zwischen Aktionen unterschiedlicher Agenten in einem verteilten System sind aber entscheidend für die Korrektheit des Systems. Also, in dieser Arbeit sind nur verteilte Abläufe geeignet.

2.3 Eigenschaften von verteilten Algorithmen

Zur Beschreibung von Eigenschaften verteilter Algorithmen benutzen wir Formeln einer temporalen Logik (vgl. [Kin95], [Peu01]), die auf die vorher definierten Systeme abgestimmt ist. Wir geben hier nur die Syntax und die Semantik dieser Formeln an.

Als Beispiel für ein System benutzen wir ein *producer/consumer* Modell (z.B. in [Rei98]). In diesem System (Abb. 2.3.1) kann der *producer* ein Produkt produzieren (Transition *produce*), das wird dann dem *consumer* angeboten (Transition *del-rem*) und von ihm konsumiert (Transition *consume*). Dieses System hat z.B. folgende Eigenschaft: Immer steht der *producer* entweder im Zustand *ready to produce* oder im Zustand *ready to deliver*.

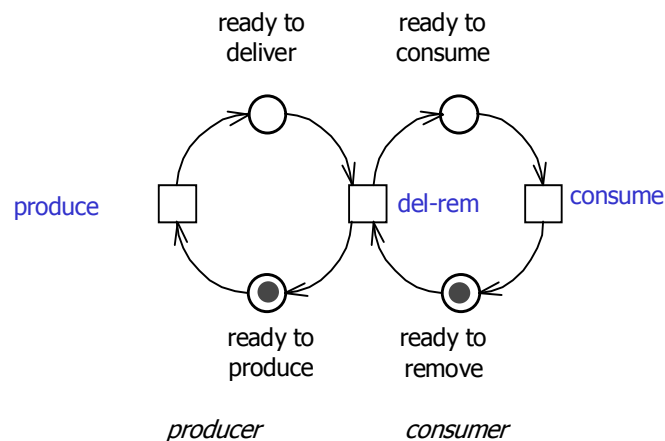


Abb. 2.3.1 System *producer/consumer*

Terme

Mit Zustandaussagen formalisieren wir Aussagen über Zustände eines Systems. Die Grundbausteine der Zustandaussagen sind Terme.

Angenommen, *ready to produce* ist ein Stellenname eines Systems. Dann ist

$\#\{ \textit{ready to produce} \}$ ein Term und $\#\{ \textit{ready to produce} \} \geq 0$ eine mit Hilfe dieses Terms formulierte Zustandaussage, die ausdrückt, dass die Anzahl der Marken auf der Stelle *ready to produce* grösser oder gleich 0 ist. Sei m eine Variable der Sorte \mathbb{N} , wobei \mathbb{N} die Menge der natürlichen Zahlen bezeichnet. Dann ist $\#\{ \textit{ready to produce} \} \geq m$ eine Zustandaussage, die ausdrückt, dass die Anzahl der Marken auf der Stelle

ready to produce grösser als m ist. m ist dabei ebenfalls ein Term.

Sei P eine Menge und V eine Menge von Variablen. Dann wird die Menge der Terme $\mathbf{ZT}(P, V)$ über P und V induktiv wie folgt definiert:

1. Für $n \in \mathbb{N}$ gilt $n \in \mathbf{ZT}(P, V)$.
2. Für $v \in V$ gilt $v \in \mathbf{ZT}(P, V)$.
3. Für jede endliche Menge $Z \subseteq P$ gilt $\#Z \in \mathbf{ZT}(P, V)$.
4. Für $u, u' \in \mathbf{ZT}(P, V)$ gilt auch $u+u' \in \mathbf{ZT}(P, V)$.

Eine Belegung für eine Variablenmenge V ist eine Funktion $\beta: V \rightarrow \mathbb{N}$, die jeder Variablen eine natürliche Zahl zuordnet. Die Terme werden in einer Markierung M unter einer Belegung β durch $\beta_M: \mathbf{ZT}(P, V) \rightarrow \mathbb{N}$ interpretiert, wobei β_M induktiv definiert ist durch:

1. $\beta_M(n) = n$ für $n \in \mathbb{N}$.
2. $\beta_M(v) = \beta(v)$ für $v \in V$.
3. $\beta_M(\#Z) = \sum_{s \in Z} M(s)$ für $Z \subseteq P$.
4. $\beta_M(u+u') = \beta_M(u) + \beta_M(u')$ für $u, u' \in \mathbf{ZT}(P, V)$.

Zustandsaussagen

Für zwei Terme $u, u' \in \mathbf{ZT}(P, V)$ ist $u \geq u'$ eine Zustandsaussage. Aus diesen Zustandsaussagen bilden wir mit den üblichen logischen Operatoren und Quantoren die Menge der Zustandsaussagen.

Definition 2.3.1 (Zustandsaussage)

Sei P eine Menge und V eine Menge von Variablen. Wir definieren die Menge $\mathbf{ZA}(P, V)$ der Zustandsaussagen über P und V wie folgt:

1. Wenn $u, u' \in \mathbf{ZT}(P, V)$, dann ist $u \geq u' \in \mathbf{ZA}(P, V)$.
2. Wenn $\varphi, \varphi' \in \mathbf{ZA}(P, V)$, dann ist $\varphi \wedge \varphi' \in \mathbf{ZA}(P, V)$.
3. Wenn $\varphi \in \mathbf{ZA}(P, V)$, dann ist $\neg\varphi \in \mathbf{ZA}(P, V)$.
4. Wenn $\varphi \in \mathbf{ZA}(P, V)$ und $v \in V$, dann ist $(\exists v \varphi) \in \mathbf{ZA}(P, V)$.

Wir interpretieren eine Zustandsaussage in einer Markierung M mit einer Belegung β der Variablen.

Definition 2.3.2 (Gültigkeit einer Zustandsaussage)

Eine Zustandsaussage φ ist genau dann in einer Markierung M und einer Belegung β gültig (Notation: $M, \beta \models \varphi$), wenn

1. $\varphi = (u \geq u')$ mit $u, u' \in \mathbf{ZT}(P, V)$, und $\beta_M(u) \geq \beta_M(u')$ oder
2. $\varphi = \varphi_1 \wedge \varphi_2$ und $M, \beta \models \varphi_1$ und $M, \beta \models \varphi_2$ oder
3. $\varphi = \neg \varphi_1$ und es gilt $M, \beta \models \varphi_1$ nicht oder
4. $\varphi = \exists v \varphi_1$ und es gibt eine Belegung β' mit $\beta(v') = \beta'(v')$ für alle $v' \neq v$, so dass $M, \beta' \models \varphi_1$.

Eine Zustandsaussage φ ist gültig in einer Markierung M (Notation: $M \models \varphi$), wenn φ in M unter jeder Belegung β gültig ist.

Temporale Aussagen

Zustandsaussagen sind die Grundbausteine von temporalen Aussagen. Temporale Aussagen interpretieren wir über verteilten Abläufen. Wir definieren zunächst nur den temporalen Operator \square (*always*) und bauen alle temporalen Aussagen induktiv aus \square , \wedge und \neg auf. Wir führen dann andere temporale Operatoren als Abkürzungen von Kombinationen aus \square , \wedge und \neg ein. Die Gültigkeit einer temporalen Aussage wird wieder induktiv über den Aufbau der Formeln definiert.

Definition 2.3.3 (Gültigkeit einer temporalen Aussage)

Sei $\rho = (K, r)$ ein verteilter Ablauf eines Systems $\Sigma = (N, m_0)$ und seien β eine Belegung der Variablenmenge V und C ein Schnitt aus ρ .

1. Wenn $\varphi \in \mathbf{ZA}(P, V)$, dann gilt $\rho, C, \beta \models \varphi$ genau dann, wenn $r(C), \beta \models \varphi$.
2. Es gilt genau dann $\rho, C, \beta \models \square \varphi$, wenn für jeden Schnitt C' mit $C \rightarrow^* C'$ aus ρ gilt $r(C'), \beta \models \varphi$.
3. Es gilt genau dann $\rho, C, \beta \models \varphi \wedge \varphi'$, wenn $\rho, C, \beta \models \varphi$ und $\rho, C, \beta \models \varphi'$.
4. Es gilt genau dann $\rho, C, \beta \models \neg \varphi$, wenn nicht $\rho, C, \beta \models \varphi$ gilt.
5. Es gilt genau dann $\rho, C, \beta \models \exists v \varphi$, wenn es eine Belegung β' gibt, mit $\beta(v') = \beta'(v')$ für alle $v' \neq v$, so dass $\rho, C, \beta' \models \varphi$.

Eine temporale Aussage φ (z.B. $\varphi = \Box p+q=1$) gilt im Ablauf ρ , wenn für jede Belegung β gilt: $\rho, \circ K, \beta \models \varphi$.

Die Aussage gilt im System Σ (Notation: $\Sigma \models \varphi$), wenn sie in jedem Ablauf gilt.

Die logischen Operatoren $\vee, \rightarrow, \leftrightarrow$ sind die üblichen Abkürzungen.

Für eine Zustandsaussage φ steht $\diamond \varphi$ abkürzend für $\neg \Box \neg \varphi$.

Für zwei Zustandsaussagen φ und φ' steht $\varphi \triangleright \varphi'$ abkürzend für $\Box (\varphi \rightarrow \diamond \varphi')$. Wir nennen dies eine *leadsto-Aussage im verteilten Ablauf*. Sie bedeutet, dass in einem verteilten Ablauf auf einen Zustand, der φ erfüllt, ein Zustand folgt, der φ' erfüllt. In einem System gilt $\varphi \triangleright \varphi'$, wenn in jedem Ablauf $\varphi \triangleright \varphi'$ gilt.

Sei φ eine Zustandsaussage. Wenn $\Box \varphi$ in einem System gilt, dann heißt $\Box \varphi$ eine Invariante des Systems. In diesem Fall gilt φ in jeder erreichbaren Markierung des Systems. Besonders einfach zu handhabende Invarianten sind Stelleninvarianten. Einer Stelleninvariante entsprechend, gibt es eine Menge von Stellen, für die sich die Summe der Marken, die auf diesen Stelle liegen, beim Schalten nicht verändert. Z.B. im *producer/consumer*-System (vgl. Abb 2.3.1) gilt, der Produzent ist entweder *ready to produce* oder *ready to deliver*. Wir beschreiben dies durch die folgende Aussage:

$$\Box \quad \text{ready to produce} + \text{ready to deliver} = 1.$$

In dieser Arbeit verwenden wir auch folgende Notation: Wenn p ein Stellenname ist, dann drücken wir mit p aus, dass mindestens eine Marke auf der Stelle p liegt. Wenn $Q = \{q_1, q_2, \dots, q_n\}$ eine Menge von Stellen ist, schreiben wir auch Q für $q_1 \wedge q_2 \wedge \dots \wedge q_n$.

Wir haben hier temporale Aussagen über verteilten Abläufen eines Systems interpretiert. Ähnlich können wir eine temporale Aussage auch über sequentiellen Abläufen eines Systems interpretieren.

Eine weitere temporale Eigenschaft ist *leadsto-Aussage im sequentiellen Ablauf*, die auch für sequentielle Abläufe definiert wird. Eine *leadsto-Aussage* schreiben wir $\varphi \mapsto \varphi'$, wobei φ, φ' Zustandsaussagen eines Systems sind. $\varphi \mapsto \varphi'$ gilt in einem

sequentiellen Ablauf $w = M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \dots$ eines Systems, wenn es für jede Markierung M_i , in der φ gilt, eine Markierung M_j mit $j \geq i$ gibt, in der φ' gilt. In einem System gilt $\varphi \mapsto \varphi'$, wenn in jedem Ablauf $\varphi \mapsto \varphi'$ gilt.

Im Folgenden wird ein Lemma über den Zusammenhang zwischen der *leadsto*-Aussage im verteilten Ablauf und der *leadsto*-Aussage im sequenziellen Ablauf angegeben (vgl. [Rei98] Seite 188).

Lemma 2.3.4 (*Zusammenhang zwischen \mapsto und \triangleright*)

Sei $\Sigma = ((P, T, F), m_0)$ ein System. Seien p, q zwei Zustandsaussagen von Σ .

- i. Wenn $\Sigma \models p \mapsto q$, dann gilt: $\Sigma \models p \triangleright q$.
- ii. Sei $Q \subseteq P$. Sei weiter $q = \bigvee Q^1$.

Dann gilt: Wenn $\Sigma \models p \triangleright q$, so $\Sigma \models p \mapsto q$.

Es folgt ein Satz über *Fairness* (vgl. [Rei98] Seite 176).

Definition 2.3.5 (*Konflikt-reduziert*)

Sei Σ ein System, $t \in T$.

t heißt *konflikt-reduziert*, gdw. es höchstens eine Stelle $p \in \bullet t$ mit $p \bullet \not\supseteq \{ t \}$ gibt.

Die Stelle p nennen wir die *Konflikt-Stelle* von t .

Satz 2.3.6 (*Fairness*)

Sei $\Sigma = (N, m_0)$ mit T_f ein System.

Sei $t \in T_f$ konflikt-reduziert mit Konflikt-Stelle p .

Für $Q := \bullet t \setminus \{ p \}$ gelte: $\Sigma \models Q \mapsto p$.

Dann gilt: $\Sigma \models Q \mapsto t \bullet$.

Von diesem Satz erhalten wir folgende:

Folgerung 2.3.7 (von Satz 2.3.6)

Sei $\Sigma = (N, m_0)$ mit T_f ein System.

Sei $t \in T_f$ konflikt-reduziert mit Konflikt-Stelle p .

¹ Sei $Q = \{q_1, q_2, \dots, q_n\} \subseteq P$ eine Teilmenge von P . $\bigvee Q$ steht für: $q_1 \vee q_2 \vee \dots \vee q_n$.

Für $Q := \bullet t \setminus \{ p \}$ gelte: $\Sigma \models Q \triangleright p$.

Dann gilt: $\Sigma \models Q \triangleright t \bullet$.

Beweis:

Aus Lemma 2.3.4 ii folgt: $\Sigma \models Q \mapsto p$.

Aus Satz 2.3.6 folgt: $\Sigma \models Q \mapsto t \bullet$.

Aus Lemma 2.3.4 i folgt: $\Sigma \models Q \triangleright t \bullet$. □

Eine Transition t heißt *abgeschwächt konflikt-reduziert*, wenn jede Stelle aus $\bullet t$ bis auf eine Stelle keine aktivierte Konflikt-Transition hat. Auch in diesem Fall können wir *Fairness* ablesen.

Definition 2.3.8 (*abgeschwächt konflikt-reduziert*)

Sei Σ ein System, $t \in T$. Sei $p \in \bullet t$, $Q := \bullet t \setminus \{ p \}$.

t heißt *abgeschwächt konflikt-reduziert*, gdw.

für jedes $q \in Q$ und jedes $t' \in q \setminus \{ t \}$ gilt $Q \rightarrow \neg \bullet t'$.

Die Stelle p nennen wir die *Konflikt-Stelle* von t .

Folgerung 2.3.9 (von Satz 2.3.6)

Sei $\Sigma = (N, m_0)$ mit T_f ein System.

Sei $t \in T_f$ abgeschwächt konflikt-reduziert mit Konflikt-Stelle p .

Für $Q := \bullet t \setminus \{ p \}$ gelte: $\Sigma \models Q \mapsto p$.

Dann gilt: $\Sigma \models Q \mapsto t \bullet$.

Für den Beweis dieser Folgerung brauchen wir folgende Bemerkung, die wir direkt von Def. 2.3.8 erhalten:

Bemerkung 2.3.10

Sei $\Sigma = (N, m_0)$ mit T_f ein System. Sei $t \in T_f$ abgeschwächt konflikt-reduziert mit Konflikt-Stelle p , $Q := \bullet t \setminus \{ p \}$.

Sei weiter $w = a_0 \xrightarrow{t_1} a_1 \xrightarrow{t_2} \dots$ ein sequentieller Ablauf von Σ .

Dann gilt: Für jede Markierung a_k von w , in der Q gilt, gilt $t_{k+1} = t$ oder

$a_{k+1} \models Q$.

Beweis von Folgerung 2.3.9:

Wegen Bemerkung 2.3.10 können wir hier mit derselben Beweisidee wie beim Beweis von Satz 2.3.6 (vgl. [Rei98] Seite 176-177) beweisen.

Weiter erhalten wir folgende *Fairness*-Beweis-Regel über *leadsto*-Eigenschaften im verteilten Ablauf:

Folgerung 2.3.11 (von Satz 2.3.6)

Sei $\Sigma = (N, m_0)$ mit T_f ein System.

Sei $t \in T_f$ abgeschwächt konflikt-reduziert mit Konflikt-Stelle p .

Für $Q := \bullet t \setminus \{ p \}$ gelte: $\Sigma \models Q \triangleright p$.

Dann gilt: $\Sigma \models Q \triangleright t \bullet$.

Beweis: Beweisidee ist dieselbe wie bei Folgerung 2.3.7.

2.4 Kausalitäten im System

Bei der verteilenden Verfeinerung eines Systems sollen temporal-logische Eigenschaften erhalten bleiben, also Lebendigkeitsaussagen und Sicherheitsaussagen. Es ist bekannt, dass z.B. die Gültigkeit von Sicherheitsaussagen durch Kausalitäten zwischen Ereignissen in den Abläufen erreicht werden, z.B. vor jedem Start eines Flugzeugs wird ein Sicherheitscheck durchgeführt.

Diese Arbeit zeigt, dass es ausreicht, derartige Kausalitäten lokal in den Verfeinerungsnetzen zu erhalten, damit sie weiter global im ganzen verfeinerten Netz gelten. Dadurch ist zu erreichen, dass auch gewünschte Sicherheitsaussagen weiter gelten.

Die Aufgabe des folgenden Abschnittes ist es einen geeigneten Kausalitätsbegriff für Systeme zu definieren. Grundlage dieser Definition ist die Kausalität in Abläufen. Weiter zeigt dieser Abschnitt, wie für 1-beschränkte Systeme unmittelbar aus der Struktur des Netzes des Systems die Gültigkeit einer Kausalitätsaussage für dieses System abgelesen werden kann.

2.4.1 Definition von Kausalitäten im System

Im Folgenden definieren wir Kausalitäten im System. Zunächst sehen wir uns ein paar Beispiele an. Das in Abb. 2.4.1 dargestellte System Σ_a beschreibt: Die Person a wird immer zunächst Geld verdienen, bevor sie Geld ausgibt (wir nehmen an, man gibt jedes mal so viel Geld aus, wie man verdient).

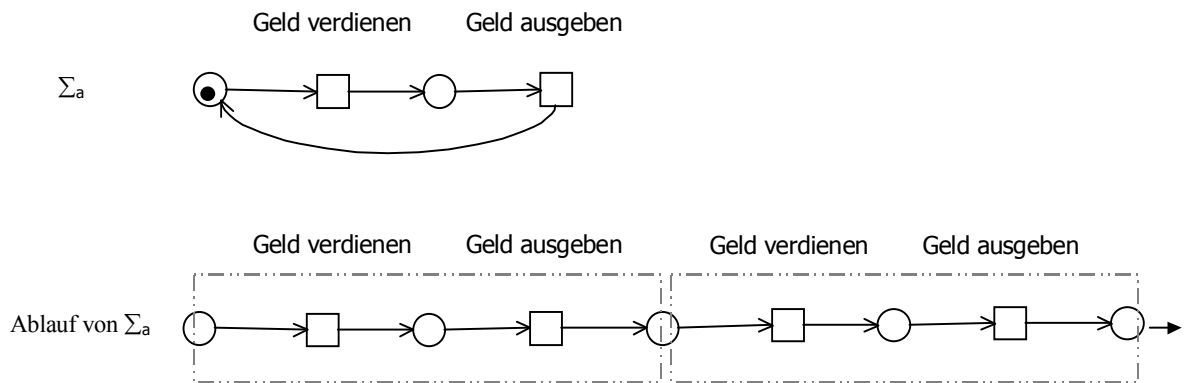


Abb. 2.4.1 System Σ_a und Ablauf von Σ_a

Im Gegensatz zu a verdient die Person b (Abb. 2.4.2) nur am Anfang einmal Geld, gibt danach aber immer wieder Geld aus. Das ist sicherlich nur möglich, wenn b noch eine andere Geldquelle hat, die nicht im Modell angegeben ist.

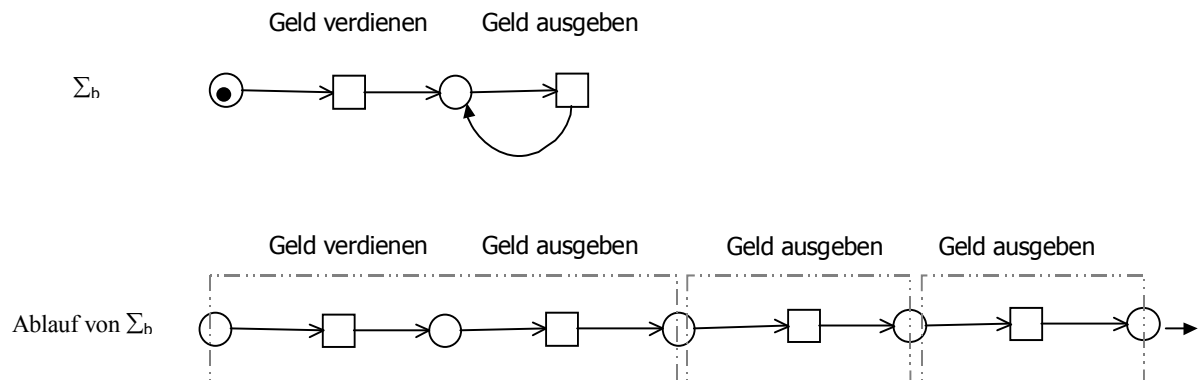


Abb. 2.4.2 System Σ_b und Ablauf von Σ_b

Wir sagen, G -verdienen bereitet im System Σ_a G -ausgeben vor, nicht aber in Σ_b und schreiben $\Sigma_a \models G\text{-verdienen} \blacktriangleleft G\text{-ausgeben}$ und $\Sigma_b \not\models G\text{-verdienen} \blacktriangleleft G\text{-ausgeben}$.

Angenommen, Σ ist ein System, t, t' sind zwei Transitionen von Σ . Wir sagen, $t \blacktriangleleft t'$ gilt in einem Ablauf K von Σ , wenn für jedes Auftreten eines

Ereignisses e' der Transition t' in K gibt es ein Auftreten e von t , so dass e vor e' liegt und e, e' immer in Paaren auftreten. In Σ_a tritt die Transition G -verdienen immer auch wieder auf, bevor die Transition G -ausgeben auftritt. Deshalb gilt

$\Sigma_a \models G\text{-verdienen} \blacktriangleleft G\text{-ausgeben}$. In Σ_b tritt dagegen die Transition G -verdienen z.B. schon beim zweiten Auftreten von G -ausgeben nicht erneut auf, deshalb gilt $\Sigma_b \not\models G\text{-verdienen} \blacktriangleleft G\text{-ausgeben}$.

In Abb. 2.4.3 ist ein weiteres System Σ_3 dargestellt. Σ_3 hat zwei Abläufe. In beiden Abläufen kommt die Transition a jeweils einmal vor. Aber nur in einem der beiden Abläufe tritt die Transition b vor dem Auftreten von a auf. Deshalb gilt nicht b bereitet a vor, also $\Sigma_3 \not\models b \blacktriangleleft a$.

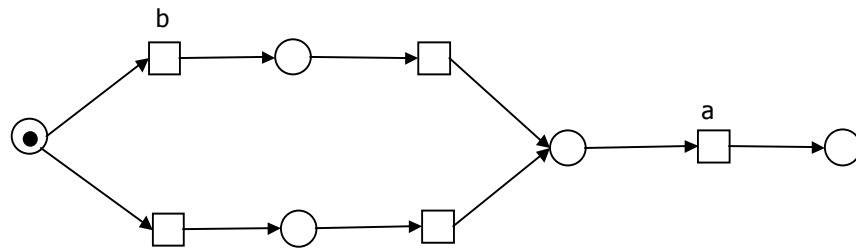


Abb. 2.4.3 System Σ_3

Definition 2.4.1 (Kausalrelation 'bereitet vor' in einem Ablauf zwischen Stellen bzw. Transitionen)

Sei $\Sigma = ((P, T, F), m_0)$ ein System, (K, r) ein Ablauf von Σ .

Weiter seien $x, y \in P \cup T$.

$x \blacktriangleleft y$ gilt in K (Notation: $K \models x \blacktriangleleft y$), gdw. zu jedem $s \in K$ mit $r(s) = y$ gibt es ein $s' \in K$ mit $r(s') = x$ und $s' < s$ und für jedes $s'' \in K$ mit $s' < s'' < s$ gilt $(s'') \neq y$.

Definition 2.4.2 (Kausalrelation 'bereitet vor' im System zwischen Stellen bzw. Transitionen)

Sei $\Sigma = ((P, T, F), m_0)$ ein System, seien $x, y \in P \cup T$.

$x \blacktriangleleft y$ gilt in Σ (Notation: $\Sigma \models x \blacktriangleleft y$), gdw. in jedem Ablauf $x \blacktriangleleft y$ gilt.

2.4.2 Ableseregeln für die Kausalitäten im System

Nun überlegen wir, wie wir direkt aus der Struktur des Netzes ablesen können, ob eine Kausalität im System gilt. Wir fangen mit dem 'einfachsten' Netz an, das ist ein Kausalnetz.

Beispiel 1:

Das in Abb. 2.4.4 dargestellte Netz hat keine Alternativen und auch keine Zyklen. Jede Stelle hat nur einen Vorgänger.

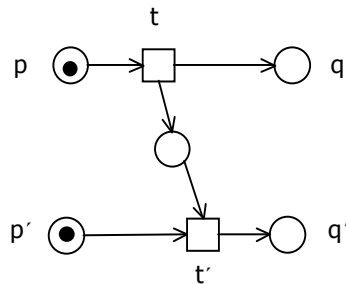


Abb. 2.4.4 Ein System, dessen Ablauf die gleiche Form hat

In diesem Fall haben das System und sein Ablauf die gleiche Form. Das liegt daran, dass das Netz des Systems ein Kausalnetz bezüglich F ist. Damit kann man die Kausalitäten, die im Ablauf des Systems gelten auch direkt am Netz sehen, z.B. von p zu q' gibt es einen Weg, damit gilt $\Sigma \models p \blacktriangleleft q'$. Es gibt keinen Weg von p' zu q , deshalb gilt $\Sigma \not\models p' \blacktriangleleft q$.

Bemerkung 2.4.3

Sei Netz $N=(P,T,F)$ ein Kausalnetz, $\Sigma=(N, m_0)$ ein System mit: $m_0(p)=1$ gdw. $p \in {}^\circ N$.

Dann gilt: Σ hat nur einen Ablauf. Dieser Ablauf kann durch (N, i) angegeben werden, wobei $i(x)=x$ für jedes $x \in N$.

Bemerkung

Kausalnetze mit Anfangsmarkierung $m_0(p)=1$ gdw. $p \in {}^\circ N$ sind 1-beschränkt.

Satz 2.4.4 (Ableseregeln der Relation \blacktriangleleft zwischen Stellen bzw. Transitionen in Kausalnetzen)

Sei Netz $N=(P,T,F)$ ein Kausalnetz. Sei $\Sigma=(N, m_0)$ ein System mit: $m_0(p)=1$ gdw. $p \in {}^\circ N$ (genau die Stellen ohne Vorgänger sind markiert).

Weiter seien $x, y \in P \cup T$.

Dann gilt: $\Sigma \models x \blacktriangleleft y \leftrightarrow x F^+ y$.

Beweis:

Nach der Bemerkung 2.4.3 hat Σ nur einen Ablauf und (N, i) beschreibt den Ablauf von Σ , wobei $i(x)=x$ für jedes $x \in N$. x, y kommen im Ablauf genau einmal vor, und die Kausalrelation $<$ im Ablauf ist gleich F^+ .

Also gilt wegen Def. 2.4.2 die Behauptung $\Sigma \models x \blacktriangleleft y, gdw. x F^+ y$. \square
 Jetzt wollen wir klären, inwieweit sich die Ableseregeln auch für Systeme mit Schleifen formulieren lassen. Systeme mit Schleifen haben unendliche Abläufe. Daher sind für diese Systeme Ableseregeln besonders interessant. Dazu betrachten wir das folgende Beispiel.

Beispiel 2:

Das in Abb. 2.4.5(a) dargestellte System hat eine Schleife aber keine Alternativen. Die in Abb. 2.4.5 (b) und (c) dargestellten Teilnetze N_{S_1} und N_{S_2} haben keine Schleife.

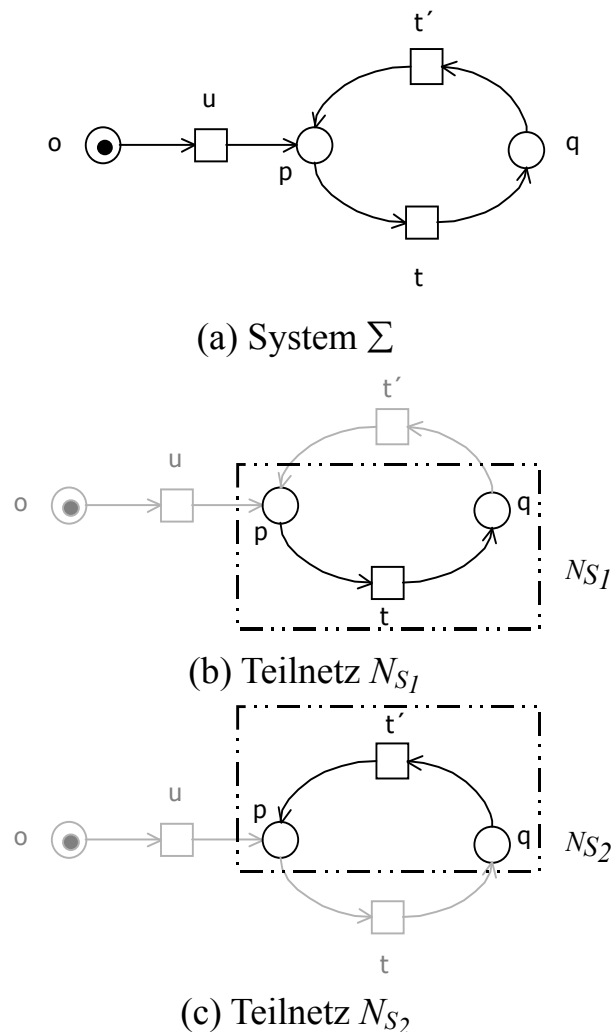


Abb. 2.4.5 System Σ mit Teilnetzen N_{S_1} und N_{S_2}

Das System Σ besitzt genau einen Ablauf, der unendlich ist.

Im Netz N_{S_1} (Abb. 2.4.5 (b)) sieht man, dass $p \text{ } F_{S_1} \text{ } q$ gilt. Da p im Ablauf von Σ immer wieder als erste Stelle von N_{S_1} markiert wird, gilt im Ablauf $p \blacktriangleleft q$, also $\Sigma \models p \blacktriangleleft q$.

Im Netz N_{S_2} (Abb. 2.4.5 (c)) sieht man zwar, dass $q \text{ } F_{S_2} \text{ } p$ gilt, kann aber nicht schliessen, dass $\Sigma \models q \blacktriangleleft p$ gilt, weil q im Ablauf von Σ nicht immer als erste Stelle von N_{S_2} markiert wird.

Also, wir können die Gültigkeit von $\Sigma \models p \blacktriangleleft q$ folgendermaßen ablesen: Zunächst suchen wir ein 'richtiges' Teilnetz N_S ohne Schleife. Und dann prüfen wir, ob $p \text{ } F_S \text{ } q$ gilt (d.h. ob es im Netz N_S einen Weg von p zu q gibt).

Ein 'richtiges' Teilnetz N_S muss folgende Bedingung erfüllen: Jede Stelle aus ${}^\circ N_S$ wird in jedem Ablauf immer als erste Stelle in N_S markiert. Eine Stelle kann als eine erste Stelle in N_S in einem Ablauf markiert werden, wenn in dieser Stelle eine Kante endet, die nicht zu N_S gehört. Wenn nur in ${}^\circ N_S$ solche Kanten enden, dann wird N_S ein 'richtiges' Teilnetz sein.

Definition 2.4.5 (Vollständige Teilnetze bzgl. Vor- und Nachbereichen von Transitionen)

Sei $N=(P,T,F)$ ein Netz.

$N_s=(P_s,T_s,F_s)$ ist ein vollständiges Teilnetz bzgl. Vor- und Nachbereichen von Transitionen (kurz: vollständiges Teilnetz) von N , wenn:

$$P_s \subseteq P;$$

$$T_s \subseteq T;$$

$$F_s = F \cap (P_s \times T_s \cup T_s \times P_s) \text{ und}$$

$$\text{für jedes } t \in T_s \text{ gilt } \bullet t \cup t \bullet \subseteq P_s.$$

Definition 2.4.6

Sei $N=(P,T,F)$ ein Netz mit: F ist unverzweigt an Stellen, d.h. für jedes $p \in P$ gilt $|p \bullet| \leq 1$.

Sei $\Sigma=(N, m_0)$ ein 1-beschränktes System.

Sei $N_s=(P_s,T_s,F_s)$ ein vollständiges Teilnetz von N mit: N_s ist ein Kausalnetz.

Sei (K, r) ein Ablauf von Σ , wobei $K=(B, E, \triangleleft)$.

Wir sagen, in N_s werden immer ${}^\circ N_s$ als erste in K markiert, wenn die folgenden Bedingungen erfüllt sind:

- i. Für jedes $p \in {}^\circ N_s$ gilt: Für jedes $b \in K$ mit $r(b) = p$ gilt: entweder $b \in {}^\circ K$ oder es gibt ein e mit $r(e) \in T \setminus T_s$ und $e \triangleleft b$.
- ii. Für jedes $p \in P_s \setminus {}^\circ N_s$ gilt: Für jedes $b \in K$ mit $r(b) = p$ gilt: Es gibt ein e mit $r(e) \in T_s$ und $e \triangleleft b$.

Beispiel 3:

Das in Abb. 2.4.6 dargestellte System hat Schleifen aber keine Alternativen.

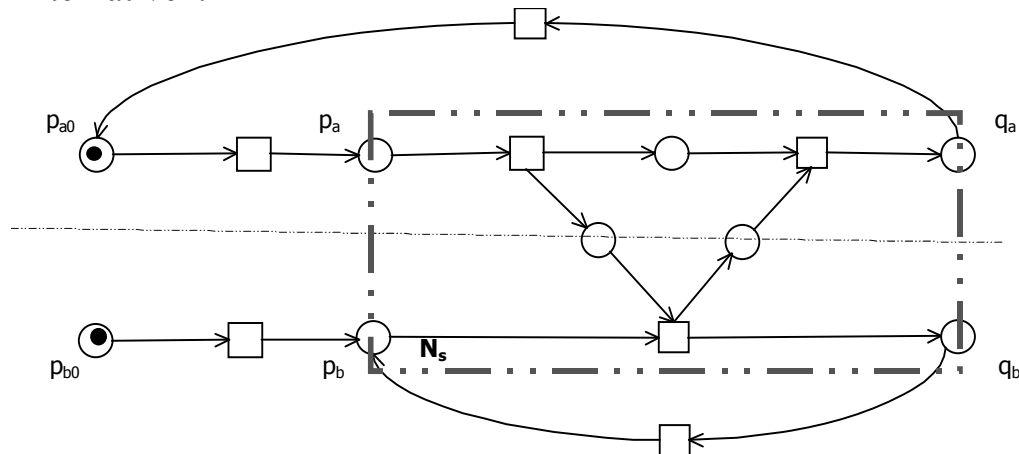


Abb. 2.4.6 System Σ mit Teilnetz N_s

N_s ist ein Teilnetz von Σ und es ist ein Kausalnetz. Weiter ist ${}^\circ N_s = \{p_a, p_b\}$. Im Ablauf von Σ kommen p_a, p_b jedesmal zusammen vor, also die Elemente von ${}^\circ N_s$ kommen immer zusammen vor.

Definition 2.4.7 (Erstes und folgendes Vorkommen einer Stelle)

Sei Σ ein System, $p \in P$ eine Stelle von Σ .

Sei (K, r) ein Ablauf von Σ , $b, b' \in B$ mit $r(b) = r(b') = p$.

- i. b ist das erste Vorkommen von p in K , gdw. es kein $b'' \in K$ mit $r(b'') = p$ und $b'' < b$ gibt.
- ii. b' ist das nächste Vorkommen nach b von p in K , gdw. $b < b'$ gilt und es kein $b'' \in K$ mit $r(b'') = p$ und $b < b'' < b'$ gibt.

Definition 2.4.8 (Zwei Stellen kommen immer zusammen vor)

Sei Σ ein System, seien $p, q \in P$ zwei Stellen von Σ .

Sei (K, r) ein Ablauf von Σ .

$B_p := \{b \in B \mid r(b) = p\}$, $B_q := \{b \in B \mid r(b) = q\}$.

Wir sagen, p, q kommen in K immer zusammen vor, wenn es eine

Relation $R \subseteq B_p \times B_q$ gibt, die wir wie folgt induktiv definieren:

1. Wenn gilt:

- Wenn es ein $u \in K$ gibt, das das erste Vorkommen von p in K ist, dann gibt es ein $v \in K$, das das erste Vorkommen von q in K ist.
- Wenn es ein $v \in K$ gibt, das das erste Vorkommen von q in K ist, dann gibt es ein $u \in K$, das das erste Vorkommen von p in K ist.

Dann gilt: $(u, v) \in R$.

2. Angenommen, $(u, v) \in R$.

Wenn gilt:

- Wenn es ein $u' \in K$ gibt, das das nächste Vorkommen nach u von p in K ist, dann gibt es ein $v' \in K$, das das nächste Vorkommen nach v von q in K ist.
- Wenn es ein $v' \in K$ gibt, das das nächste Vorkommen nach v von q in K ist, dann gibt es ein $u' \in K$, das das nächste Vorkommen nach u von p in K ist.

Dann gilt: $(u', v') \in R$.

Definition 2.4.9 (Stellen aus $Q \subseteq P$ kommen immer zusammen vor)

Sei Σ ein System, $Q \subseteq P$.

Sei (K, r) ein Ablauf von Σ .

Wir sagen, Elemente aus Q kommen in K immer zusammen vor, wenn für beliebige $p, q \in Q$ gilt: p, q kommen in K immer zusammen vor.

In Abb. 2.4.7 ist ein Anfangsstück des Ablaufs K von Σ dargestellt.

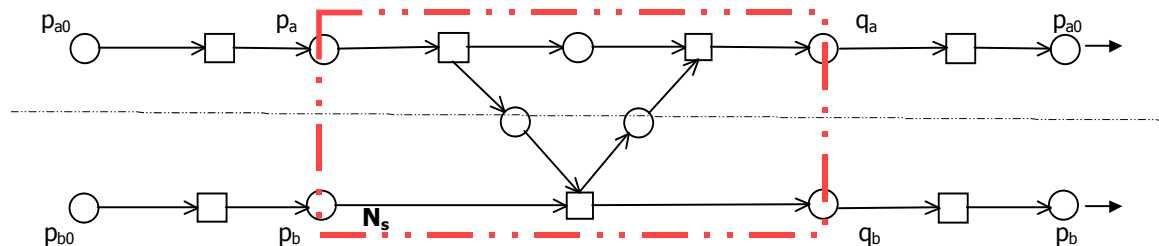


Abb. 2.4.7 Ein Anfangsstück des Ablaufs K von Σ in Abb. 2.4.6

Der Teil im roten Kästchen ist ein Teilablauf von K , der gerade ein Auftreten von N_s ist. Teilabläufe sind zusammenhängend, was von vollständigen Teilnetzen nicht gefordert war.

Definition 2.4.10 (Teilablauf eines Ablaufs)

Sei Σ ein System, sei (K, r) ein Ablauf von Σ .

Sei K_s ein vollständiges Teilnetz von K .

K_s ist ein Teilablauf von K , gdw.

- i. K_s ist ein Kausalnetz;
- ii. Es gibt in K einen Schnitt C mit ${}^\circ K_s \subseteq C$;
- iii. Für jeden Schnitt C in K mit ${}^\circ K_s \subseteq C$ gilt:

$$C' := (C \setminus {}^\circ K_s) \cup K_s. C' \text{ ist ein Schnitt in } K.$$

Definition 2.4.11 (Auftreten eines Teilnetzes in einem Ablauf)

Sei $\Sigma = (N, m_0)$ ein System, $N_s = (P_s, T_s, F_s)$ ein Teilnetz von N mit: N_s ist ein Kausalnetz.

Sei (K, r) ein Ablauf von Σ , $K_s = (B_s, E_s, \leq_s)$ ein Teilablauf von K mit:

Für zwei verschiedene Elemente $s, s' \in K$ gilt $r(s) \neq r(s')$.

K_s ist ein N_s -Auftreten in K , gdw.

$$r(B_s) = P_s, r(E_s) = T_s \text{ und } \forall s, s' \in K_s: s \leq s' \leftrightarrow r(s) F_s r(s').$$

(d.h. K_s und N_s sind isomorph bezüglich r)

Im Beispiel 3 (siehe Abb. 2.4.6, Abb. 2.4.7) sieht man, die Elemente von N_s , z.B. p_a, q_b kommen jedesmal nur in einem N_s -Auftreten vor.

Deshalb gilt $\Sigma \models p_a \blacktriangleleft q_b$, weil wir im Teilnetz N_s ablesen können, dass $p_a F_s^+ q_b$ gilt.

Bemerkung 2.4.12 (Auftreten eines $p \in {}^\circ N_s$ in einem Ablauf von Σ sichert das Auftreten von N_s)

Sei $N = (P, T, F)$ ein Netz mit: F ist unverzweigt an Stellen, d.h. für jedes $p \in P$ gilt $|p^\bullet| \leq 1$.

Sei $\Sigma = (N, m_0)$ ein 1-beschränktes System, und (K, r) ein Ablauf.

Sei $N_s = (P_s, T_s, F_s)$ ein vollständiges Teilnetz von N mit:

- i. N_s ist ein Kausalnetz;
- ii. Die Stellen aus ${}^\circ N_s$ werden in N_s immer als erste im Ablauf K markiert;
- iii. Die Stellen aus ${}^\circ N_s$ kommen im Ablauf K immer zusammen vor.

Wenn $b \in K$ mit $r(b) = p$ wobei $p \in {}^\circ N_s$, dann gibt es ein N_s -Auftreten K_s mit $b \in K_s$.

Beweis:

Angenommen, $b \in K$ mit $r(b) = p$, wobei $p \in {}^\circ N_s$.

Zu zeigen ist: Es gibt ein N_s -Auftreten K_s mit $b \in K_s$.

Es gibt ein $B_I \subseteq B$ mit $b \in B_I$ und $r(B_I) = {}^\circ N_s$ und B_I ist eine *co*-Menge (da Elemente aus ${}^\circ N_s$ immer zusammen vorkommen, Voraussetzung *iii*).

In N gibt es keine alternativen Transitionen zu $({}^\circ N_s)^\bullet$.

Auf B_I folgt ein vollständiges N_s -Auftreten, da N_s ein Kausalnetz ist (Voraussetzung *i*).

Also, es gibt ein N_s -Auftreten in K , das b enthält. □

Wir verallgemeinern jetzt noch Bem 2.4.12 dahingehend, dass schon das Auftreten einer beliebigen Stelle aus P_s ein vollständiges Auftreten von N_s sichert.

Bemerkung 2.4.13 (*Auftreten von $p \in N_s$ in einem Ablauf von Σ*)

Sei $N = (P, T, F)$ ein Netz mit: F ist unverzweigt an Stellen, d.h. für jedes

$p \in P$ gilt $|p^\bullet| \leq 1$.

Sei $\Sigma = (N, m_0)$ ein 1-beschränktes System, und (K, r) ein Ablauf.

Sei $N_s = (P_s, T_s, F_s)$ ein vollständiges Teilnetz von N mit:

i. N_s ist ein Kausalnetz;

ii. Die Stellen aus ${}^\circ N_s$ werden in N_s immer als erste im Ablauf K markiert;

iii. Die Stellen aus ${}^\circ N_s$ kommen im Ablauf K immer zusammen vor.

Für jedes $p \in N_s$ gilt:

Für jedes $b \in K$ mit $r(b) = p$ gilt: b steht in einem N_s -Auftreten.

Beweis:

Fall 1. $p \in {}^\circ N_s$:

Angenommen, $b \in K$ mit $r(b) = p$.

Wegen Bem. 2.4.12 gilt: b steht in einem N_s -Auftreten.

Fall 2. $p \in P_s \setminus {}^\circ N_s$:

Angenommen, $b \in K$ mit $r(b) = p$.

Wir nehmen an, b steht nicht in einem N_s -Auftreten.

Dann gibt es ein $t \in T \setminus T_s$ mit $r(e)=t$, wobei $e \triangleleft b$.
Das ist ein Widerspruch dazu, dass nur Stellen aus ${}^\circ N_s$ als erste markiert werden (Voraussetzung ii und Def. 2.4.6 ii).
Daraus folgt, b steht in einem N_s -Auftreten. □

Satz 2.4.14 (*Ableseregel für Netze mit Schleifen ohne Alternativen*)
Sei $N=(P,T,F)$ ein Netz mit: F ist unverzweigt an Stellen, d.h. für jedes $p \in P$ gilt $|p^\bullet| \leq 1$.

Sei $\Sigma=(N, m_0)$ ein 1-beschränktes System, und (K, r) ein Ablauf.

Weiter seien $p, q \in P$ zwei verschiedene Stellen.

Sei $N_s=(P_s, T_s, F_s)$ ein vollständiges Teilnetz von N mit $p, q \in P_s$ mit:

- i. N_s ist ein Kausalnetz;
- ii. Im Ablauf von Σ werden die Stellen aus ${}^\circ N_s$ in N_s immer als erste markiert;
- iii. Die Stellen aus ${}^\circ N_s$ kommen im Ablauf von Σ immer zusammen vor.

Es gilt: $\Sigma \models p \blacktriangleleft q$, wenn $p \in F_s^+ q$.

Beweis:

Sei (K, r) der Ablauf von Σ .

Angenommen, $p \in F_s^+ q$ gilt. Zu zeigen ist: Für jedes $b' \in K$ mit $r(b')=q$ gibt es ein $b \in K$ mit $r(b)=p$ und $b \triangleleft b'$ und wenn $b \triangleleft b'' \triangleleft b'$, so $r(b'') \neq q$.

Angenommen, $b' \in K$ mit $r(b')=q$.

Dann gibt es ein N_s -Auftreten K_s mit $b' \in K_s$ in K (wegen Bem. 2.4.13).

Dann gibt es ein $b \in K_s$ mit $r(b)=p$ (wegen $p \in N_s$) und $b \triangleleft b'$ (wegen $p \in F_s^+ q$).

Es gibt kein $b'' \in K_s$ mit $r(b'')=q$ und $b'' \neq b'$ (da q in K_s nur einmal vorkommt).

Deshalb gilt auch: Es gibt kein $b'' \in K_s$ mit $r(b'')=q$ und $b \triangleleft b'' \triangleleft b'$.

Also, es gibt ein $b \in K$ mit $r(b)=p$ und $b \triangleleft b'$ und wenn $b \triangleleft b'' \triangleleft b'$, so $r(b'') \neq q$.

Daraus folgt die Behauptung. □

Für die bei einer verteilenden Verfeinerung eingesetzten Verfeinerungsnetze (siehe Katipel 6) wird immer einer der beiden Ableseregeln anwendbar sein. Deshalb beschränken wir uns hier auf diese beiden Ableseregeln.

Um den Satz besser zu verstehen, sehen wir uns noch ein Beispiel an. In Abb.2.4.8 und Abb. 2.4.9 werden zwei Systeme Σ_A, Σ_B dargestellt. Für beide Systeme gilt im Teilsystem² $N_t \quad p \blacktriangleleft q$. Aber $p \blacktriangleleft q$ gilt in Σ_A und in Σ_B nicht.

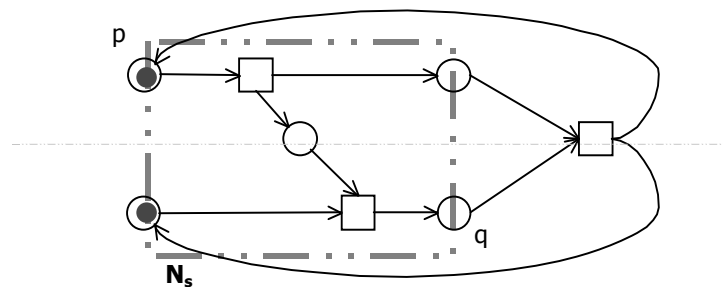


Abb. 2.4.8 Σ_A

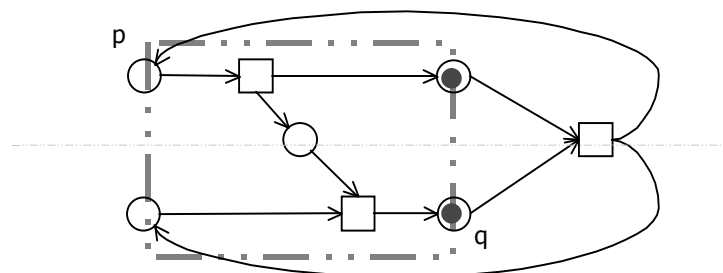


Abb. 2.4.9 Σ_B

Der Grund dafür ist folgender. Für die Stelle q aus der Stellenmenge von N_t aber nicht aus ${}^\circ N_t$ gilt: In Σ_A kommt q immer in einem Auftreten von N_t vor (Abb. 2.4.10), in Σ_B kommt q beim ersten mal nicht in einem Auftreten von N_t vor (Abb. 2.4.11).

² Hier gemeint ist das System (N_b, m_0) , wobei m_0 die Anfangsmarkierung ist, mit: $m_0(p)=1$ gdw. $p \in {}^\circ N_t$.

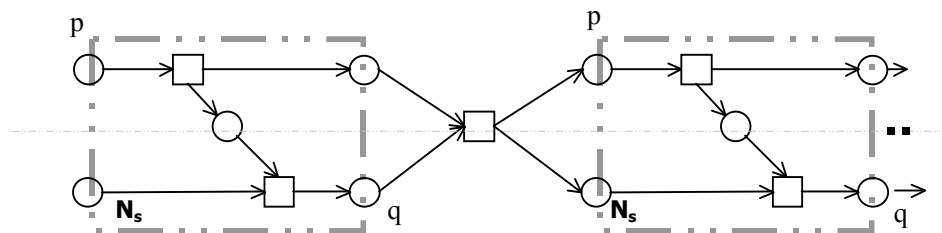


Abb. 2.4.10 Ablauf von Σ_A

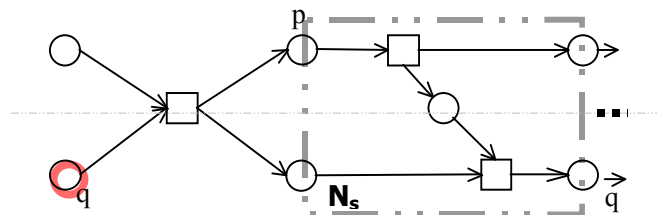


Abb. 2.4.11 Ablauf von Σ_B

Zusammenhang mit der Blockbedingung

Die Bedingung *iii* im Satz 2.4.14 sichert für die betrachteten Systeme (keine Alternative) folgendes: Wenn eine Transition vom Teilnetz auftritt, dann tritt das Teilnetz vollständig auf. Das ist ähnlich wie in der Bemerkung 6.2.1 über die Blockbedingung im Kapitel 6.

Der Satz 2.4.14 besagt, wenn jede Stelle aus dem Teilnetz N_s nur beim Auftreten von N_s vorkommt, dann garantiert die Gültigkeit einer Kausalität im Teilnetz die Gültigkeit dieser Kausalität im ganzen System.

Im nächsten Kapitel werden wir zeigen, wenn die Blockbedingung erfüllt ist, dann garantiert die Gültigkeit einer Kausalität im lokalen Teilsystem, dass die Gültigkeit dieser Kausalität im System erhalten bleibt. Z.B. das System Σ_A (Abb. 2.4.8) ist eine Verfeinerung von Σ in Abb. 2.4.12, die die Blockbedingung erfüllt. Wie schon erwähnt, im Teilnetz N_t gilt $p \blacktriangleleft q$ und im System Σ vor der Verfeinerung gilt $p \blacktriangleleft q$. Deshalb gilt im System Σ_A nach der Verfeinerung $p \blacktriangleleft q$ auch noch.

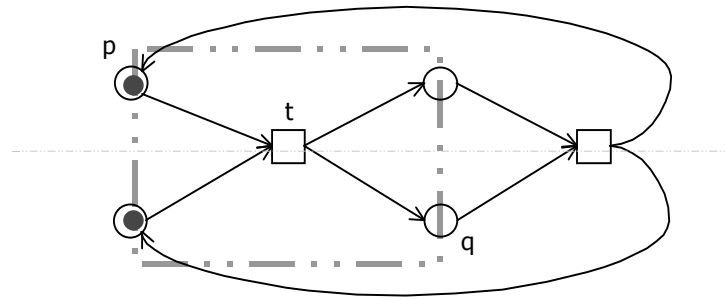


Abb. 2.4.12 System Σ

Für beliebige Systeme, also Systeme, deren Netze Schleifen und Alternativen enthalten, reicht es nicht, kausale Teilnetze zu betrachten, deren Vorbereiche immer als erste und gleichzeitig markiert werden. Die Forderungen reichen nicht aus zu sichern, dass das Teilnetz in jedem Ablauf vollständig auftritt. Erst im Zusammenhang mit der Blockbedingung für Teilnetze (siehe Kapitel 3) können für Agentensysteme (siehe Kapitel 4) Kriterien dafür angegeben werden, wann ein Teilnetz in jedem Ablauf vollständig auftritt (Sätze aus Kapitel 6).

3. Halbordnung erhaltende Transitionsverfeinerung

Als Vorbereitung auf die in dieser Arbeit definierte verteilende Transitionsverfeinerung betrachten wir die von S. Peuker [Peu01] vorgestellte Transitionsverfeinerung mit Blockbedingung. Als Grundlage dazu geben wir im ersten Abschnitt eine Übersicht über vorangehende Verfeinerungsbegriffe. Transitionsverfeinerung mit Blockbedingung erhält wichtige Eigenschaften des verfeinerten Systems. Welche Eigenschaften sicher erhalten bleiben und einige Eigenschaften, die verloren gehen können, werden angegeben. Wir zeigen, dass mit Hilfe von zusätzlichen Kausalitätsforderungen auch diese Eigenschaften des Systems erhalten werden können. Weitere Eigenschaften, die erst bei der Zusammenarbeit mehrerer durch Nachrichten kommunizierender Agenten wichtig sind, werden dann im nächsten Abschnitt nach der Definition der Agentennetze behandelt.

3.1 Verfeinerungsbegriffe, eine Übersicht

Um den neuen Verfeinerungsbegriff verständlicher zu machen, möchten wir in diesem Abschnitt einen Rückblick darauf geben, wie sich die grundlegenden Verfeinerungsbegriffe in der Verifikation verteilter Algorithmen entwickelt haben. Von dem gebräuchlichsten Verfeinerungsbegriff aus hat man die Beobachterverfeinerung (vgl. [AL91]) entwickelt, damit man Daten verfeinern kann. Aber mit diesem Verfeinerungsbegriff kann man eine Aktion (Transition) nicht verfeinern. Dann hat man die Transitionsverfeinerung entwickelt. Eine Transitionsverfeinerung, die auf verteilten Abläufen basiert, ist Vogler's Transitionsverfeinerung (siehe z.B. [Vog87],[Vog92],[BGV90]). Aber dieser Verfeinerungsbegriff lässt nur eine sehr eingeschränkte Klasse von Verfeinerungsnetzen zu, die für viele Anwendungsfälle zu speziell ist. Deshalb wurde in [Peu01] eine allgemeinere Form der Transitionsverfeinerung definiert, die wir in den folgenden Abschnitten etwas genauer betrachten werden, die Transitionsverfeinerung mit Blockbedingung. Auf dieser Definition baut dann unsere Definition der verteilenden Verfeinerung auf.

So unterschiedlich die Verfeinerungsbegriffe sind, haben sie eines gemeinsam: Die Korrektheit des zu verfeinernden Systems muss bei einem Verfeinerungsschritt erhalten bleiben, und das wird versucht, durch

möglichst lokale Eigenschaften der Verfeinerung zu erreichen.

3.1.1 Der erste Verfeinerungsbegriff

Gebräuchlichst wird der Begriff Verfeinerung wie folgt definiert (siehe z.B. [AL91]): Angenommen, wir haben zwei Systeme – ein altes System Σ_1 und ein neues System Σ_2 (*alt* und *neu* beziehen sich auf *vor* und *nach* einem Verfeinerungsschritt eines Systems). Σ_2 ist eine Verfeinerung von Σ_1 , wenn jeder sequentielle Ablauf von Σ_2 ein sequentieller Ablauf von Σ_1 ist. D.h. Σ_2 soll keinen für Σ_1 unerwarteten Ablauf haben.

Warum soll das neue System keine solchen Abläufe haben, die das alte System nicht hat? Wie vergleicht man zwei sequentielle Abläufe? Wir sehen uns ein Beispiel an.

In Abb. 3.1.1 ist ein System Σ_1 angegeben. Links ist die Prozedur l , rechts ist die Prozedur r , in der Mitte ist eine Stelle v (v modelliert z.B. eine shared-Variable).

Die Prozedur l hat eine Aktion W . Durch die Ausführung von W wird l die Variable v schreiben. l hat eine lokale Variable v_l mit Anfangswert 10. Wenn die Aktion W (*Write*) ausgeführt wird, wird v auf den aktuellen Wert von v_l gesetzt, also $v = v_l$.

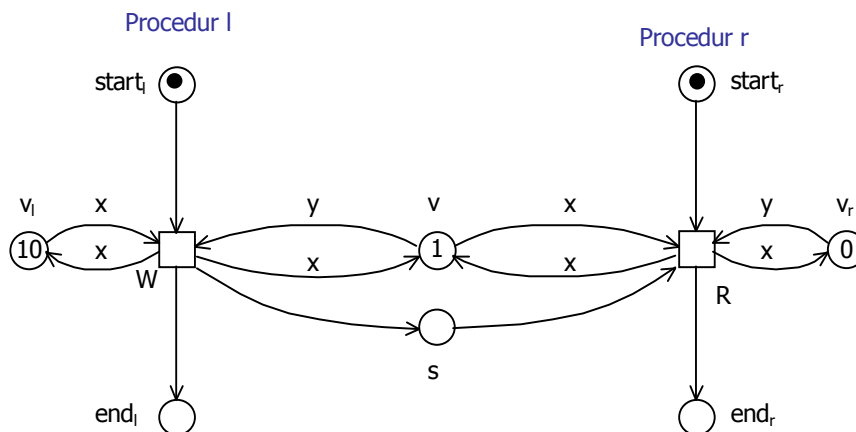


Abb. 3.1.1 System Σ_1 (*Write / Read*)

Auf der rechten Seite hat die Prozedur r eine Aktion R (*Read*). r hat eine lokale Variable v_r mit Anfangswert 0. Bei der Ausführung von R wird die Variable v gelesen und v_r wird auf den aktuellen Wert von v gesetzt, also $v_r = v$.

Der Anfangswert von v ist gleich 1 (das Token 1 an der Stelle v).

Die Stelle s sorgt dafür, dass die Aktion W vor der Aktion R ausgeführt wird. Damit hat Σ_1 nur einen Ablauf ρ_1 :

$$\begin{array}{rcccc}
 v_l: & 10 & & 10 & & 10 \\
 v: & 1 & \xrightarrow{W} & 10 & \xrightarrow{R} & 10 \\
 v_r: & 0 & & 0 & & 10
 \end{array} \quad (3-1-1)$$

d.h. im Anfangszustand gilt: $v_l = 10, v = 1, v_r = 0$. Die Aktion W wird zunächst ausgeführt, der Wert von v wird zu 10 geändert. Und dann wird die Aktion R ausgeführt, der Wert von v_r wird zu 10 geändert (Synchronisation zwischen W und R durch die Stelle s).

In Abb. 3.1.2 ist ein System Σ_2 angegeben.

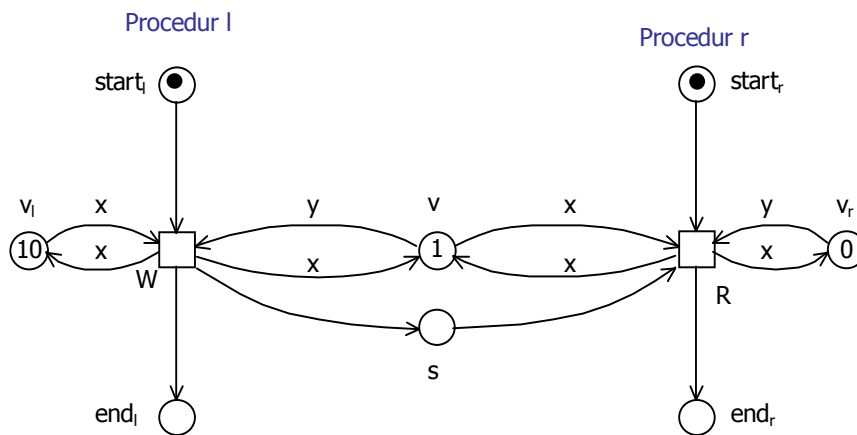


Abb. 3.1.2 System Σ_2 (ein weiteres System *Write / Read*)

Σ_2 sieht fast genauso wie Σ_1 aus, und der Ablauf ρ_1 (3-1-1) ist auch ein Ablauf von Σ_2 . Σ_2 hat aber noch einen weiteren Ablauf ρ_2 :

$$\begin{array}{rcccc}
 v_l: & 10 & & 10 & & 10 \\
 v: & 1 & \xrightarrow{R} & 1 & \xrightarrow{W} & 10 \\
 v_r: & 0 & & 1 & & 1
 \end{array} \quad (3-1-2)$$

Also, im Ablauf ρ_2 wird v zunächst gelesen und dann geschrieben. Die Ergebnisse in ρ_1 und in ρ_2 sind *ganz verschieden*. Der Ablauf ρ_2 ist kein Ablauf von Σ_1 , anders gesagt, ρ_2 ist ein für Σ_1 unerwarteter Ablauf (also Σ_2 ist keine Verfeinerung von Σ_1 , Σ_1 ist aber eine Verfeinerung von Σ_2). Eine Verfeinerung beschreibt also einen Spezialfall des gegebenen Algorithmus. Er schränkt die Anzahl der Abläufe ein.

Wie vergleichen wir zwei sequentielle Abläufe? Wir untersuchen, ob die

Folgen der Schritte der beiden Abläufe gleich sind. Wie vergleichen wir zwei Schritte? Wir sehen nach, ob in den beiden Schritten die Änderungen der globalen Zustände beider Systeme gleich sind. Z.B. der Ablauf ρ_2 ist nicht gleich dem Ablauf ρ_1 .

3.1.2 Beobachterverfeinerung

Beobachterverfeinerung werden wir zur Verfeinerung eines einfachen Petrinetzmodells zu einem Agentennetz (Abschnitt 4.3.2) verwenden. Das kann auch in anderen Fällen hilfreich sein, wenn sich die Eigenschaften des zu entwerfenden Algorithmens durch ein besonders einfaches Petrinetzmodell spezifizieren lassen, das Agentennetz, das zusätzlich die algorithmische Idee modelliert aber deutlich komplizierter ist.

Es gibt oft solche Verfeinerungsschritte, durch die neue Variablen dem System hinzugefügt werden.

Beispiel: *time*

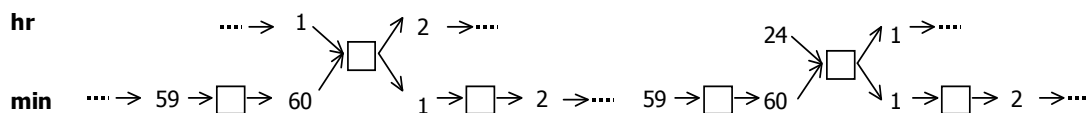


Abb. 3.1.3 System Σ_1 (*time*)

In Abb. 3.1.3 ist ein System Σ_1 angegeben, das ist ein Zeit-System. In Σ_1 gibt es zwei Variablen *hr* und *min*. *hr* erhöht sich immer um 1, wenn *min*=60.

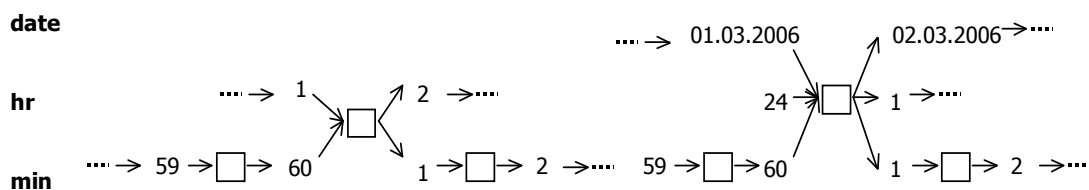


Abb. 3.1.4 System Σ_2 (*time* mit Datum)

In Abb. 3.1.4 ist das neue System Σ_2 angegeben. In Σ_2 gibt es eine neue Variable *date*, sie erhöht sich immer um 1, wenn *hr*=24.

Offensichtlich verhalten sich die alten Variablen *hr* und *min* in Σ_2 genauso wie in Σ_1 . Andererseits gibt es in Σ_2 eine Variable mehr als in Σ_1 , d.h. die beiden Systeme Σ_1 und Σ_2 haben unterschiedliche Zustandsräume. Wie vergleichen wir die globalen Zustände beider Systeme? Bei Σ_2 beobachten wir nur die alten Variablen *hr* und *min*. So erhalten wir für

jeden sequentiellen Ablauf von Σ_2 einen sequentiellen Beobachterablauf. Wenn dieser Beobachterablauf gleich einem sequentiellen Ablauf von Σ_1 ist, dann ist Σ_2 eine Beobachterverfeinerung von Σ_1 .

Offenbar ist Σ_2 (siehe Abb. 3.1.4) eine Beobachterverfeinerung von Σ_1 (siehe Abb. 3.1.3).

Bei Beobachterverfeinerungsbegriff kann man neue Variablen einführen. Aber bei diesem Verfeinerungsbegriff kann man Aktionen nicht zerlegen.

3.1.3 Vogler's Transitionsverfeinerung

Oft gibt es folgende Verfeinerungsschritte: Eine Aktion im alten System wird zu mehreren Aktionen verfeinert, d.h. die Aktion wird im alten System in nur einem Schritt ausgeführt, sie wird aber im neuen System in mehrere Schritte aufgeteilt.

Ein Verfeinerungsbegriff, mit dem man eine Aktion verfeinern kann und der auf verteilten Abläufen basiert, ist Vogler's Transitionsverfeinerung (siehe z.B. [Vog87],[Vog92],[BGV90]).

Beispiel: *producer/consumer*

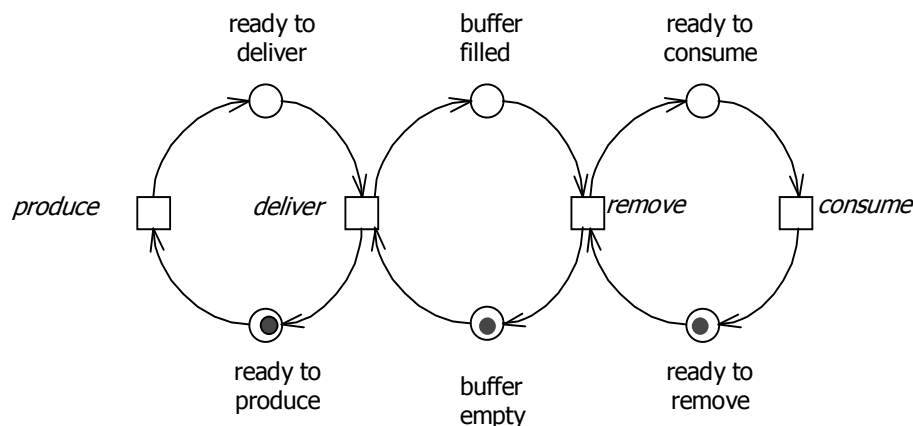


Abb. 3.1.5 System Σ_1 (*producer/consumer*)

In Abb. 3.1.5 ist das System Σ_1 angegeben. Das ist ein bekanntes Beispiel (z.B. in [Rei98]) - das System *producer/consumer*.

Wir verfeinern die Transition *produce* zu zwei Transitionen *produce a* und *produce b*. Das neue System Σ_2 ist in Abb. 3.1.6 angegeben.

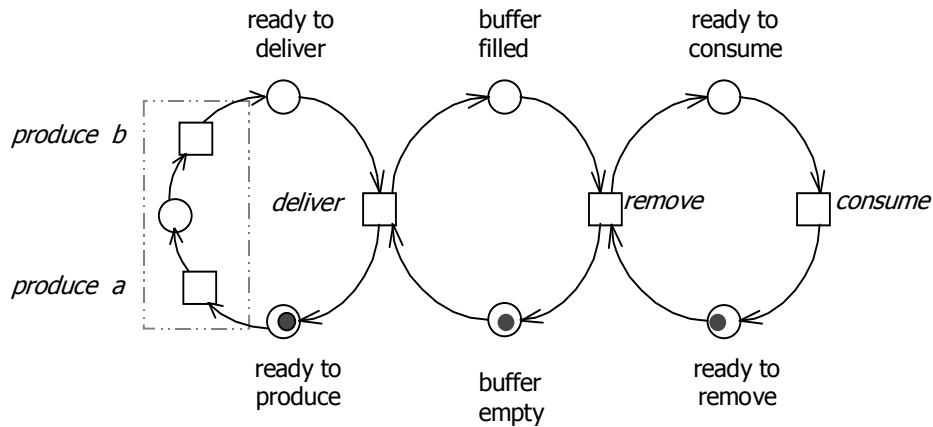
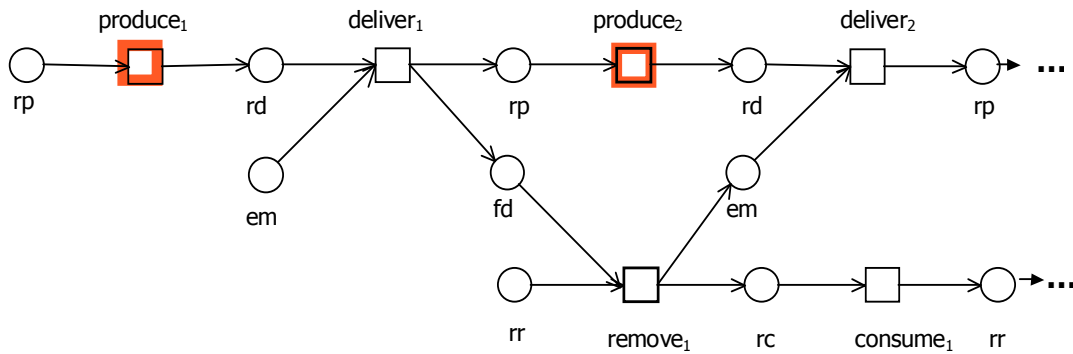
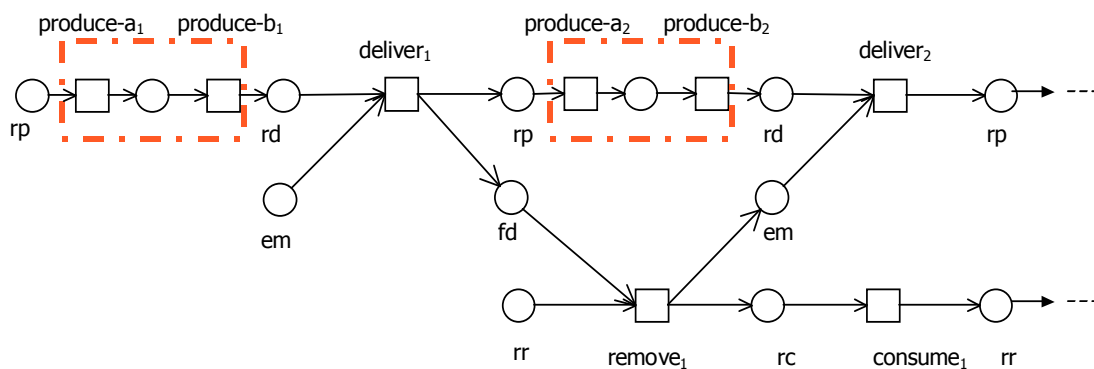


Abb. 3.1.6 System Σ_2 (ein verfeinertes *producer/consumer*)

Intuitiv wissen wir schon, dass sich die beiden Systeme ähnlich verhalten. In Abb. 3.1.7 sind die verteilten Abläufe K_1 und K_2 der beiden Systeme Σ_1 und Σ_2 angegeben.



(a) Ablauf K_1 von Σ_1



(b) Ablauf K_2 von Σ_2

Abb. 3.1.7 Verteilte Abläufe von Σ_1 und Σ_2

Bei diesem Verfeinerungsbegriff vergleicht man verteilte Abläufe der Systeme. In Abb. 3.1.7 sehen wir, jedes rot gestrichene Kästchen in K_2 entspricht einem Auftreten der Transition *produce* in K_1 . Und alle Kausal-Ordnungen von K_1 bleiben in K_2 erhalten. Nach diesem Verfeinerungsbegriff ist Σ_2 eine Transitionsverfeinerung von Σ_1 .

3.1.4 Vogler's Transitionsverfeinerung und unsere verteilende Verfeinerung

Bei Vogler's Verfeinerungsbegriff \sqsubseteq_v bleiben alle Kausal-Ordnungen der Aktionen des alten Systems erhalten. Dieser Verfeinerungsbegriff verlangt die *input-Synchronisation* und *output-Synchronisation* (vgl. [BGV90]).

Beispiel: *Phasensynchronisation* (für zwei Agenten)

Das System Σ_1 in Abb. 3.1.8 ist ein vereinfachter Phasensynchronisations-Algorithmus für zwei Agenten a und b . Jeder Agent hat zwei Arbeiten zu machen (oder sagen wir, zwei Phasen einer Arbeit), zunächst die erste Arbeit t_{1a} und t_{1b} , und danach die zweite t_{2a} und t_{2b} . Sie haben eine gemeinsame Transition t , die Folgendes beschreibt: Wenn sie beide mit der ersten Phase der Arbeit fertig sind, dann können sie mit der zweiten Phase beginnen.

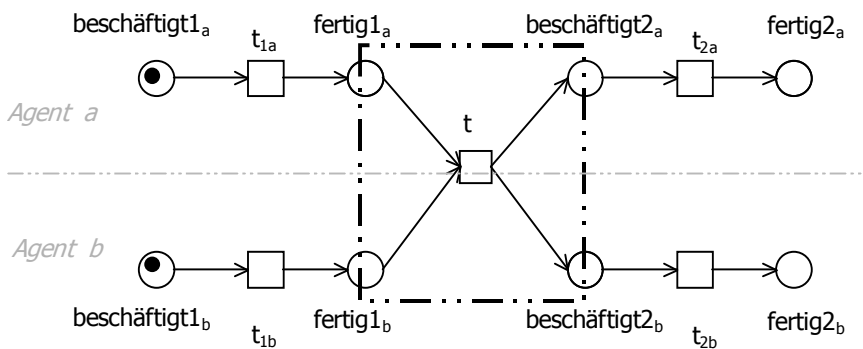


Abb. 3.1.8 System Σ_1 (vereinfachter *Phasensynchronisation*)

Das System Σ_2 in Abb. 3.1.9 ist eine Verfeinerung von Σ_1 laut \sqsubseteq_v .

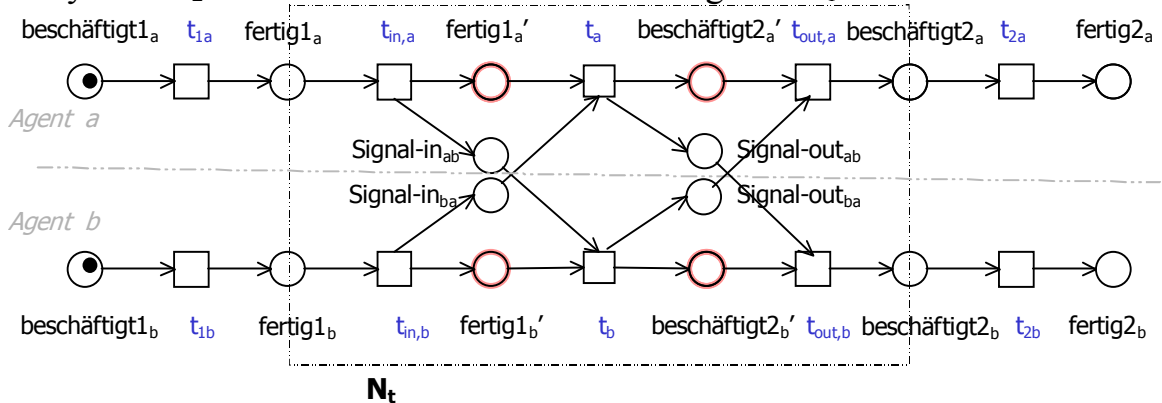
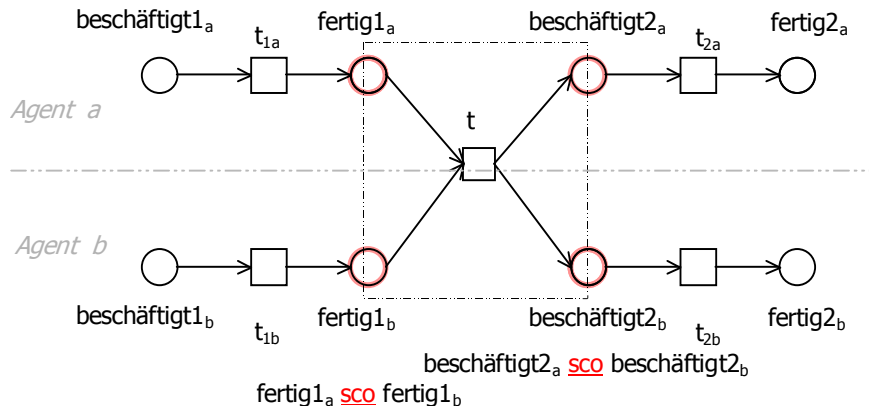
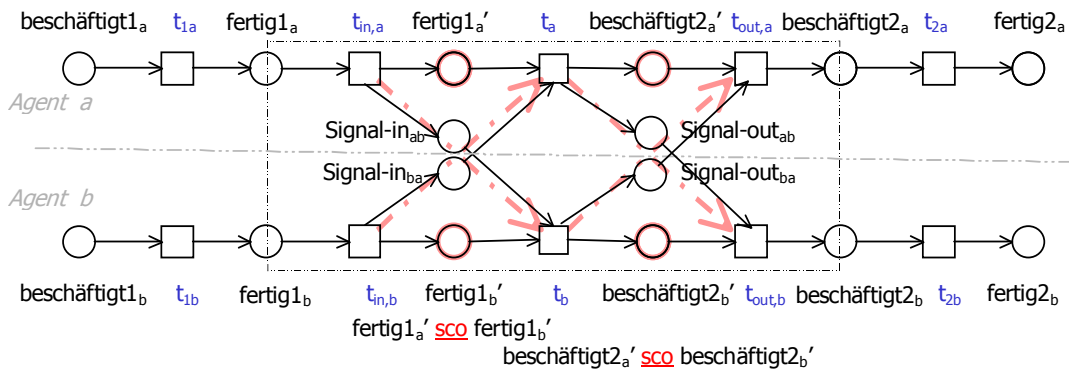


Abb. 3.1.9 System Σ_2

Die Transition t in Σ_1 wird zu einem Netz N_t verfeinert.



(a) Ablauf K_1 von Σ_1



(b) Ablauf K_2 von Σ_2

Abb. 3.1.10 Verteilte Abläufe von Σ_1 und Σ_2 ³

Die Kausal-Ordnungen der Aktionen

$$(t_{1b} < t_a) \wedge (t_{1a} < t_b) \quad (3-1-3)$$

garantieren die *input*-Synchronisation⁴ im verteilten Fall. Und die Kausal-Ordnungen

$$(t_b < t_{2a}) \wedge (t_a < t_{2b}) \quad (3-1-4)$$

garantieren die *output*-Synchronisation⁵ im verteilten Fall.

³ Der Begriff *strong concurrency* (*sco*) aus der Literatur [Rei87] hat uns zu der Idee inspiriert, mit Hilfe von Kausalitäten verteilende Verfeinerung zu analysieren. Bei *sco* erhält ein System durch ein Paar von kausalen Ordnungen auch im verteilten Fall solche Eigenschaften, die es normalerweise wegen einer unverteilten Transition erhält. Z.B. in Abb. 3.1.10 hat jeder sequentielle Ablauf von K_1 wegen der unverteilten Transition t einen Schnitt, der $fertig1_a$ und $fertig1_b$ enthält. Durch das Paar von kausalen Ordnungen $\cdot fertig1_b' < fertig1_a' \cdot$ und $\cdot fertig1_a' < fertig1_b' \cdot$ hat auch jeder sequentielle Ablauf von K_2 einen Schnitt, der $fertig1_a'$ und $fertig1_b'$ enthält.

⁴ *input*-Synchronisation bedeutet: Ein Agent wird erst seine Aufgabe für t ausführen, nachdem alle Vorbedingungen für t von allen Agenten schon erfüllt worden sind. Z.B. bevor t_a schaltet, sind $fertig1_a, fertig1_b$ erfüllt worden sind.

⁵ *output*-Synchronisation bedeutet: Ein Agent wird erst seine nächste Aufgabe nach t ausführen, nachdem alle Agenten schon die Aufgaben für t ausgeführt haben. Z.B. bevor t_{2a} schaltet, sind t_a, t_b ausgeführt.

Wenn das System Σ_2 nur dann korrekt ist, wenn die Kausalordnung (3-1-3) gilt⁶, dann muss diese Kausalordnung (3-1-3) bei der Verfeinerung erhaltenbleiben.

Im Gegensatz dazu hat die Kausalordnung (3-1-4) keinen Einfluß auf die Korrektheit dieses Systems, d.h. egal ob diese Kausalordnung gilt, bleibt die Korrektheit des Systems erhalten. Aber ohne diese Kausalordnung werden die Aktion t_b und t_{2a} nebenläufig zueinander.

Vogler's Verfeinerungsrelation \sqsubseteq_v verlangt, dass *alle* Kausal-Ordnungen der Aktionen in verteilten Abläufen des alten Systems erhalten bleiben. Dadurch kann man keine neue Nebenläufigkeit einführen. Wir wollen neue Nebenläufigkeiten einführen, deshalb müssen wir diese Beschränkung aufheben. Wir verlangen also nicht, dass alle Kausal-Ordnungen der Aktionen des alten Systems erhalten bleiben, sondern nur, dass alle Kausal-Ordnungen der Aktionen, die die Korrektheit garantieren, erhalten bleiben. So sind wir zu dem Begriff der verteilenden Verfeinerung gekommen. Bei unserer verteilenden Verfeinerung müssen nur die für die Korrektheit notwendigen Kausalitäten erhaltenbleiben. Wir können dann möglichst viel Nebenläufigkeiten erhalten. Durch das Einführen von Agenten- und Kanalstellen können wir außerdem konkretere Aussagen über zulässige Verfeinerungen machen (Abschnitt 5).

Das System Σ_3 in Abb. 3.1.11 ist eine verteilende Verfeinerung von Σ_1 (siehe Abb. 3.1.8) ohne Ausgabesynchronisation. Sie erhält wegen Satz 3.5.1 (siehe Abschnitt 3.5) alle Kausalitäten von Σ_1 .

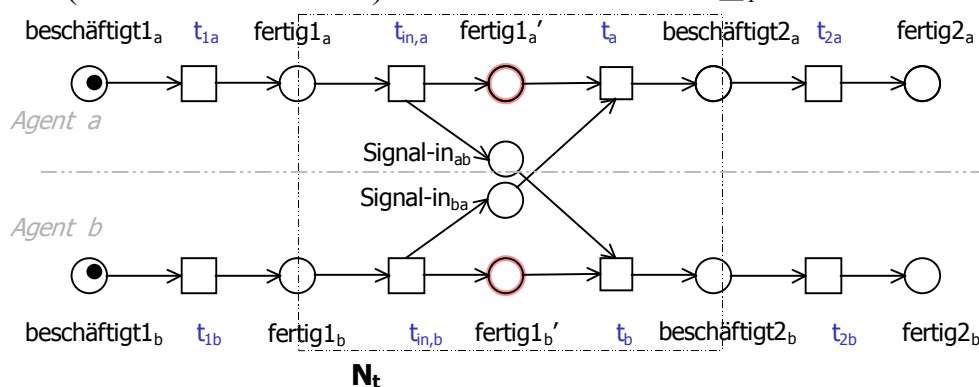


Abb. 3.1.11 System Σ_3 (eine weitere Verfeinerung von Σ_1)

⁶ In Σ_1 kann der Agent a erst seine Aufgabe t_a für t machen, wenn b die Aktion t_{1b} schon ausgeführt hat (d.h. $t_{1b} < t_a$), damit es nicht passiert, dass a schon mit der zweiten Phase aber b noch mit der ersten Phase beschäftigt ist.

3.2 Transitionsverfeinerung mit Blockbedingung

Wenn wir eine Transition in einem System verfeinern, dann sollen die uns interessierenden Eigenschaften des Systems erhalten bleiben. In der Arbeit [Vog87] (siehe vorigen Abschnitt) wurden sehr starke Anforderungen an die Verfeinerungsnetze gestellt. In der Arbeit [Peu01] wird dieser Begriff daher weiter verallgemeinert. Die strenge Anforderung an die Ersetzungsnetze wird durch eine Forderung an die Abläufe des Systems ersetzt.

In diesem Abschnitt wird der Begriff der Transitionsverfeinerung mit Blockbedingung angegeben. Zunächst wird der Begriff der Substitution in einem System definiert. Anschliessend wird der Begriff der Substitution in einem Ablauf definiert. Danach wird definiert, wann eine Substitution eines Systems die Blockbedingung erfüllt.

In Abb. 3.2.1 wird ein System Σ angegeben. Die Transition t wird zu einem Netz N_t verfeinert (Abb. 3.2.2).

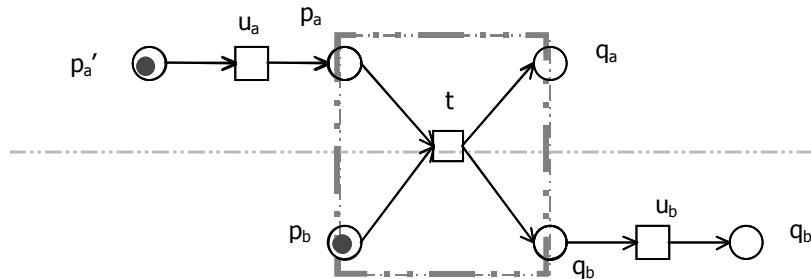
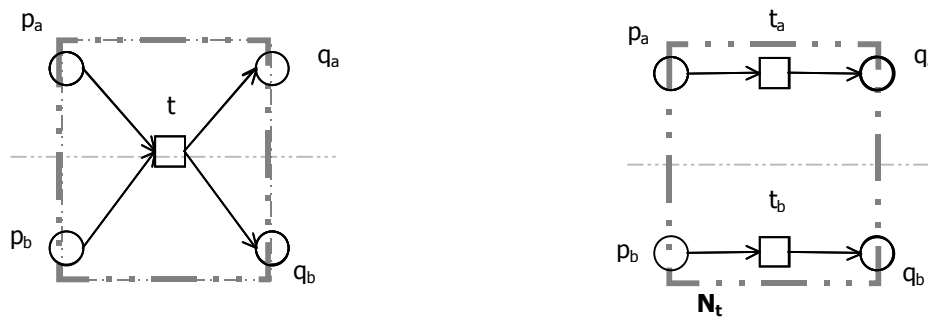


Abb. 3.2.1 System Σ



(a) Transition t (b) Ersetzungsnetz N_t

Abb. 3.2.2 Die Transition und ihr Ersetzungsnetz

Das Ersetzungsnetz N_t muss dabei die folgenden Bedingungen erfüllen:
 Zuerst muss N_t die gleichen Interface-Stellen $\bullet t \cup t \bullet$ mit der Umgebung wie die Transition t haben, und N_t enthält keine gleichnamigen Transitionen bzw. Stellen außer den Interface-Stellen wie das zu

verfeinernde System Σ .

Zweitens muss N_t sich so verhalten wie die Transition t , d.h. jeder Ablauf⁷ von (N_t, t^-) mit dem Anfangszustand t^- , endet im Zustand t^+ .

Definition 3.1 (Ersetzbarkeit, Ersetzungsnetz)

Sei Σ ein System, $t \in T$. Sei N_t ein Netz.

t ist ersetzbar durch N_t in Σ , wenn:

a) $P_{N_t} \cap P = \bullet t \cup t \bullet$, $T_{N_t} \cap T = \emptyset$ und $\bullet(\bullet t) = \emptyset$ in N_t .

b) Für jeden Ablauf (K_{N_t}, r_{N_t}) von (N_t, t^-) gilt: K_{N_t} ist endlich und $r_{N_t}(K_{N_t}^\circ) = t^+$.

Wir nennen N_t ein Ersetzungsnetz für t .

In unserem Beispiel hier ist t ersetzbar durch N_t .

Nun ersetzen wir die Transition t in Σ (siehe Abb.3.2.1) durch N_t (siehe Abb.3.2.2(b)) und erhalten das neue System Σ' (Abb.3.2.3).

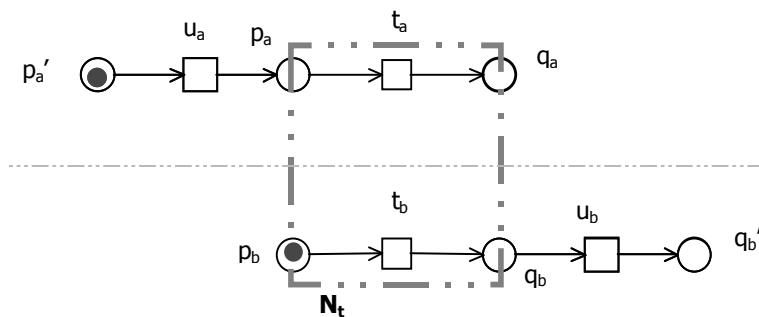


Abb. 3.2.3 Eine $t \rightarrow N_t$ Substitution Σ' von Σ

Definition 3.2.2 ($t \rightarrow N_t$ Substitution in einem System)

Sei $\Sigma = (N, m_0)$ ein System mit $N = (P, T, F)$, $t \in T$.

Sei $N_t = (P_{N_t}, T_{N_t}, F_{N_t})$ ein Netz mit: t ist ersetzbar durch N_t in Σ .

$\Sigma' =_{def} (N', m_0)$, wobei:

$N' =_{def} (P', T', F')$;

$P' =_{def} P \cup P_{N_t}$;

$T' =_{def} T \cup T_{N_t} \setminus \{t\}$;

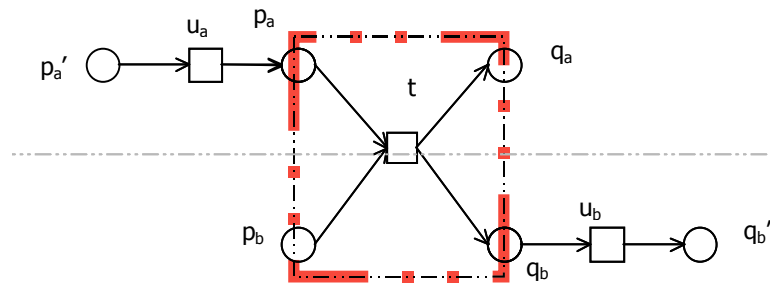
$F' =_{def} (F \cup F_{N_t}) \cap (P' \times T' \cup T' \times P')$.

Σ' heißt die $t \rightarrow N_t$ Substitution von Σ . Notation: $\Sigma' = \Sigma(N_t \setminus t)$.

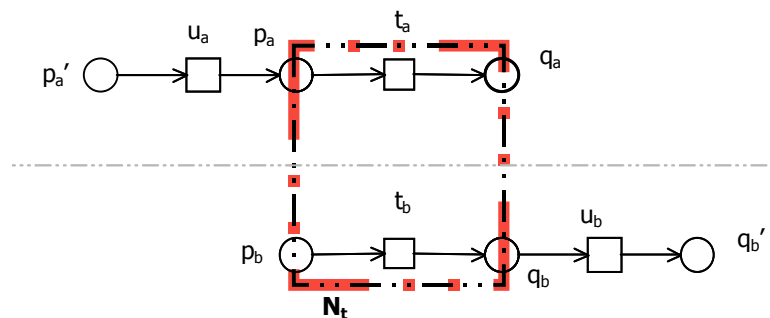
$\Sigma \rightarrow \Sigma'$: Die entsprechende Systemtransformation von Σ nach Σ' .

⁷ Weil wir meist nur verteilte Abläufe betrachten, nennen wir verteilte Abläufe auch kurz *Abläufe*.

In Abb. 3.2.4 sind die Abläufe K und K' jeweils von Σ und Σ' , wobei $\Sigma' = \Sigma(N_t \setminus t)$.



(a) Ablauf K von Σ (vor der Verfeinerung)



(b) Ablauf K' von Σ' (nach der Verfeinerung)

Abb. 3.2.4 Verteilte Abläufe von Σ und Σ' (Verfeinerungsschritt $\Sigma \rightarrow \Sigma'$)

Der Unterschied zwischen K und K' ist nur der folgende: Wo ein Auftreten der Transition t in K steht, steht in K' ein Auftreten des Ersatznetztes N_t , also ein Ablauf K_{N_t} von (N_t, t^-) . Wenn es bei jedem Ablauf von Σ' so ist, dann sagen wir, Σ' erfüllt die Blockbedingung.

Im Folgenden definieren wir die Blockbedingung mit Hilfe des Begriffes Ablauf-Substitution.

Die Ablauf-Substitution bedeutet folgendes: Sei K ein verteilter Ablauf eines Systems, das eine Transition t enthält. Wir ersetzen jedes Auftreten von t in K durch einen Ablauf von (N_t, t^-) . Der erhaltene Ablauf ist eine $t \rightarrow N_t$ -Substitution von K .

Die Ablauf-Substitution wird durch zwei Definitionen formuliert.

Definition 3.2.3 (*Passender Ablaufblock*)

Sei Σ ein System, sei $t \in T$ eine Transition und ersetzbar durch N_t in Σ .

Sei (K, r) ein Ablauf von Σ , sei e ein t -Auftreten in K .

Sei (K_{N_t}, r_{N_t}) ein Ablauf von (N_t, t^-) .

(K_{N_t}, r_{N_t}) heißt passend zu e in K , wenn:

a) $\bullet e \cup e^\bullet = {}^\circ K_{N_t} \cup K_{N_t}^\circ = K \cap K_{N_t}$

(beide Abläufe K und K_{N_t} haben genau die Bedingungen gemeinsam, die im Vor- und Nachbereich von e enthalten sind);

b) Für jedes $b \in \bullet e \cup e^\bullet$ gilt: $r(b) = r_{N_t}(b)$

(Diese Bedingungen entsprechen den gleichen Stellen aus N_t und damit aus $\bullet t \cup t^\bullet$). □

Definition 3.2.4 (*Ablauf-Substitution*)

Sei Σ ein System, sei $t \in T$ eine Transition und ersetzbar durch N_t in Σ .

Sei (K, r) ein Ablauf von Σ , wobei $K = (B, E, \leq)$.

$E(t) := \{ e \in E \mid r(e) = t \}$.

Für jedes $e \in E(t)$, sei (K_e, r_e) ein Ablauf von (N_t, t^-) und passend zu e in K , wobei $K_e = (B_e, E_e, \leq_e)$.

Für alle $e \in E(t)$, seien K_e paarweise disjunkt (keine Bedingungen und Ereignisse in den eingesetzten Abläufen gleich benannt).

$K' =_{\text{def}} (B', E', \leq')$, wobei:

$$B' =_{\text{def}} B \cup \left(\bigcup_{e \in E(t)} B_e \right),$$

$$E' =_{\text{def}} E \cup \left(\bigcup_{e \in E(t)} (E_e \setminus \{e\}) \right),$$

$$\leq' =_{\text{def}} \leq \cup \left(\bigcup_{e \in E(t)} (\leq_e \setminus (\bullet e \times \{e\} \cup \{e\} \times e^\bullet)) \right).$$

(K', r') heißt eine $t \rightarrow N_t$ Substitution von (K, r) .

K_e heißt ein Auftreten von N_t in K' .

In Abb. 3.2.4(b) ist K' eine $t \rightarrow N_t$ Ablauf-Substitution von K .

Bedingungen und Ereignisse aus K sind auch Bedingungen und Ereignisse von K' . Die restlichen Bedingungen und Ereignisse aus K' sind Auftreten von Stellen bzw. Transitionen des Ersetzungsnetzes N_t .

Bemerkung 3.2.5

Sei $\Sigma = ((P, T, F), m_0)$ ein System, sei $t \in T$ ersetzbar durch N_t in Σ , wobei $N_t = (P_{N_t}, T_{N_t}, F_{N_t})$.

Sei (K, r) ein Ablauf von Σ , wobei $K = (B, E, \prec)$.

Sei (K', r') eine $t \rightarrow N_t$ Substitution von (K, r) , wobei $K' = (B', E', \prec')$.

Dann gilt:

- i. Für jedes $s \in B \cup E$ mit $r(s) \neq t$ gelten $s \in B' \cup E'$ und $r'(s) = r(s)$.
- ii. Für jedes $s \in (B' \cup E') \setminus (B \cup E)$ gilt $r'(s) \in P_{N_t} \cup T_{N_t}$.

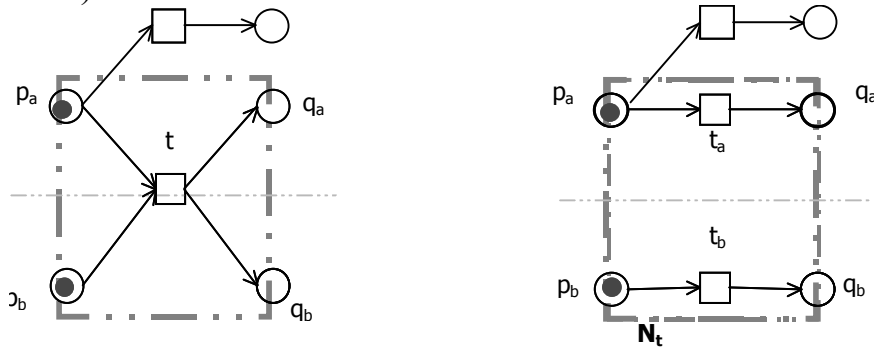
Definition 3.2.6 (Block-Bedingung)

Sei $\Sigma' = \Sigma(N_t \setminus t)$.

- i. Ein Ablauf (K', r') von Σ' erfüllt die Block-Bedingung, gdw. es einen Ablauf (K, r) von Σ gibt, so dass (K', r') eine $t \rightarrow N_t$ Substitution von (K, r) ist.
- ii. Σ' erfüllt die Block-Bedingung, gdw. jeder Ablauf von Σ' die Blockbedingung erfüllt.

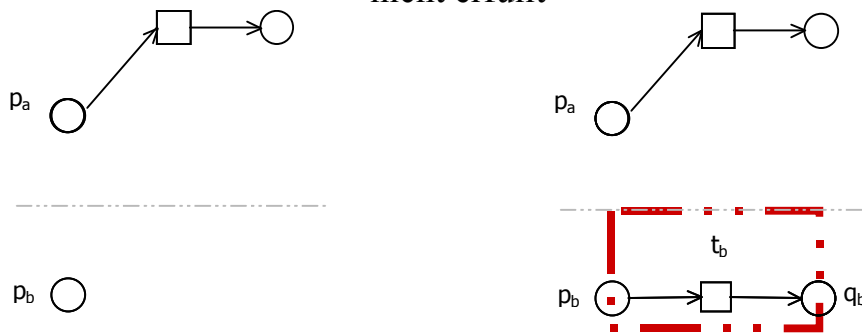
Im Beispiel (siehe Abb. 3.2.3) erfüllt die Transitionsverfeinerung von t die Blockbedingung.

Ein typisches Beispiel für eine Transitionsverfeinerung, die nicht die Blockbedingung erfüllt, zeigen die folgenden Abbildungen (Abb. 3.2.5, Abb. 3.2.6):



(a) System Σ (vor der Verfeinerung) (b) System Σ' (nach der Verfeinerung)

Abb. 3.2.5 Ein Verfeinerungsschritt ($\Sigma \rightarrow \Sigma'$), der die Blockbedingung nicht erfüllt



(a) Ablauf K von Σ (b) Ablauf K' von Σ'
(vor der Verfeinerung) (nach der Verfeinerung)

Abb. 3.2.6 Ablauf K' von Σ' , der nicht die Blockbedingung erfüllt

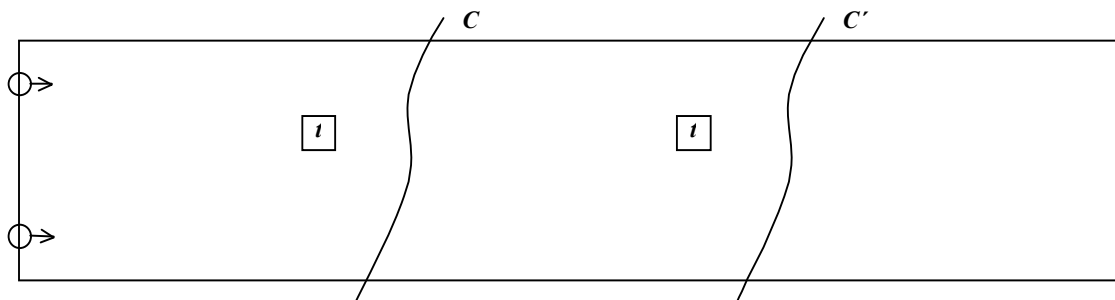
Die Arbeit von S. Peuker (vgl. [Peu01]) klärt nicht den Zusammenhang zwischen Eigenschaften des Verfeinerungsnetzes und dem Erfülltsein der Blockbedingung. In Kapitel 6 werden wir sehen, wie dieser Zusammenhang für die hier definierten Agentennetze beschrieben werden kann.

3.3 Eigenschaften der Blockbedingung

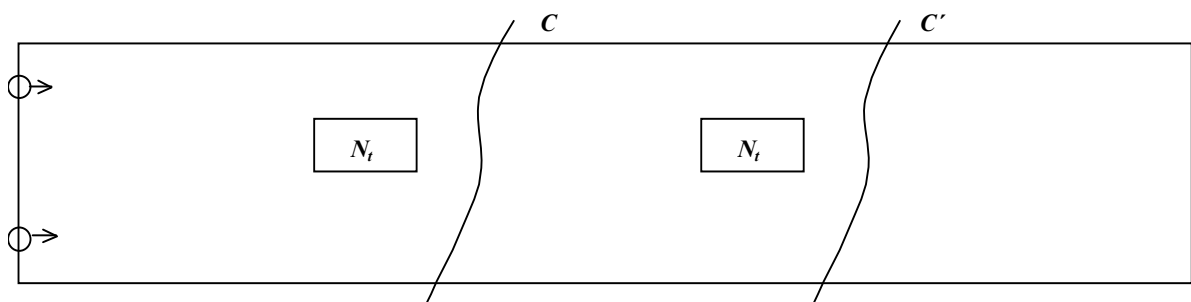
In diesem Abschnitt betrachten wir, welche temporal logische Eigenschaften bei einem Verfeinerungsschritt erhalten bleiben, wenn die Blockbedingung erfüllt ist. Der Abschnitt fasst die wichtigen Aussagen zur Blockbedingung aus [Peu01] zusammen.

In Abb. 3.3.1 (a) wird ein Ablauf K eines Systems vor der Verfeinerung illustriert, wobei \boxed{t} für in Auftreten von t in K steht und C, C' für Schnitte in K stehen. In Abb. 3.3.1 (b) wird ein Ablauf K' des neuen Systems nach der Verfeinerung illustriert, der K entspricht (d.h. K' ist

eine $t \rightarrow N_t$ Substitution von K), wobei $\boxed{N_t}$ für ein Auftreten von N_t in K' steht.



(a) Veranschaulichung des Auftretens von t in einem Ablauf K von Σ und von Schnitten C und C' mit $C < C'$



(b) Veranschaulichung des Auftretens von Abläufen von (N_t, t^-) in dem zugeordneten Ablauf K' von Σ' und der Schnitte C und C'

Abb. 3.3.1 Veranschaulichung einer $t \rightarrow N_t$ Substitution in einem Ablauf K

Die Blockbedingung sichert folgende Eigenschaften:

- Jeder Schnitt von K ist auch ein Schnitt von K' . In Abb. 3.3.1 ist der

- Schnitt C in K auch ein Schnitt in K' .
- Wenn ein Schnitt C in (K, r) auf eine Markierung m abgebildet wird, dann wird C in (K', r') auch auf die gleiche Markierung m abgebildet, d.h.
 $r(C) = r'(C)$.
 - Wenn in K gilt: C' ist von C erreichbar, dann gilt in K' auch: C' ist von C erreichbar.

Satz 3.3.1

Erfülle $\Sigma' = \Sigma(N_t \setminus t)$ die Blockbedingung.

Seien (K, r) , (K', r') jeweils Abläufe von Σ, Σ' mit: (K', r') ist eine $t \rightarrow N_t$ Substitution von (K, r) .

Dann gilt:

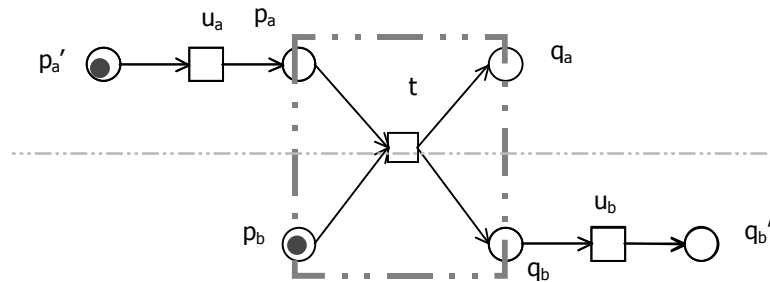
- i. Jeder Schnitt C von K ist auch ein Schnitt von K' und es gilt $r(C) = r'(C)$.*
- ii. Für zwei Schnitte C, C' von K gilt: wenn C' in K erreichbar von C ist, dann ist C' auch in K' erreichbar von C .*

Aus den oben genannten Eigenschaften, die die Blockbedingung erhält, kann man weitere Eigenschaften ableiten, die die Blockbedingung erhält.

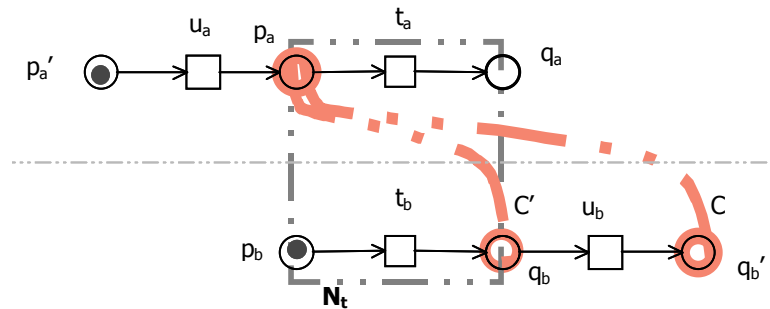
- $\diamond \varphi$ (d.h. in jedem Ablauf gibt es einen Zustand, in dem φ gilt);
 - $\square \diamond \varphi$ (d.h. in jedem Ablauf gibt es immer wieder einen Zustand, in dem φ gilt),
- wobei φ eine Zustandsaussage ist.

3.4 Was bleibt nicht erhalten

Nun fragen wir uns: Welche Eigenschaften können in einem Verfeinerungsschritt verloren gehen, auch wenn die Blockbedingung erfüllt ist? Dafür sehen wir uns nochmals das Beispiel im vorherigen Abschnitt an (siehe Abb. 3.4.1).



(a) System Σ (vor der Verfeinerung)



(b) System Σ' (nach der Verfeinerung)

Abb. 3.4.1 Verfeinerungsschritt $\Sigma \rightarrow \Sigma'$

Lebendigkeitseigenschaft:

Im System Σ vor der Verfeinerung gilt offensichtlich die folgende Eigenschaft:

$$p_a \triangleright q_b \quad (L)$$

(d.h. in jedem Ablauf K von Σ gilt: Für jeden Schnitt C in K , in dem p_a gilt, gibt es einen von C erreichbaren Schnitt C' , in dem q_b gilt.)

Im System Σ' nach der Verfeinerung (siehe Abb. 3.4.1(b)) wird der Zustand C erreichbar, in dem p_a, q_b' gelten. Dieser Zustand ist ein Zustand, in dem p_a gilt, aber von diesem Zustand aus kann man keinen Zustand erreichen, in dem q_b gilt. D.h. in Σ' gilt die Eigenschaft (L) nicht mehr. Also, im Verfeinerungsschritt $\Sigma \rightarrow \Sigma'$ bleibt die Lebendigkeitseigenschaft (L) nicht erhalten.

Sicherheitseigenschaft:

Im System Σ vor der Verfeinerung (siehe Abb. 3.4.1(a)) gilt offensichtlich die folgende Eigenschaft:

$$\square \neg (p_a \wedge q_b) \quad (\text{S})$$

(d.h. es gilt immer: die Stelle p_a und die Stelle q_b sind nicht gleichzeitig belegt.)

Im System Σ' nach der Verfeinerung (siehe Abb. 3.4.1(b)) wird der Zustand C' erreichbar, in dem p_a, q_b gelten. D.h. in Σ' gilt auch die Eigenschaft (S) nicht mehr. Also, in Verfeinerungsschritt $\Sigma \rightarrow \Sigma'$ bleibt die Sicherheitseigenschaft (S) nicht erhalten.

Wie in Abb. 3.4.2 dargestellt, brauchen wir nur die Kausalbedingung $t_a \blacktriangleleft t_b$ im lokalen System (N_t, t^-) zu erfüllen, dann bleiben sowohl die Lebendigkeitseigenschaft (L) als auch die Sicherheitseigenschaft (S) erhalten. Das ist die Verfeinerung mit Kausalitäten, die wir im nächsten Abschnitt vorstellen werden.

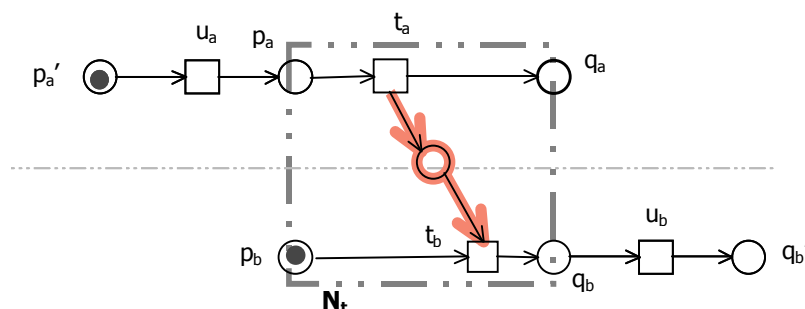


Abb. 3.4.2 Eine weitere Verfeinerung Σ'' von Σ' (aus Abb. 3.4.1(a))

Zusätzliche Kausalitätsbedingungen im Verfeinerungsnetz verringern die Anzahl der im verfeinerten System zusätzlich erreichbaren Zustände. Auf diesem Zusammenhang basiert die Bedeutung von Kausalitätsbedingungen für Verfeinerungsnetze.

3.5 Erhaltenbleiben der Kausalitäten in einem System

3.5.1 Erhaltenbleiben der Kausalordnung

Die Blockbedingung sichert nur die Kausalordnung zwischen Schnitten (siehe Satz 3.3.1 im Abschnitt 3.3). Die Kausalordnung zwischen Ereignissen und Bedingungen kann verändert werden. Im vorherigen Beispiel im Abschnitt 3.2 gilt im Ablauf K von Σ vor der Verfeinerung $u_a < u_b$ (siehe Abb. 3.2.4). Im entsprechenden Ablauf K' der Verfeinerung Σ' gilt $u_a \underline{co} u_b$.

Wir wollen zeigen, dass alle Kausalitäten in Σ bei einer Transitionsverfeinerung erhalten bleiben, wenn das Verfeinerungsnetz (N_b, t^-) die Kausalität $\bullet t \blacktriangleleft t \bullet$ erfüllt. Dabei gilt $\bullet t \blacktriangleleft t \bullet$ in (N_b, t^-) , wenn für jedes $p \in \bullet t$ und $q \in t \bullet$ in (N_b, t^-) $p \blacktriangleleft q$ gilt.

Notation ($\bullet t \blacktriangleleft t \bullet$)

Sei $\Sigma = ((P, T, F), m_0)$ ein System, sei $t \in T$ ersetzbar durch N_t .

$(N_b, t^-) \models \bullet t \blacktriangleleft t \bullet$ gdw.

$$\forall p \in \bullet t \text{ und } \forall q \in t \bullet \text{ gilt } (N_b, t^-) \models p \blacktriangleleft q.$$

Wir betrachten jetzt die Abläufe von Σ und die von $\Sigma' = \Sigma(N_t \setminus t)$ etwas genauer. Sei (K, r) ein Ablauf von Σ , in dem t auftritt und für Ereignisse oder Bedingungen e, f, g, h durch Linien angedeutete Kausalitäten gelten.

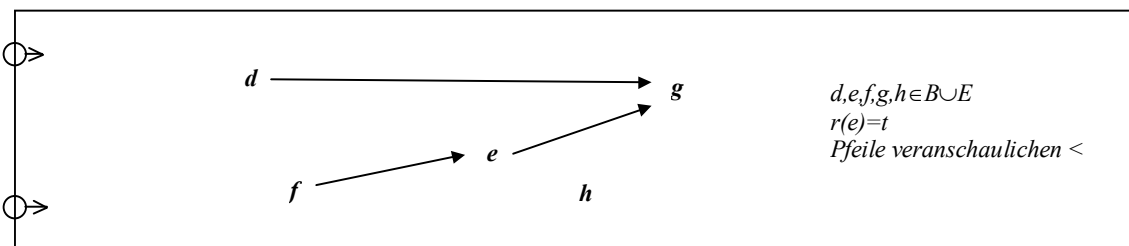


Abb. 3.5.1 Illustration für einen Ablauf (K, r) von Σ

Offensichtlich ist, dass nach der Ersetzung von e durch einen Ablauf von (N_b, t^-) keine neuen kausalen Abhängigkeiten entstehen. Es gilt also weiter $f \underline{co} h$. Auch die Kausalität $d < g$ bleibt erhalten, weil der Weg von d nach g erhalten bleibt. Genauer zu betrachten ist also nur der Fall $f < e < g$

mit $r(e)=t$. Wie wir im vorangegangenen Beispiel gesehen haben, können durch Beseitigung der Synchronisation bei der verteilenden Verfeinerung aufeinanderfolgende Ereignisse nebenläufig werden. Das wird gerade durch $\bullet t \blacktriangleleft t \bullet$ verhindert.

Die folgende Abbildung veranschaulicht diesen Zusammenhang:

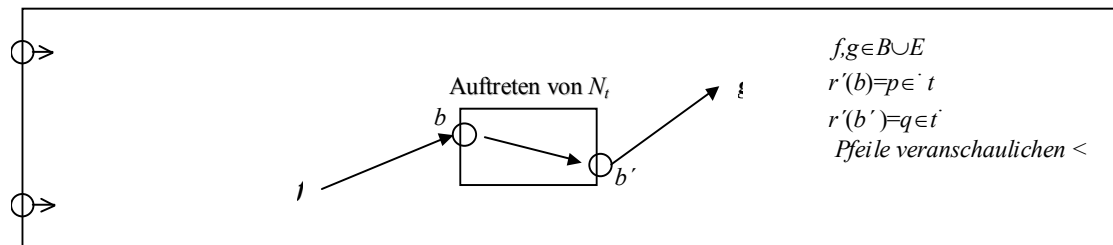


Abb. 3.5.2 Illustration für einen Ablauf (K', r') von Σ' mit $\bullet t \blacktriangleleft t \bullet$

Weil in (K', r') $f < e < g$ mit $r(e)=t$ gilt, hat also t zwischen f und g geschaltet. Daher muss für ein $p \in \bullet t$ und $q \in t \bullet$ gelten, dass $f \leq b$ mit $r'(b)=p$ und $p \in \bullet t$ und $e < b' \leq g$ mit $r'(b')=q$ und $q \in t \bullet$, also $f \leq b$ und $b' \leq g$. Wegen $\bullet t < t \bullet$ in K' gilt $b < b'$ also $f \leq b < b' \leq g$ also auch $f < g$.

Satz 3.5.1 (Erhaltenbleiben der Kausalordnung zwischen Bedingungen und Ereignissen)

Erfülle $\Sigma' = \Sigma(N_t \setminus t)$ die Blockbedingung.

Seien $(K, r), (K', r')$ jeweils Abläufe von Σ, Σ' mit: (K', r') ist eine $t \rightarrow N_t$ Substitution von (K, r) .

Gilt $(N_t, t^-) \models \bullet t \blacktriangleleft t \bullet$, so bleibt die Kausalitätsrelation $<$ von K zwischen Bedingungen und Ereignissen in K' erhalten (d.h. $\forall s, t \in B \cup E: s <^+ t \rightarrow s <'^+ t$).

Beweis:

1. Sei e Ereignis mit $r(e)=t$, so gilt in K $\bullet e < e \bullet$. Wegen $(N_t, t^-) \models \bullet t \blacktriangleleft t \bullet$ bleibt diese Relation erhalten. Es gilt weiter, sei $b \in \bullet e$ und $b' \in e \bullet$, so gilt $b < b'$.
2. Seien $g, f \in B \cup E$ mit $g < f$ und für jedes $e \in E$ mit $g < e < f$ gilt $r(e) \neq t$ (kein Auftreten von t liegt zwischen f und g), dann bleibt offensichtlich die Kausalitätsrelation $g < f$ erhalten.

3. Seien $g, f \in B \cup E$ mit $g < f$ und es existiert ein $e \in E$ mit $r(e) = t$ und $g \leq e \leq f$. Dann gilt für ein $b \in \bullet e$ und ein $b' \in e \bullet$ in K' $g \leq b < b' \leq f$ (wegen 1.). Also bleibt $g < f$ erhalten.

4. Seien $g, f \in B \cup E$ mit $\neg(g < f)$, dann gilt auch in K' $\neg(g < f)$. □

Das System Σ'' in Abb. 3.5.3 ist eine weitere Verfeinerung von Σ (siehe Abb. 3.2.1).

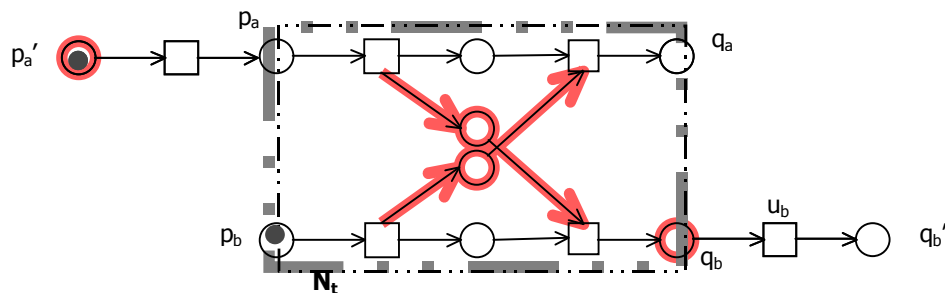
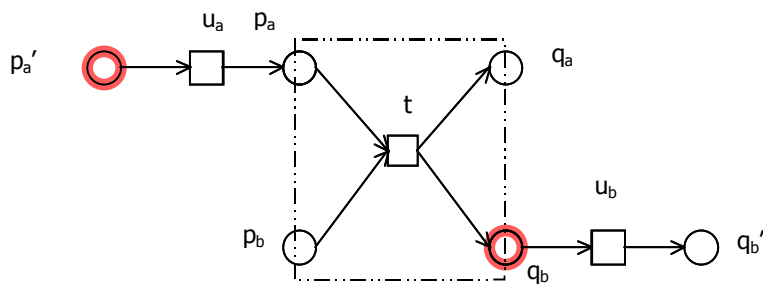
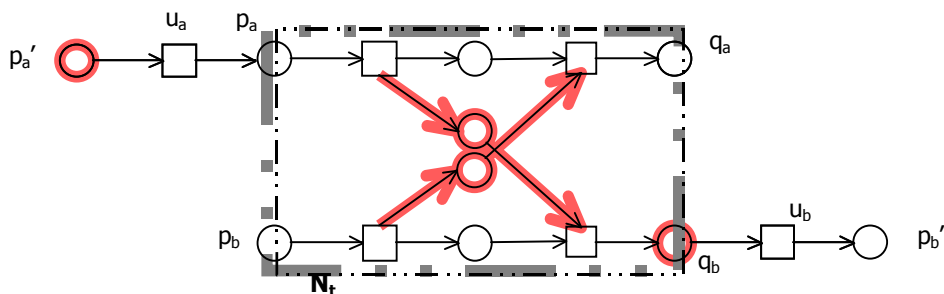


Abb. 3.5.3 Eine weitere Verfeinerung Σ'' von Σ (siehe Abb. 3.2.1)

In Abb. 3.5.4 werden die Abläufe der Systeme Σ und Σ'' (also vor und nach der Verfeinerung) angegeben.



(a) Ablauf K von Σ



(b) Ablauf K'' von Σ''

Abb. 3.5.4 Abläufe von Σ und Σ''

Im Verfeinerungsschritt $\Sigma \rightarrow \Sigma''$ bleiben alle kausalen Ordnungen von K

in K' erhalten, weil im Ersetzungsnetz die Kausalität $\bullet t \blacktriangleleft t \bullet$ gilt, d.h. $(N_t, t^-) \models \bullet t \blacktriangleleft t \bullet$.

Aus dem Satz 3.5.1 können wir ableiten: Für jeden Schnitt C' in einem Ablauf K' des neuen Systems Σ' nach der Verfeinerung gibt es einen Schnitt C in dem entsprechenden Ablauf K des alten Systems Σ vor der Verfeinerung, so dass jede Stelle aus P (Stellenmenge von Σ), die in C' markiert ist, auch in C markiert ist.

In Abb. 3.5.5 ist der Ablauf K'' von Σ'' (siehe Abb. 3.5.3) nochmal angegeben. C' ist ein Schnitt von K'' . In C' ist aus P nur die Stelle p_a' markiert. Der Schnitt C von K'' , in dem p_a' auch markiert ist, ist auch ein Schnitt von K .

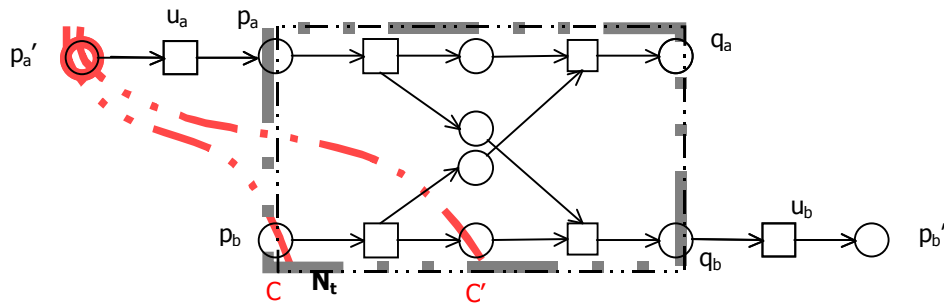


Abb. 3.5.5 Ablauf K'' von Σ'' (Illustration für die Folgerung 3.5.2)

Folgerung 3.5.2

Erfülle $\Sigma' = \Sigma(N_t \setminus t)$ die Blockbedingung, wobei: $N_t = (P_{N_t}, T_{N_t}, F_{N_t})$.

Seien (K, r) , (K', r') jeweils Abläufe von Σ , Σ' mit: (K', r') ist eine $t \rightarrow N_t$ Substitution von (K, r) .

Sei C' ein Schnitt von K' , $C_1 := \{ b \in C' \mid r'(b) \in P_{N_t} \setminus (\bullet t \cup t \bullet) \}$.

Wenn $(N_t, t^-) \models \bullet t \blacktriangleleft t \bullet$ gilt, dann gibt es einen Schnitt C von K mit $C' \setminus C_1 \subseteq C$.

Beweis:

Wir führen den Beweis indirekt, angenommen, es gibt keinen Schnitt C von K mit $C' \setminus C_1 \subseteq C$.

Dann gibt es zwei Bedingungen b, b' mit: $b, b' \in C' \setminus C_1$ und in K gilt $\neg(b \leq b')$, d.h. in K gilt entweder $b < b'$ oder $b' < b$.

Aus dem Satz 3.5.1 folgt: Auch in K' gilt entweder $b < b'$ oder $b' < b$.

Das ist ein Widerspruch dazu, dass in $K' \quad b \underline{\text{co}} b'$ gilt (da $b, b' \in C'$).
Daraus folgt die Behauptung. \square

Aus dem Satz 3.5.1 können wir die folgende Bemerkung (Bem. 3.5.3) erhalten. Diese Bemerkung impliziert: Wenn eine Transition t_2' im System Σ vor der Verfeinerung eine nicht aktivierte Konflikt-Transition von t_2 ist (d.h. $\bullet t_2 \cap \bullet t_2' \neq \emptyset$ und $\square (\bullet t_2 \rightarrow \neg \bullet t_2')$), dann ist t_2' auch im System $\Sigma' = \Sigma(N_t \setminus t)$ nach der Verfeinerung eine nicht aktivierte Konflikt-Transition von t_2 .

Beim Nachweis der Blockbedingung müssen wir oft zeigen, dass eine konkurrente Transition nicht aktiviert ist (vgl. Abschnitt 6.2-6.4). Bei der Vertauschbarkeit der Verfeinerungsschritte (vgl. Abschnitt 6.5) müssen wir dann entsprechend zeigen, dass eine vor einer Verfeinerung nicht aktivierte konkurrente Transition nach der Verfeinerung auch nicht aktiviert ist. Dort verwenden wir oft Bemerkung 3.5.3.

Bemerkung 3.5.3

$\Sigma' = \Sigma(N_t \setminus t)$ erfülle die Blockbedingung. Seien $p_1, \dots, p_n, q \in P$.

Gilt $(N_b, t^-) \models \bullet t \blacktriangleleft t \bullet$, so bleibt beim Verfeinerungsschritt $(\Sigma \rightarrow \Sigma')$ die Sicherheitseigenschaft $\square (p_1 \wedge \dots \wedge p_n \rightarrow \neg q)$ erhalten.

Beweis:

Angenommen, $\Sigma \models \phi$, wobei $\phi = \square (p_1 \wedge \dots \wedge p_n \rightarrow \neg q)$ (1)

Zu zeigen ist: $\Sigma' \models \phi$. (2)

Seien $(K, r), (K', r')$ Abläufe von Σ, Σ' mit: (K', r') ist eine $t \rightarrow N_t$

Substitution von (K, r) , wobei $K = (B, E, \triangleleft), K' = (B', E', \triangleleft')$.

Sei C ein Schnitt von K' , in dem $p_1 \wedge \dots \wedge p_n$ gilt.

Seien $b_1, \dots, b_n \in C$ mit $r'(b_k) = p_k, k=1, \dots, n$.

Weiter sei $b \in B'$ ein beliebiges Auftreten von q in K' , d.h. $r'(b) = q$.

Wir brauchen nur zu zeigen,

dass es ein $b_k \in \{b_1, \dots, b_n\}$ gibt, so dass in $K' \quad \neg (b_k \underline{\text{co}} b)$ gilt. (3)

Wegen Bem. 3.2.5 und Satz 3.3.1 gilt:

C ist auch ein Schnitt von K und

$b_1, \dots, b_n \in C$ und $r(b_k) = r'(b_k) = p_k, k=1, \dots, n$ und

$b \in B$ und $r(b) = r'(b) = q$.

Weil ϕ in K gilt, (wegen (1))

gibt es ein $b_k \in \{b_1, \dots, b_n\}$, so dass in $K \quad \neg (b_k \underline{\text{co}} b)$ gilt.

Dann gilt: entweder $b_k \leq^+ b$ oder $b \leq^+ b_k$.

In K' gilt weiter:

$$b_k \leq'^+ b \quad \text{falls } b_k \leq^+ b;$$

$$b \leq'^+ b_k \quad \text{falls } b \leq^+ b_k. \quad (\text{wegen Satz 3.5.1})$$

Also, in K' gilt $\neg (b_k \underline{\text{co}} b)$.

Daraus folgt: (3) gilt. □

3.5.2 Erhaltenbleiben der Kausalitäten im System

Im Verfeinerungsschritt $\Sigma \rightarrow \Sigma''$ in unserem Beispiel (siehe Abb. 3.5.3) bleibt die Kausalität $u_a \blacktriangleleft u_b$ im System erhalten. Im Gegensatz dazu bleibt diese Kausalität im Verfeinerungsschritt $\Sigma \rightarrow \Sigma'$ (siehe Abb. 3.4.1) nicht erhalten.

Satz 3.5.4 (Erhaltenbleiben von Kausalitäten zwischen Stellen und Transitionen bei der verteilenden Verfeinerung)

$\Sigma' = \Sigma(N_t \setminus t)$ erfülle die Blockbedingung.

Gilt $(N_t, t^-) \models \bullet t \blacktriangleleft t^*$, so gilt die Relation \blacktriangleleft zwischen Stellen und Transitionen des Systems Σ auch im System Σ' .

Beweis:

Die Definition der Relation \blacktriangleleft im System basiert alleine auf den Kausalitäten in den Abläufen der Systeme. Nach Satz 3.5.1 bleiben alle Kausalitäten in den Abläufen erhalten, falls $\bullet t \blacktriangleleft t^*$ für die Verfeinerung gilt. □

Die folgende Bemerkung besagt, wenn für zwei Transition t_a, t_b in einem Ersetzungsnetz N_t für t gilt, dass im Teilsystem $(N_t, t^-) \quad t_a \blacktriangleleft t_b$ gilt, dann gilt $t_a \blacktriangleleft t_b$ auch im System $\Sigma' = \Sigma(N_t \setminus t)$. Für das Beispiel Σ'' im Abschnitt 3.4 (siehe Abb. 3.4.2) gilt im Teilsystem die Kausalität $t_a \blacktriangleleft t_b$. Deshalb gilt diese Kausalität auch im System Σ'' .

Bemerkung 3.5.5

Sei $\Sigma = ((P, T, F), m_0)$ ein System, sei $t \in T$ ersetzbar durch N_b , wobei $N_t = (P_{N_t}, T_{N_t}, F_{N_t})$. Weiter seien $t_a, t_b \in T_{N_t}$.

Erfülle $\Sigma' = \Sigma(N_t \setminus t)$ die Blockbedingung.

Wenn $(N_b, t^-) \models t_a \blacktriangleleft t_b$ gilt, dann gilt $\Sigma' \models t_a \blacktriangleleft t_b$.

Beweis:

Sei (K', r') ein Ablauf von Σ' , wobei $K' = (B', E', \prec')$.

Sei $e_{t_b} \in E'$ mit $r'(e_{t_b}) = t_b$.

Dann gibt es ein Auftreten K_{N_t} von N_t in K' mit $K_{N_t} = (B_{N_t}, E_{N_t}, \prec_{N_t})$ und $e_{t_b} \in E_{N_t}$.

Wegen $(N_b, t^-) \models t_a \blacktriangleleft t_b$ gilt:

Es gibt ein $e_{t_a} \in E_{N_t}$ mit: $r'(e_{t_a}) = t_a$ und $e_{t_a} \prec'^+ e_{t_b}$ und für jedes $e \in E'$ mit $e_{t_a} \prec'^+ e \prec'^+ e_{t_b}$ gilt $r'(e) \neq t_b$.

Daraus folgt: In K' gilt $t_a \blacktriangleleft t_b$.

Dann gilt $\Sigma' \models t_a \blacktriangleleft t_b$. □

Im Folgenden geben wir eine Bemerkung über die Gültigkeit einer Kausalität im System an, die wir im Kapitel 7 (vgl. Abschnitt 7.2.2) direkt benutzen können.

Bemerkung 3.5.6

Sei $\Sigma = ((P, T, F), m_0)$ ein System, sei $t \in T$ ersetzbar durch N_b , wobei $N_t = (P_{N_t}, T_{N_t}, F_{N_t})$, sei N_t ein Kausalnetz.

Weiter seien $p \in \bullet t$, $t_b \in T_{N_t}$, $t_a \in T \setminus \{t\}$.

Erfülle $\Sigma' = \Sigma(N_t \setminus t)$ die Blockbedingung.

Wenn $(N_b, t^-) \models p \blacktriangleleft t_b$ und $\Sigma' \models t_a \blacktriangleleft p$ gelten, dann gilt $\Sigma' \models t_a \blacktriangleleft t_b$.

Beweis:

Sei (K', r') ein Ablauf von Σ' , wobei $K' = (B', E', \prec')$.

Sei $e_{t_b} \in E'$ mit $r'(e_{t_b}) = t_b$.

Dann gibt es ein Auftreten K_{N_t} von N_t in K' mit $K_{N_t} = (B_{N_t}, E_{N_t}, \prec_{N_t})$ und $e_{t_b} \in E_{N_t}$.

Wegen $(N_b, t^-) \models p \blacktriangleleft t_b$ und weil N_t ein Kausalnetz ist, gilt:

Es gibt ein $b_p \in {}^\circ K_{N_t}$ mit: $r'(b_p) = p$ und $b_p \ll'^+ e_{t_b}$ und für jedes $e \in E'$ mit $b_p \ll'^+ e \ll'^+ e_{t_b}$ gilt $r'(e) \neq t_b$. (1)

Wegen $\Sigma' \models t_a \blacktriangleleft p$ gilt:

Es gibt ein $e_{t_a} \in E'$ mit: $r'(e_{t_a}) = t_a$ und $e_{t_a} \ll'^+ b_p$ und für jedes $b \in B'$ mit $e_{t_a} \ll'^+ b \ll'^+ b_p$ gilt $r'(b) \neq p$. (2)

Aus (1), (2) folgt: In K' gilt $t_a \blacktriangleleft t_b$.

Daraus folgt: $\Sigma' \models t_a \blacktriangleleft t_b$. □

3.5.3 Erhaltenbleiben von Eigenschaften durch Kausalitäten

Im Folgenden geben wir ein Lemma über das Erhaltenbleiben von Lebendigkeitseigenschaften an. Vorher geben wir noch eine Bemerkung an, die wir zum Beweis benutzen werden.

Im Verfeinerungsschritt $\Sigma \rightarrow \Sigma''$ in unserem Beispiel (siehe Abb. 3.5.3) bleibt die Lebendigkeitseigenschaft $p_a' \triangleright q_b$ erhalten. Im Gegensatz dazu bleibt diese Eigenschaft im Verfeinerungsschritt $\Sigma \rightarrow \Sigma'$ (siehe Abb. 3.4.1) nicht erhalten.

Notation 3.5.7 (vgl. [Rei87])

Sei $K = (B, E, \ll)$ ein Kausalnetz, seien $S, S' \subseteq B$ co-Mengen von K .
 $S \text{ [co } S' \text{ gdw.}$

Für jedes $s \in S$ und $s' \in S'$ gilt: $s \ll^+ s'$ oder $s \underline{\text{co}} s'$.

Der Einfachheit halber schreiben wir auch $S \text{ [co } b$ anstelle $S \text{ [co } \{b\}$, wobei $S \subseteq B$, $b \in B$.

Bemerkung 3.5.8

Sei $\Sigma = ((P, T, F), m_0)$ ein System, $p, q \in P$.

Sei (K, r) ein Ablauf von Σ , wobei $K = (B, E, \ll)$.

In (K, r) gilt $p \triangleright q$, gdw.

für jedes $b_p \in B$ mit $r(b_p) = p$ gibt es ein $b_q \in B$ mit $r(b_q) = q$ mit:

entweder $b_q \in K^\circ$;

oder $b_p \ll^+ b_q'$ wobei $b_q' \in B$ mit $b_q \ll e \ll b_q'$ für ein $e \in E$.

Beweis:

(\leftarrow):

Angenommen, für jedes Auftreten b_p von p gilt: Es gibt ein Auftreten b_q von q mit: entweder $b_q \in K^\circ$ oder $b_p \leq^+ b_q'$, wobei $b_q' \in B$ mit $b_q \leq e \leq b_q'$ für ein $e \in E$. (1)

Zu zeigen ist: Für jeden Schnitt C , in dem p gilt, gibt es einen von C erreichbaren Schnitt C' , in dem q gilt. (2)

Sei $b_p \in B$ ein beliebiges Auftreten von p , also $r(b_p) = p$.

Sei C ein Schnitt mit $b_p \in C$.

Wegen (1) gilt:

Es gibt ein $b_q \in B$ mit $r(b_q) = q$ und entweder $b_q \in K^\circ$ oder $b_p \leq^+ b_q'$, wobei $b_q \leq e \leq b_q'$ für ein $e \in E$.

Fall 1. $b_q \in K^\circ$.

Dann gilt q in K° .

Offensichtlich gilt $C \rightarrow^* K^\circ$.

Daraus folgt: (2) gilt.

Fall 2. $b_p \leq^+ b_q'$, wobei $b_q \leq e \leq b_q'$ für ein $e \in E$.

Dann gilt $C \text{ [co } b_q$.

Daraus folgt: Es gibt einen von C erreichbaren Schnitt C' , der b_q enthält.

Daraus folgt: (2) gilt.

(\rightarrow):

Angenommen, für jeden Schnitt C , in dem p gilt, gibt es einen von C erreichbaren Schnitt C' , in dem q gilt. (3)

Zu zeigen ist: Für jedes Auftreten b_p von p gilt: Es gibt ein Auftreten b_q von q mit: entweder $b_q \in K^\circ$ oder $b_p \leq^+ b_q'$, wobei $b_q' \in B$ mit $b_q \leq e \leq b_q'$ für ein $e \in E$. (4)

Sei $b_p \in B$ ein Auftreten von p , d.h. $r(b_p) = p$.

Sei C ein Schnitt mit $b_p \in C$.

Wegen (3) gilt:

Es gibt einen von C erreichbaren Schnitt C' , in dem q gilt.

D.h. C' enthält ein Auftreten b_q von q , also $r(b_q) = q$.

Wegen $C \rightarrow^* C'$ gilt: $C \text{ [co } b_q$.

D.h. es gibt ein Auftreten b_q von q mit $C \text{ [co } b_q$.

Falls $b_q \notin K^\circ$:

Sei $b_q' \in B$ mit $b_q \leq e \leq b_q'$ für ein $e \in E$.

Dann gilt $b_p \leq^+ b_{q'}$.

Also, es gibt ein Auftreten b_q von q mit: entweder $b_q \in K^\circ$ oder $b_p \leq^+ b_{q'}$,
wobei $b_{q'} \in B$ mit $b_q \leq e \leq b_{q'}$ für ein $e \in E$.

Daraus folgt: (4) gilt. □

Lemma 3.5.9

$\Sigma' = \Sigma(N_t \setminus t)$ erfülle die Blockbedingung. Seien $p, q \in P$ mit $q \notin \bullet t$.

Gilt $(N_t, t^-) \models \bullet t \blacktriangleleft t \bullet$, so bleibt beim Verfeinerungsschritt $(\Sigma \rightarrow \Sigma')$ die Lebendigkeitseigenschaft $p \triangleright q$ erhalten.

Beweis:

Seien $(K, r), (K', r')$ jeweils Abläufe von Σ, Σ' mit: (K', r') ist eine $t \rightarrow N_t$ Substitution von (K, r) , wobei $K = (B, E, \leq), K' = (B', E', \leq')$.

Angenommen, in K gilt $p \triangleright q$. Das heisst, für jedes $b_p \in B$ mit $r(b_p) = p$ gibt es ein $b_q \in B$ mit $r(b_q) = q$ mit: entweder $b_q \in K^\circ$ oder $b_p \leq^+ b_{q'}$, wobei $b_{q'} \in B$ mit $b_q \leq e \leq b_{q'}$ für ein $e \in E$. (1)

Zu zeigen ist: Für jedes $b_p \in B'$ mit $r'(b_p) = p$ gibt es ein $b_q \in B'$ mit $r'(b_q) = q$ mit: entweder $b_q \in K'^\circ$ oder $b_p \leq'^+ b_{q'}$, wobei $b_{q'} \in B'$ mit $b_q \leq' e \leq' b_{q'}$ für ein $e \in E'$.

Sei $b_p \in B'$ mit $r'(b_p) = p$.

Dann gilt auch $b_p \in B$ und $r(b_p) = r'(b_p) = p$. (wegen Bem. 3.2.5)

Dann gibt es ein $b_q \in B$ mit $r(b_q) = q$ mit: entweder $b_q \in K^\circ$ oder $b_p \leq^+ b_{q'}$,
wobei $b_q \leq e \leq b_{q'}$ für ein $e \in E$. (wegen (1))

Dann gilt auch $b_q \in B'$ und $r'(b_q) = r(b_q) = q$. (wegen Bem. 3.2.5)

Fall 1. $b_q \in K^\circ$, so $b_q \in K'^\circ$.

Fall 2. $b_q \notin K^\circ$:

Dann gilt: $b_{q'} \in B'$ und $r'(b_{q'}) = r(b_{q'})$. (wegen Bem. 3.2.5)

Es gilt $b_p \leq'^+ b_{q'}$. (wegen Satz 3.5.1)

Es gilt $b_q \leq' e \leq' b_{q'}$ für ein $e \in E'$. (wegen Voraussetzung $q \notin \bullet t$)

Das heisst, es gibt ein $b_q \in B'$ mit $r'(b_q) = q$ mit: entweder $b_q \in K'^\circ$ oder

$b_p \leq'^+ b_q'$, wobei $b_q \leq' e \leq' b_q'$ für ein $e \in E'$. □

Wir geben eine Erweiterung von Lemma 3.5.9 und vorher auch noch eine Bemerkung an.

Bemerkung 3.5.10

Sei $\Sigma = ((P, T, F), m_0)$ ein System, $R, Q \subseteq P$.

Sei (K, r) ein Ablauf von Σ , wobei $K = (B, E, \leq)$.

In (K, r) gilt $R \triangleright Q$, gdw.

für jede co-Menge B_R in K mit $r(B_R) = R$ gibt es eine co-Menge B_Q in K mit $r(B_Q) = Q$ mit:

Für jedes $b_q \in B_Q$ gilt:

entweder $b_q \in K^\circ$;

oder es gibt ein $b_r \in B_R$ mit $b_r \leq^+ b_q'$, wobei $b_q' \in B$ mit $b_q \leq e \leq b_q'$ für ein $e \in E$.

Beweis:

(\leftarrow):

Angenommen, für jede co-Menge B_R in K mit $r(B_R) = R$ gilt: Es gibt eine co-Menge B_Q in K mit $r(B_Q) = Q$ mit: Für jedes $b_q \in B_Q$ gilt: entweder $b_q \in K^\circ$ oder es gibt ein $b_r \in B_R$ mit $b_r \leq^+ b_q'$, wobei $b_q' \in B$ mit $b_q \leq e \leq b_q'$ für ein $e \in E$. (1)

Zu zeigen ist: Für jeden Schnitt C , in dem R gilt, gibt es einen von C erreichbaren Schnitt C' , in dem Q gilt. (2)

Sei $B_R \subseteq B$ co-Menge B_R in K mit $r(B_R) = R$.

Sei C ein Schnitt mit $B_R \subseteq C$.

Wegen (1) gilt:

Es gibt eine co-Menge B_Q in K mit $r(B_Q) = Q$ mit: Für jedes $b_q \in B_Q$ gilt: entweder $b_q \in K^\circ$ oder es gibt ein $b_r \in B_R$ mit $b_r \leq^+ b_q'$, wobei $b_q' \in B$ mit $b_q \leq e \leq b_q'$ für ein $e \in E$.

Dann gilt: $C \uparrow \text{co } B_Q$.

Daraus folgt: Es gibt einen von C erreichbaren Schnitt C' , der B_Q enthält.

Daraus folgt: (2) gilt.

(\rightarrow):

Angenommen, für jeden Schnitt C , in dem R gilt, gibt es einen von C erreichbaren Schnitt C' , in dem Q gilt. (3)

Zu zeigen ist: Für jede co-Menge B_R in K mit $r(B_R) = R$ gilt: Es gibt

eine co-Menge B_Q in K mit $r(B_Q)=Q$ mit: Für jedes $b_q \in B_Q$ gilt: entweder $b_q \in K^\circ$ oder es gibt ein $b_r \in B_R$ mit $b_r \leq^+ b_q'$, wobei $b_q' \in B$ mit $b_q \leq e \leq b_q'$ für ein $e \in E$.

(4)

Sei $B_R \subseteq B$ co-Menge B_R in K mit $r(B_R)=R$.

Sei C ein Schnitt mit $B_R \subseteq C$.

Wegen (3) gilt:

Es gibt einen von C erreichbaren Schnitt C' , in dem Q gilt.

D.h. C' enthält eine co-Menge B_Q in K mit $r(B_Q)=Q$.

Wegen $C \rightarrow^* C'$ gilt: $C \text{ [co } B_Q$.

Also, es gibt eine co-Menge B_Q in K mit $r(B_Q)=Q$ und $C \text{ [co } B_Q$.

Daraus folgt: Für jedes $b_q \in B_Q$ gilt $C \text{ [co } b_q$. Falls $b_q \notin K^\circ$ gilt, dann gibt es ein $b_r \in B_R$ mit $b_r \leq^+ b_q'$, wobei $b_q' \in B$ mit $b_q \leq e \leq b_q'$ für ein $e \in E$.

Daraus folgt: (4) gilt. □

Lemma 3.5.11

$\Sigma' = \Sigma(N_t \setminus t)$ erfülle die Blockbedingung. Seien $R, Q \subseteq P$ mit $Q \cap \bullet t = \emptyset$.

Gilt $(N_t, t^-) \models \bullet t \triangleleft t \bullet$, so bleibt beim Verfeinerungsschritt $(\Sigma \rightarrow \Sigma')$ die Lebendigkeitseigenschaft $R \triangleright Q$ erhalten.

Beweis:

Seien $(K, r), (K', r')$ jeweils Abläufe von Σ, Σ' mit: (K', r') ist eine $t \rightarrow N_t$ Substitution von (K, r) , wobei $K = (B, E, \leq), K' = (B', E', \leq')$.

Angenommen, in K gilt $R \triangleright Q$. (1)

Zu zeigen ist: In K' gilt $R \triangleright Q$. (2)

Sei B_R eine beliebige co-Menge B_R in K' mit $r'(B_R)=R$.

Dann ist $B_R \subseteq B$ auch in K eine co-Menge mit $r(B_R)=R$.

Wegen (1) gilt: (wegen Bem. 3.5.10)

Es gibt eine co-Menge B_Q in K mit $r(B_Q)=Q$ mit: Für jedes $b_q \in B_Q$ gilt: entweder $b_q \in K^\circ$ oder es gibt ein $b_r \in B_R$ mit $b_r \leq^+ b_q'$, wobei $b_q' \in B$ mit $b_q \leq e \leq b_q'$ für ein $e \in E$.

Dann gilt auch $B_Q \subseteq B'$

und B_Q ist auch eine co-Menge in K' . (wegen Bem. 3.2.5)

Für ein $b_q \in B_Q$:

Fall 1. $b_q \in K^\circ$, so $b_q \in K'^\circ$.

Fall 2. $b_q \notin K^\circ$:

Dann gilt: $b_q' \in B'$ und $b_q \leq'^+ b_q'$.

und $b_r \leq'^+ b_q'$. (wegen Bem. 3.2.5 und Satz 3.5.1)

Es gilt $b_q \leq' e \leq' b_q'$ für ein $e \in E'$. (wegen Voraussetzung $Q \cap \bullet t = \emptyset$)

Daraus folgt: (2) gilt. (wegen Bem. 3.5.10)

Daraus folgt die Behauptung. \square

Lemma 3.5.12

$\Sigma' = \Sigma(N_t \setminus t)$ erfülle die Blockbedingung. Seien $p, q \in P$.

Gilt $(N_t, t^-) \models \bullet t \blacktriangleleft t \bullet$, so bleibt im Verfeinerungsschritt $(\Sigma \rightarrow \Sigma')$ die Sicherheitseigenschaft $\square \neg(p \wedge q)$ erhalten. \square

Beweis:

Angenommen, in Σ gilt: $\square \neg(p \wedge q)$ (1)

Zu zeigen ist: In Σ' gilt $\square \neg(p \wedge q)$ (2)

Seien $(K, r), (K', r')$ Abläufe von Σ, Σ' mit: (K', r') ist eine $t \rightarrow N_t$

Substitution von (K, r) , wobei $K = (B, E, \leq), K' = (B', E', \leq')$.

Seien $b_p, b_q \in B'$ beliebige Auftreten von p, q in K' , d.h. $r'(b_p) = p, r'(b_q) = q$.

Dann gilt:

$b_p, b_q \in B$ und $r(b_p) = r'(b_p) = p, r(b_q) = r'(b_q) = q$. (wegen Bem. 3.2.5)

Dann gilt in K : $\neg(b_p \underline{\text{co}} b_q)$. (wegen (1))

Dann gilt: entweder $b_p \leq^+ b_q$ oder $b_q \leq^+ b_p$.

In K' gilt weiter:

$b_p \leq'^+ b_q$ falls $b_p \leq^+ b_q$;

$b_q \leq'^+ b_p$ falls $b_q \leq^+ b_p$. (wegen Satz 3.5.1)

D.h. in K' gilt $\neg(b_p \underline{\text{co}} b_q)$.

Daraus folgt: (2) gilt. \square

4. Agentensysteme

Durch Agentensysteme beschreiben wir die Klasse von Petrinetzsystemen, für die wir das durch diese Arbeit behandelte Thema der verteilenden Verfeinerung untersuchen wollen.

Wir wissen schon, dass für verteilende Verfeinerungen zumindest die im vorigen Kapitel eingeführte Blockbedingung gelten muss. Deshalb ist es wichtig, dass für Agentensysteme möglichst leicht die Gültigkeit der Blockbedingung nachgewiesen werden kann. Das geschieht dadurch, dass die Synchronisation der Arbeit von Agenten auf den Nachrichtenaustausch zurückgeführt wird.

Aus dem vorigen Abschnitt wissen wir weiter, dass zusätzlich zur Blockbedingung bei einer Transitionsverfeinerung Kausalitäten im System erhalten bleiben müssen, wenn bei der Verfeinerung spezielle Lebendigkeits- und Sicherheitseigenschaften erhalten bleiben sollen. Kausalitäten im System lassen sich aber nur für 1-beschränkte Systeme hinreichend einfach bestimmen (vgl. Abschnitt 2.4).

In diesem Kapitel klären wir zuerst, was für uns Agenten und Agentensysteme sind. Dann zeigen wir, warum es sinnvoll ist, weitere Eigenschaften der Agentensysteme zu fordern.

4.1 Warum Agentensysteme

Dieser Abschnitt zeigt, warum die Stellen eines Netzes und nicht die Transitionen zur Definition von Agenten geeignet sind.

In verteilten Algorithmen kommunizieren Agenten mit Hilfe von Nachrichten. Es ist intuitiv klar, was die Agenten sind und was die Nachrichtenkanäle. Bei unserer Verifikationsmethode haben wir schrittweise einfachere Algorithmen gesucht, aus denen der zu verifizierende Algorithmus durch einen einfach zu beweisenden Verfeinerungsschritt zu erhalten ist. Die einfacheren Algorithmen waren nicht verteilt. Aber es war weiter intuitiv klar, was Agenten und Kanäle sind, falls der gröbere Algorithmus noch Kanäle enthielt. Wir konnten die Vergrößerung in den von uns untersuchten Beispielen so beenden, dass die geforderten Eigenschaften leicht direkt zu beweisen waren. Außerdem war weiter zu erkennen, was die Agenten sind. Nicht in allen Netzen ist das möglich. Daher lag es nahe, nach einer Definition einer Netzklasse zu

suchen, für die klar ist, was Agenten sind.

Wir möchten hier mit Hilfe eines Beispiels erklären, warum wir den Begriff Agentensysteme im Petrinetzmodell eingeführt haben. In Abb. 4.1.1 ist der Algorithmus *token-passing-Mutex* (aus [Rei 98]) angegeben. Dieser Algorithmus ist einerseits ein typisches Kommunikationsprotokoll, andererseits ist das *Mutex*-Problem eines der bekanntesten Probleme für verteilte Algorithmen. An der Abbildung erkennt man vielleicht schon, dass der Algorithmus wegen der Kommunikation kompliziert ist. Wir werden zeigen, was die einfachste Version dieses Algorithmus ist, in der die Agenten gerade noch erkennbar sind.

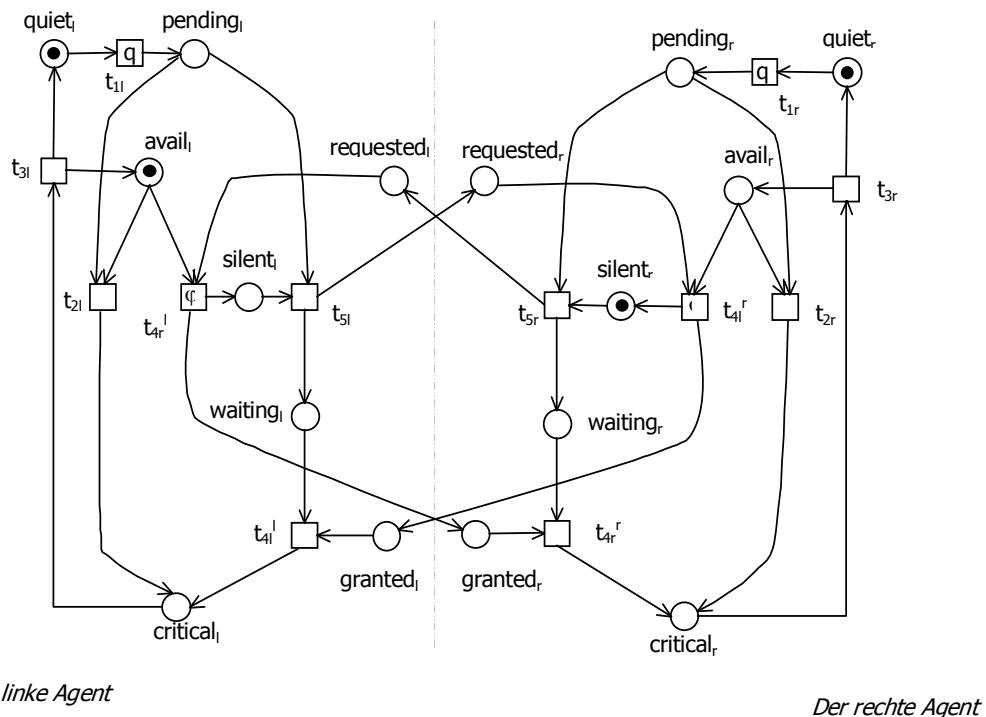


Abb. 4.1.1 Algorithmus *token-passing-Mutex* (Σ^*) (aus [Rei 98])

Der Algorithmus *token-passing-Mutex* Σ^* ist ein Algorithmus über Ressourcen-Management für ein verteiltes System, in dem es zwei Komponenten (auch *Agenten* genannt) gibt, die nur durch Nachrichten miteinander kommunizieren können. Die beiden Agenten haben gemeinsam eine Ressource. Dieser Algorithmus garantiert den wechselseitigen Ausschluss, d.h. die beiden werden nie gleichzeitig die Ressource benutzen. Weiter garantiert der Algorithmus, dass jeder Agent irgendwann die Ressource benutzen wird, wenn er sie braucht.

In der Abb. 4.1.1 steht links der linke Agent l , und rechts ist der rechte Agent r . Ein Agent kann im Zustand 'kein Bedarf an der Ressource' (das ist mit der Stelle $quiet_l$ bzw. der Stelle $quiet_r$ modelliert, wobei der Index l bzw. r den Agenten l und den Agenten r bezeichnet) stehen, er kann aber auch im Zustand 'brauchen die Ressource' (Stelle $pending_l$ bzw. $pending_r$) stehen.

Die Ressource kann sowohl im Zustand 'benutzt von einem Agenten' (Stelle $critical_l$ bzw. $critical_r$) als auch im Zustand 'frei und bei einem Agenten' (Stelle $avail_l$ bzw. $avail_r$) stehen.

Wenn l die Ressource braucht (Stelle $pending_l$), und die Ressource bei ihm ist (Stelle $avail_l$), dann kann er sie benutzen und in den Zustand $critical_l$ ⁸ übergehen (Transition t_{2l}).

Wenn er die Ressource nicht mehr braucht, geht er in den Zustand $quiet_l$ zurück, die Ressource wird wieder frei und bleibt bei ihm (Stelle $avail_l$, Transition t_{3l}).

Wenn l die Ressource braucht und die Ressource nicht bei ihm ist (Stelle $silent_l$) sondern bei dem anderen Agenten r ist, dann sendet er eine Nachricht $requested_l$ an r , und geht in den Zustand $waiting_l$ über. Wenn r diese Nachricht $requested_l$ bekommt, und die Ressource frei (Stelle $avail_r$) ist, dann soll r (laut *fairness*-Annahme) eine Nachricht $granted_r$ zurückschicken (Transition t_{4l}^r). Wenn l die Antwort $granted_r$ bekommt, dann benutzt er die Ressource (Transition t_{4l}^l).

Für den rechten Agenten r ist es auch ähnlich: Wenn er die Ressource braucht, und sie aber bei dem anderen Agenten ist, dann schickt er eine Nachricht $requested_r$. Wenn er eine Antwort $granted_l$ bekommt, dann benutzt er die Ressource.

Das ist also ein Kommunikationsprotokoll für eine verteilte Architektur mit mehreren Agenten. Wie schon erwähnt, wir beweisen solche Algorithmen mit der Verfeinerungsmethode. Wir suchen zunächst einen einfachen Anfangsalgorithmus.

⁸ Die stelle $critical_l$ modelliert auch, dass der linke Agent l gerade die Ressource benutzt.

In Abb. 4.1.2 wird nur das Mittelteil des Algorithmus angegeben.

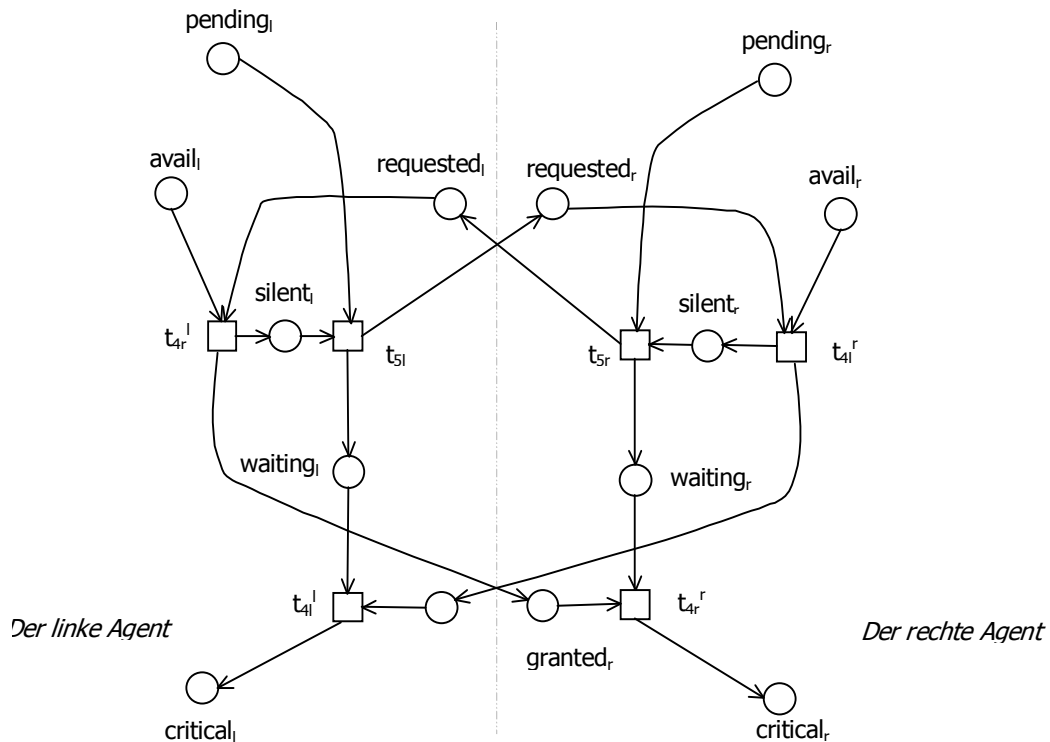


Abb. 4.1.2 Ein Teil N_1 von Σ^*

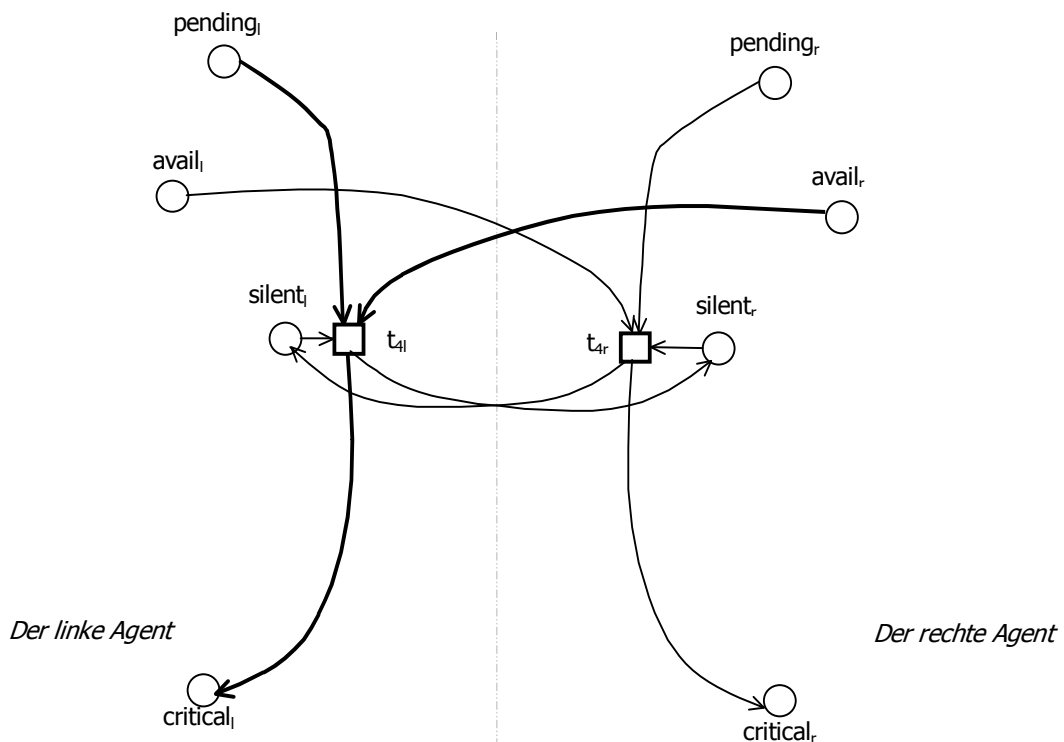


Abb. 4.1.3 Ein vereinfachtes Netz N_2 von N_1

Wenn wir nicht genau beschreiben, wie die beiden Agenten miteinander kommunizieren, dann wird es viel einfacher (Abb. 4.1.3): Wenn l die Ressource braucht, und die Ressource frei und bei r ist, dann wird l irgendwann die Ressource benutzen, und gleichzeitig ist die Ressource

nicht mehr bei r (Transition t_{4l}). Für r ist es ähnlich (Transition t_{4r}). Für jeden Agenten gibt es jetzt nur noch eine einzige Transition für den Austausch der Ressource (noch weniger geht nicht).

Wenn wir im Algorithmus Σ^* das Teil N_1 (Abb. 4.1.2) durch N_2 (Abb.4.1.3) ersetzen und sonst nichts verändern, erhalten wir den Algorithmus Σ_0 (Abb.4.1.4). Das ist bei einem solchen Kommunikationsprotokoll für die verteilte Architektur mit mehreren Agenten schon der einfachste Anfangsalgorithmus. Also, Σ_0 ist unser Anfangsalgorithmus für Σ^* .

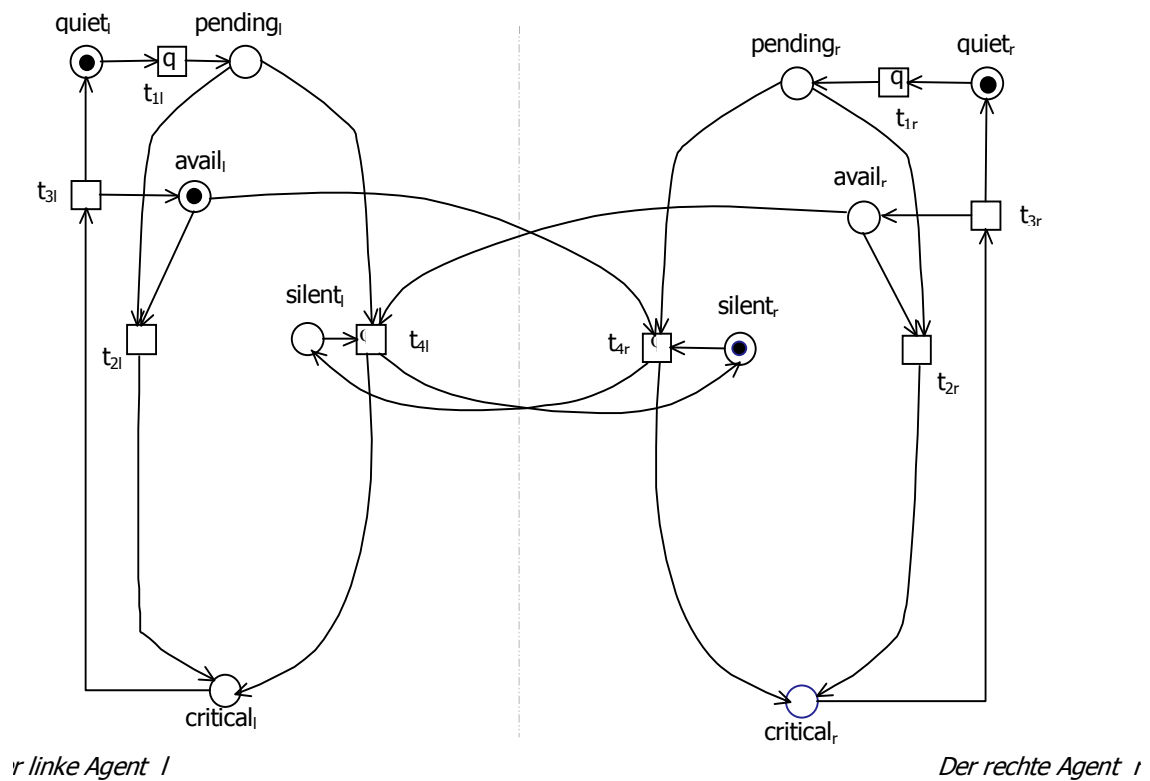
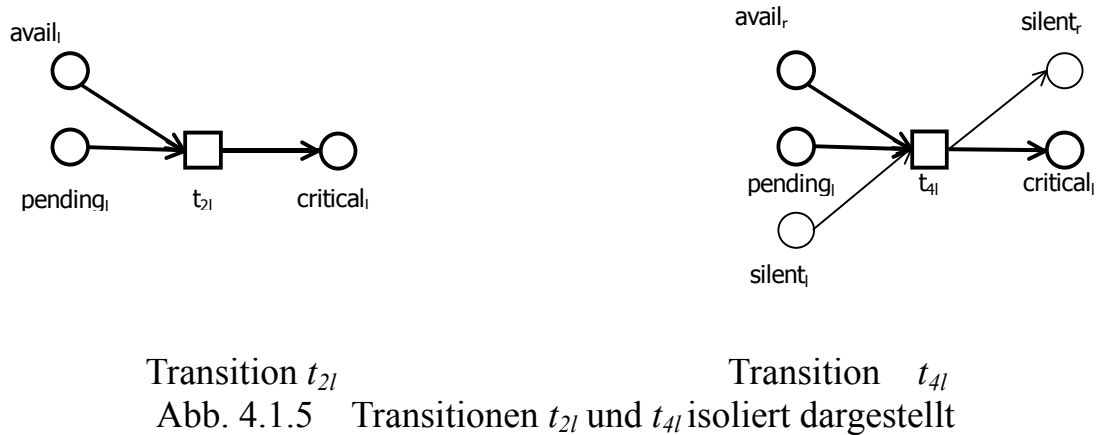


Abb. 4.1.4 Anfangsalgorithmus Σ_0

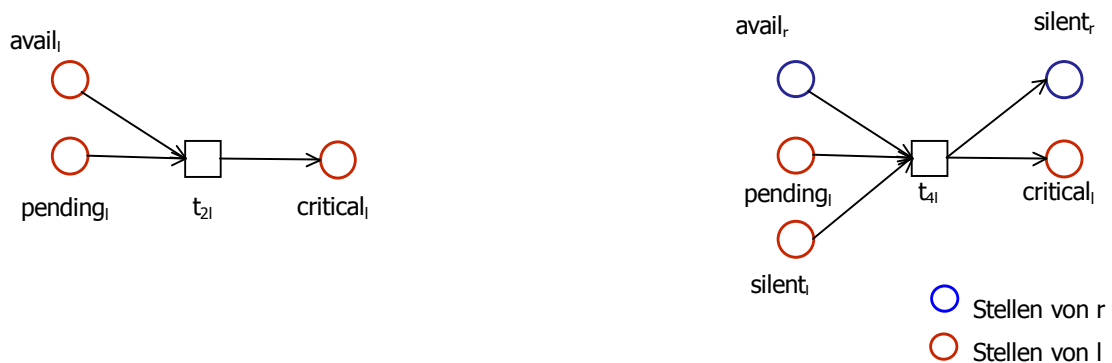
In Σ_0 sind die Aufgaben der beiden Transitionen t_{2l} und t_{4l} . Wenn l die Ressource braucht und sie frei ist, dann wird l sie benutzen. In Abb. 4.1.5 werden die beiden Transitionen isoliert dargestellt.



In der Abb. 4.1.5 sehen die beiden Transitionen ähnlich aus, sind aber sehr unterschiedlich: t_{2l} brauchen wir gar nicht zu verfeinern, sie ist schon ausführbar (für die verteilte Architektur). t_{4l} müssen wir aber Schritt für Schritt verfeinern, bis sie ausführbar ist. Warum? Denn t_{2l} betrifft nur einen Agenten. Sie ist eine lokale (auch *verteilt* genannt) Aktion. t_{4l} betrifft nicht nur einen Agenten, sondern die beiden. Sie ist eine nicht-lokale (auch *unverteilt* genannt) Aktion.

Wie können wir im Petrinetzmodell deutlich sehen, ob eine Transition schon verteilt oder noch nicht verteilt ist? Dafür brauchen wir den Begriff *Agent* bzw. *Agentensysteme*.

In Abb. 4.1.6 bezeichnen die roten Stellen die Stellen des linken Agenten l und die blauen sind Stellen des Agenten r . Jetzt ist ganz klar, t_{2l} ist verteilt (lokale Aktion von l), und t_{4l} nicht. t_{4l} müssen wir verfeinern.



In Abb. 4.1.7 ist der Anfangsalgorithmus Σ_0 durch ein Agentensystem modelliert. Wir sehen, die Transition t_{3l} ist auch verteilt, die brauchen wir nicht zu verfeinern.

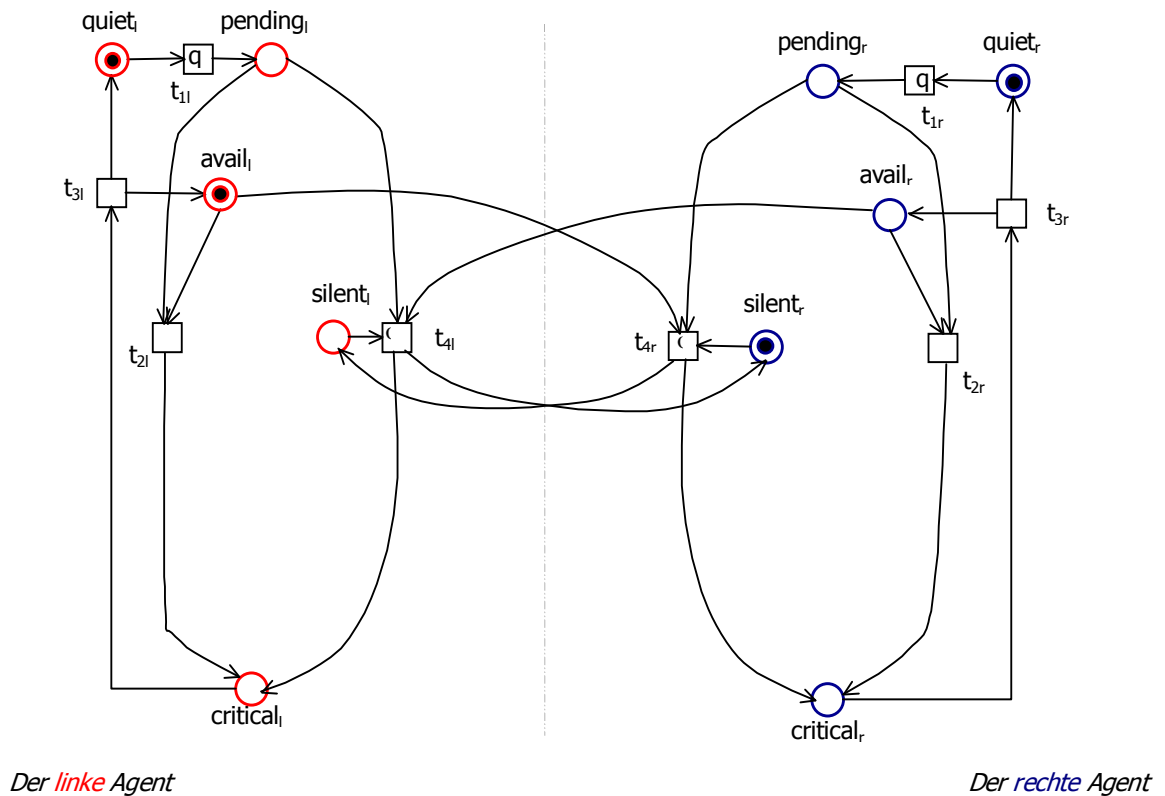


Abb. 4.1.7 Σ_0 als Agentensystem

Der Verfeinerungsprozess ist also: Wir verfeinern die Transitionen t_{4l} , t_{4r} jeweils zu einem Netz, das nur noch lokale Aktionen der beiden Agenten enthält. Danach haben wir einen verteilten Algorithmus, den Algorithmus Σ^* .

Mit Hilfe des Begriffs der Agentensysteme ist auch der Verfeinerungsprozess übersichtlich.

4.2 Definition der Agentensysteme

Als Voraussetzung für die Definition der Agentensysteme sind vorher die Begriffe *Agent* und *Kanal* zu definieren. Zunächst werden wir durch einfache Beispiele erklären, was wir als Agenten bezeichnen wollen und was nicht. Anschließend zeigen wir, wie ein Kanal aussehen soll und was bei der Definition eines Kanals ausgeschlossen werden soll. Schließlich werden wir formale Definitionen für die Begriffe *Agentensysteme*, *verteilt* und *unverteilt* angeben.

4.2.1 Was ist ein Agent in einem Petrinetz

Ein Agent kann durch seine Aktionen oder durch seinen Zustand charakterisiert werden. Im *Mutex*-Algorithmus in Abb. 4.1.1 bestanden die Aktionen (also die Transitionen) in der wechselseitigen Benutzung einer gemeinsamen Ressource. Der Zustand der Agenten wurde durch die Belegung der Stellen des Netzes charakterisiert. Nach der Vergrößerung des Algorithmus konnten die Transitionen t_{4l} und t_{4r} keinem Agenten zugeordnet werden. Sie modellierten gemeinsame Aktionen beider Agenten. Die Belegung der Stellen charakterisierte aber weiter den Zustand der Agenten. Hier eigneten sich nur noch die Stellen zur Charakterisierung der Agenten. Weiter werden wir die 1-Beschränktheit für Agentensysteme fordern. Netze, die sich durch Invarianten mit der Summe 1 überdecken lassen, sind 1-beschränkt. Alle als Beispiele betrachteten Agentensysteme ließen sich durch Invarianten mit Summe 1 überdecken, so dass dieses Kriterium als brauchbares Kriterium für die 1-Beschränktheit von Agentensystemen angesehen werden kann.

Beispiel *producer/consumer* (aus [Rei98])

In Abb. 4.2.1 ist das System Σ_1 *producer/consumer* dargestellt.

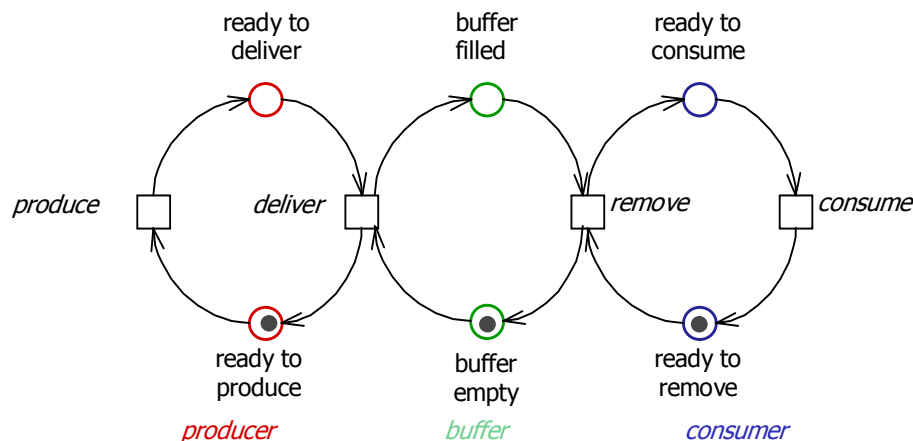


Abb. 4.2.1 *producer/consumer* (System Σ_1)

Der Produzent (links) steht entweder im Zustand *ready to produce* oder im Zustand *ready to deliver*, d.h. *ready to produce* und *ready to deliver* sind die Zustände des Produzenten. Die Stellenmenge *producer*:

$$\text{producer} = \{ \text{ready to produce}, \text{ready to deliver} \}$$

können wir als einen Agenten bezeichnen, also als Agenten *producer*.

Hier ist

$$\text{ready to produce} + \text{ready to deliver} = 1$$

eine Stelleninvariante von Σ_1 .

Analog, der Konsument (rechts) steht entweder im Zustand *ready to remove* oder im Zustand *ready to consume*, d.h.

$$\text{ready to remove} + \text{ready to consume} = 1$$

ist eine Stelleninvariante von Σ_1 . Die Stellenmenge *consumer*:

$$\text{consumer} = \{ \text{ready to remove}, \text{ready to consume} \}$$

können wir auch als einen Agenten bezeichnen, also als Agenten *consumer*.

In der Mitte steht der Puffer, er ist entweder *empty* oder *filled*, d.h.

$$\text{buffer empty} + \text{buffer filled} = 1$$

ist eine Stelleninvariante von Σ_1 . Die Stellenmenge *buffer*:

$$\text{buffer} = \{ \text{buffer empty}, \text{buffer filled} \}$$

können wir auch als einen Agenten bezeichnen, also als Agenten *buffer*.

Wir können auch den Produzenten und den Puffer zusammen als einen Agenten betrachten (Abb. 4.2.2), d.h. die Stellenmenge *p-b*:

$$p-b = \{ \text{ready to produce}, \text{ready to deliver}, \text{buffer empty}, \text{buffer filled} \}$$

kann auch ein Agent sein.

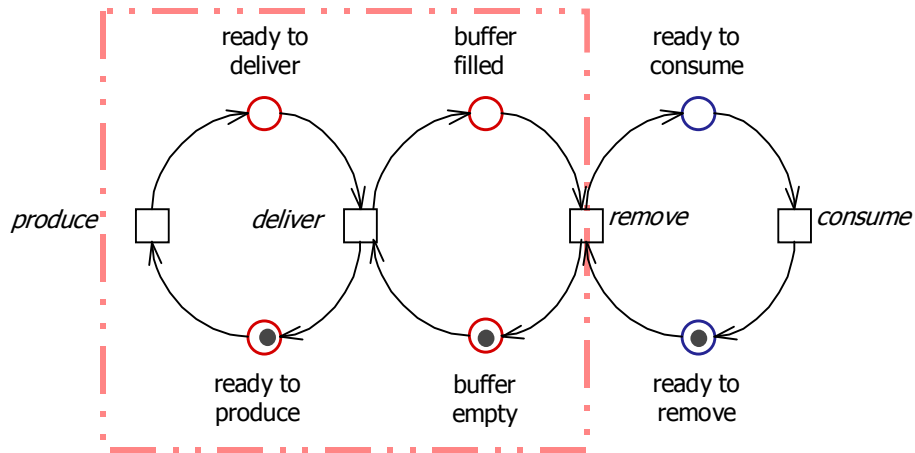


Abb. 4.2.2 Ein weiterer Agent – der *Produzent* mit dem *Puffer* zusammen (System Σ_1)

Beispiel, das kein Agentensystem sein soll, weil es nicht 1-beschränkt ist:

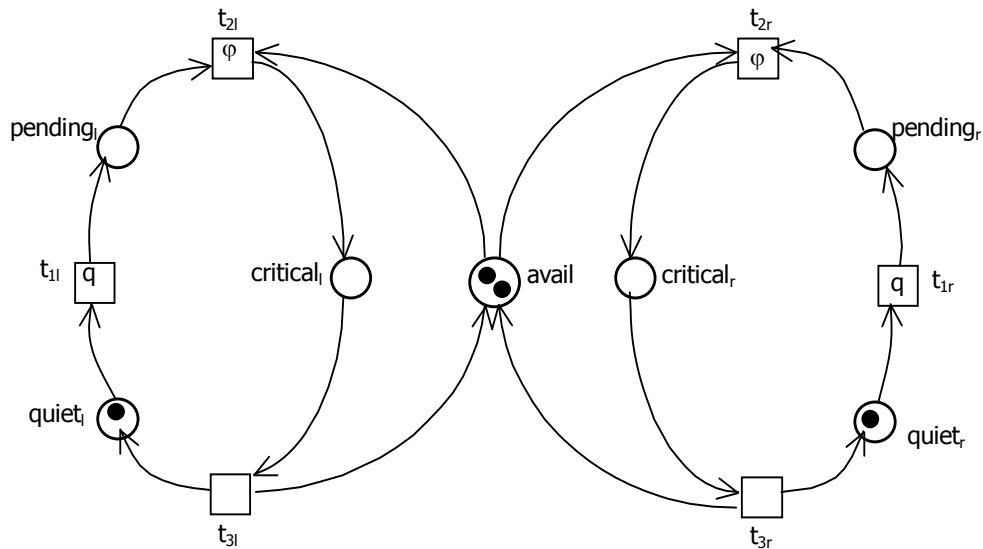


Abb. 4.2.3 Ein System, das kein Agentensystem sein soll

Obwohl beide Seiten eine gemeinsame Stelle haben, arbeiten beide vollständig unabhängig voneinander. Ein verteilter Algorithmus mit gleicher Funktion ist:

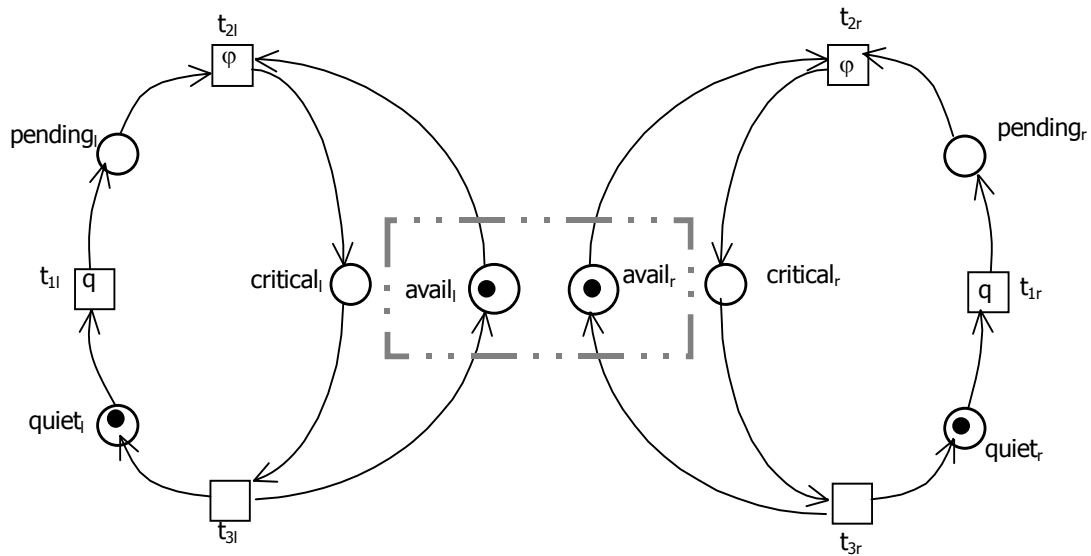


Abb. 4.2.4 Ein verteiltes System mit gleicher Funktion wie das System in Abb. 4.2.3

Hier wurde eine Stelle verteilt und ein trivialer verteilter Algorithmus erhalten.

Dieses Beispiel zeigt nochmals, ohne 1-Beschränktheit gibt es keine strukturellen Kriterien zur Bestimmung der in dieser Arbeit wichtigen temporal-logischen Eigenschaften.

Der asymmetrische *Mutex* kann dagegen als Agentensystem angesehen werden, z.B. wenn wir die Ressource Links zuordnen. Dann wird t_{2r} und t_{3r} unverteilt. Die algorithmische Idee ist dabei, t_{2r} borgt sich die Ressource und t_{3r} gibt sie nach Gebrauch sofort zurück. Wie wir schon in der Einleitung gesehen haben, lassen sich diese beiden Transitionen durch verteilende Verfeinerungen verteilen. Es wird der asymmetrische nachrichtenbasierte *Mutex* (vgl. [Rei98] Abschnitt 13.10) erhalten.

4.2.2 Was ist ein Kanal in Agentensystemen?

Üblicher Weise versteht man folgendes unter einem Kanal: Über einen Kanal kann eine Menge von Agenten eine Nachricht senden. Die Nachricht kann von einer dazu disjunkten Menge von Agenten empfangen werden. Diese Definition kann nicht direkt auf unverteilte Algorithmen übertragen werden.

Zur Vorbereitung der Definition einer Kanalstelle in Agentensystemen sehen wir uns daher zunächst ein paar Beispiele an.

Beispiele

In Abb. 4.2.5 ist ein System Σ_1 angegeben. Es gibt zwei Agenten: Agent a und Agent b .

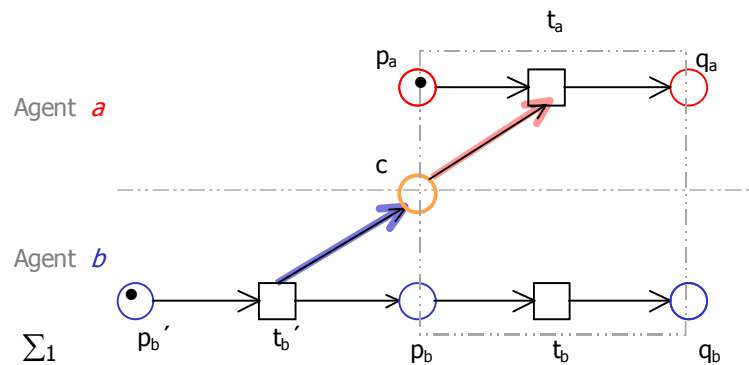


Abb. 4.2.5 Ein Kanal

Agent b sendet ein Signal c (Transition t_b'). Dieses Signal wird von dem anderen Agenten a empfangen (Transition t_a). Die Stelle c können wir als einen *Kanal* bezeichnen.

In Abb. 4.2.6 ist ein anderes System Σ_2 angegeben.

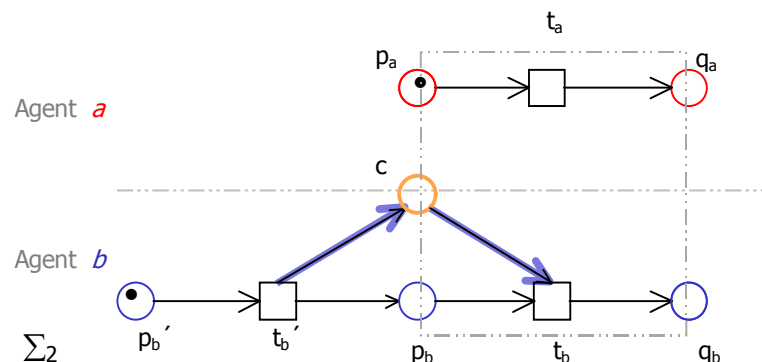


Abb. 4.2.6 Kein Kanal

Agent b legt auch ein Token auf die Stelle c . Dieses Token wird später aber von ihm selbst konsumiert. Die Stelle c wollen wir nicht als einen Kanal bezeichnen.

Jetzt betrachten wir noch das Problem, dass eine Transition eine gemeinsame Aktion mehrerer Agenten modellieren kann. In Abb. 4.2.7 ist das dritte System Σ_3 angegeben.

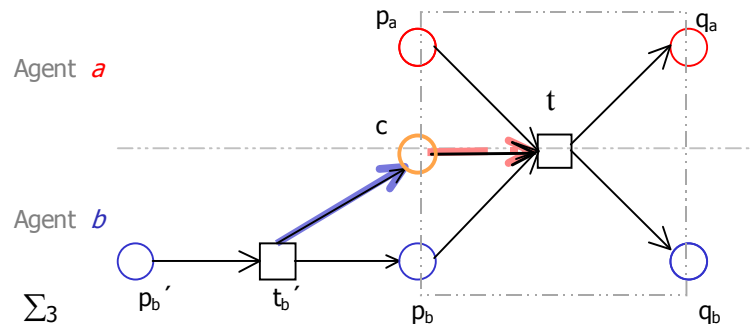


Abb. 4.2.7 Der Kanal c vor einer Verfeinerung in Σ_3

In Σ_3 soll c eine Kanalstelle sein. Die Transition t , die das Signal c empfängt, betrifft neben dem Agenten a auch noch den Agenten b , den Sender dieses Signals.

Wir können trotzdem die Stelle c als einen Kanal betrachten, weil wir die gemeinsame Aktion t von a und b in einem Verfeinerungsschritt in Aktionen t_a von a und t_b von b so zerlegen können (Abb. 4.2.8), dass nur noch t_a auf c , die gesendete Information, zugreift.

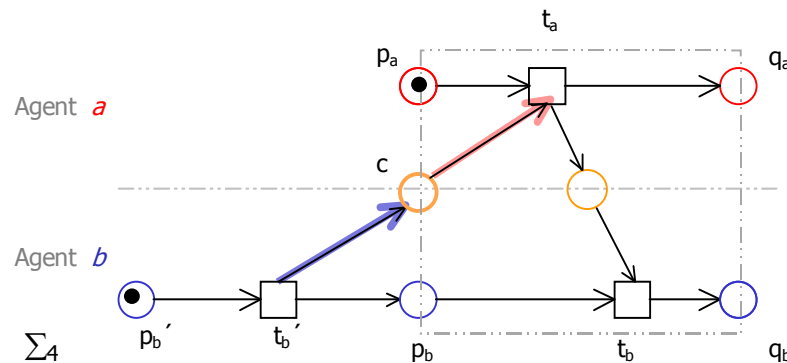


Abb. 4.2.8 Der Kanal c nach einer Verfeinerung in Σ_3

An diesem Beispiel sehen wir, dass die Festlegung der Kanalstellen in unverteilter Algorithmen eine Entwurfsentscheidung sein kann.

Das Beispiel *Mutex* zeigt, dass eine gemeinsame Variable keine Kanalstelle sein sollte.

Die Stelle *avail* (Abb. 4.2.9) ist kein Kanal, weil es sein kann, dass der Agent *l* ein Token auf die Stelle *avail* legt und später dieses Token wieder selbst konsumiert.

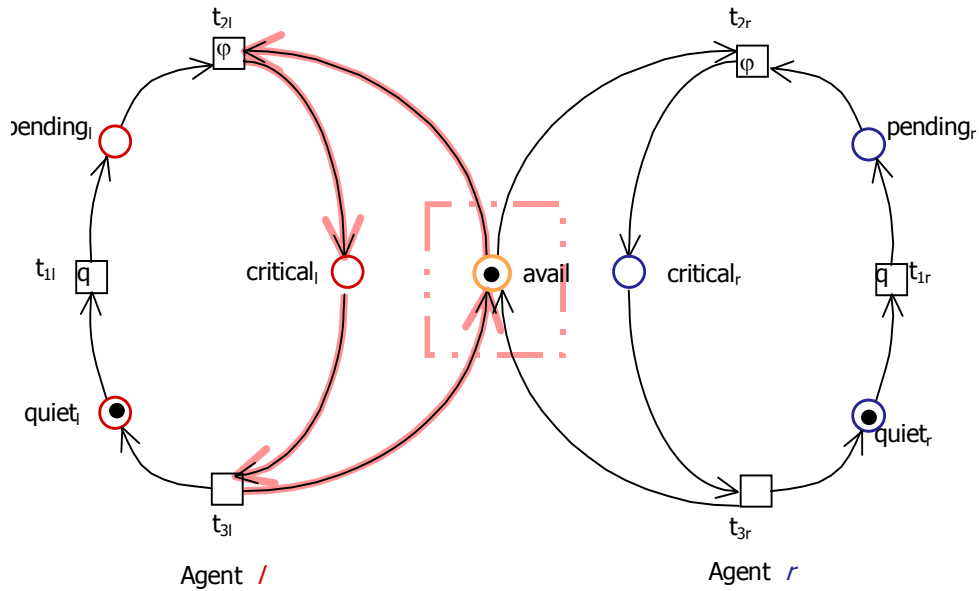


Abb. 4.2.9 Kein Kanal (*Mutex*)

4.2.3 Definitionen Agent, Kanal, Agentensystem

Vorhin haben wir schon diskutiert, welche Bedingungen ein Agent und ein Kanal erfüllen sollen. Nun definieren wir den Begriff Agentensystem.

Wenn eine Stelle des Systems eine Kanalstelle ist, dann soll diese Stelle nicht auch eine Agentenstelle sein. Wenn eine Stelle zu einem Agenten gehört, dann soll diese Stelle nicht gleichzeitig zu einem anderen Agenten gehören. Sei P die Stellenmenge des Systems Σ . Dann ist die Kanalmenge C von Σ eine Teilmenge von P , und die Agentenmenge von Σ eine Partition⁹ von $P \setminus C$.

Wie schon erwähnt, wir betrachten auch Systeme, die nicht-lokale Aktionen enthalten. Deshalb kann es sein, dass es in einem Agentensystem keinen Kanal gibt.

Definition in zwei Schritten: In der ersten Definition betrachten wir die Struktur von Netzen mit Agenten und Kanälen. In der zweiten Definition formulieren wir die Anforderungen an die von uns weiter untersuchten Agentensysteme.

Definition 4.1 (*Netz mit Agenten und Kanälen*)

Sei $N=(P,T,F)$ ein Netz. Sei $C \subseteq P$, A eine Partition von $P \setminus C$, dann heißt N ein Netz mit Agentenmenge A und Kanalmenge C .

Die Elemente von A und C nennen wir Agenten und Kanäle.

Wenn alle Agentenstellen aus dem Vor- und Nachbereich einer Transition demselben Agenten gehören, dann ist diese Transition verteilt, sonst ist sie unverteilt. Wenn alle Transitionen von einem System verteilt sind, dann ist dieses System verteilt, sonst ist es nicht verteilt.

Definition 4.2 (*verteilt, unverteilt*)

Sei $N=(P,T,F)$ ein Netz mit Agentenmenge A und Kanalmenge C , sei $t \in T$.

i. Sei $a \in A$ ein Agent.

t ist lokal in a , gdw. $\bullet t \cup t \bullet \subseteq a \cup C$ (t greift nur auf Kanalstellen und Stellen von a zu).

ii. t ist verteilt, gdw. es ein $a \in A$ gibt, so dass t lokal in a ist.

iii. t ist unverteilt, gdw. t nicht verteilt ist.

⁹ Eine Partition einer Menge M ist eine Familie P aus nichtleeren, disjunkten Teilmengen von M , so dass jedes Element von M in genau einer Menge von P enthalten ist.

iv. N ist verteilt, gdw. jedes $t \in T$ verteilt ist.

Wir betrachten nur solche Systeme, bei denen die Agenten und die Kanalstellen die von uns geforderten Eigenschaften besitzen. Solche Systeme nennen wir Agentensysteme.

Definition 4.3 (Agentensystem)

Sei $\Sigma = (N, m_0)$ ein System mit: $N = (P, T, F)$ ist ein Netz mit Agentenmenge A und Kanalmenge C .

Σ ist ein Agentensystem, gdw. die folgenden Bedingungen erfüllt sind:
 Σ ist 1-beschränkt.

2. Für jedes $c \in C$ gilt:

Sind $t_1, t_2 \in T$ mit $t_1 \neq t_2$ und lokal zu $a \in A$, dann gilt $c \in \bullet t_1 \rightarrow \neg c \in t_2 \bullet$
(ein Agent kann nicht gleichzeitig Sender und Empfänger einer Nachricht sein.)

Als ein einfach zu handhabendes Kriterium für die 1-Beschränktheit benutzen wir: Alle Agentenstellen eines Systems $\Sigma = ((P, T, F), m_0)$ sind 1-beschränkt, wenn für jeden Agenten $a \in A$ gilt: Zu jeder Stelle $p \in a$ gibt es eine Teilmenge $v \subseteq a$ mit $p \in v$, so dass $\sum v = 1$ eine Stelleninvariante vom System Σ ist¹⁰. Im Abschnitt 4.3.3 zeigen wir, dass Σ besonders einfach zu implementieren ist, wenn jede Stelle eines Agenten durch eine Stelleninvariante mit der Summe 1 dieses Agenten zu überdecken ist.

Wir beschränken uns hier auf 1-beschränkte Systeme. Der wichtigste Grund dafür ist, dass die Kausalitätsrelation \blacktriangleleft nur für 1-beschränkte Systeme einfach bestimmt werden kann und die Relation \blacktriangleleft eine wesentliche Rolle bei der korrekten Verfeinerung spielt.

Im Einführungsbeispiel in Abschnitt 4.1, ist der Anfangsalgorithmus Σ_0 (siehe Abb. 4.1.7) ein Agentensystem mit Agentenmenge $A = \{l, r\}$, wobei
 $l = \{ \text{quiet}_l, \text{pending}_l, \text{critical}_l, \text{avail}_l, \text{silent}_l \}$
der linke Agent ist, und
 $r = \{ \text{quiet}_r, \text{pending}_r, \text{critical}_r, \text{avail}_r, \text{silent}_r \}$

¹⁰ Sei $v = \{p_1, p_2, \dots, p_n\}$. $\sum v$ steht für $p_1 + p_2 + \dots + p_n$.

der rechte Agent ist. In Σ_0 gibt es keinen Kanal, d.h. $C = \emptyset$.
Die Transition t_{2l} und t_{3l} sind verteilt, und die Transition t_{4l} ist unverteilt. Da das System Σ_0 unverteilte Transitionen enthält, ist es unverteilt. In diesem Beispiel ist jede Stelle eines Agenten durch eine Stelleninvariante mit Summe 1 dieses Agenten zu überdecken. Z. B. Stelle $avail_l$ vom Agenten l wird durch die Stelleninvariante \square ($critical_l + avail_l + silent_l = 1$) von l überdeckt.

4.3 Agentensysteme als Anfangsnetze für den Entwurf von verteilten Algorithmen

- am Beispiel des verteilten *Mutex*

Im Abschnitt 4.1 haben wir zu dem zu verifizierenden verteilten Algorithmus Σ^* durch Vergrößerung einen einfacheren Algorithmus Σ_0 konstruiert, der noch die zu beweisenden Eigenschaften besitzt. Σ_0 war kein verteilter Algorithmus mehr, war aber nach unserer Definition noch ein Agentensystem. Wir gehen jetzt der Frage nach, ob dieser Algorithmus Σ_0 einfach genug ist, um die Gültigkeit der gewünschten Eigenschaften des Algorithmus¹¹ leicht direkt beweisen zu können. Es gibt noch ein einfaches Petrinetz Σ_a (Grundmodell) das auch die gewünschte Sicherheits- und Lebendigkeitseigenschaft besitzt (Abb. 4.2.9). Es ist kein Agentensystem und modelliert nicht die algorithmische Idee des nachrichtenbasierten *Mutex*: Jeder Agent behält die Ressource solange, bis der andere Agent die Ressource benutzen möchte. Dazu wollen wir Σ_0 mit einem anderen Algorithmus, der abstrakter und noch kein Agentensystem ist, vergleichen. Dadurch möchten wir weiter erklären, warum wir Agentensysteme als Anfangsalgorithmen für verteilte zu beweisende Ziel-Algorithmen ausgewählt haben.

Wir zeigen zunächst die Korrektheit des Anfangsalgorithmus Σ_0 , der ein Agentnetz ist. Dabei vergleichen wir diesen Beweis mit dem Beweis des Grundmodells Σ_a . Dadurch zeigen wir, das Agentensystem Σ_0 ist fast so leicht zu beweisen wie das Grundmodell Σ_a . Und deshalb sagen wir, unser Anfangsalgorithmus Σ_0 ist zwar komplizierter aber nur unwesentlich komplizierter.

Zweitens geben wir einen Entwurfsprozess des Anfangsalgorithmus Σ_0 an, bei dem wir von dem nicht Agentensystem Σ_a ausgehend das Agentensystem Σ_0 entwerfen. Dabei werden wir auch zeigen, unser Anfangsalgorithmus Σ_0 ist zwar komplizierter, enthält aber schon die algorithmische Idee des verteilten Ziel-Algorithmus Σ^* . Der Entwurfsprozess besteht aus einer Folge von Beobachterverfeinerungen.

¹¹ Gewünschte Eigenschaften eines Algorithmus sind die Eigenschaften, die dieser Algorithmus garantieren soll. Üblicherweise ist eine gewünschte Eigenschaft eine Sicherheitseigenschaft oder eine Lebendigkeitseigenschaft. Bei *Mutex*-Algorithmen sind folgende Eigenschaften gewünscht: Beide Agenten werden nie gleichzeitig die Ressource benutzen, und ein Agent wird irgendwann die Ressource benutzen, wenn er sie braucht.

Damit besteht eine weitere Möglichkeit die Korrektheit unseres Anfangsalgorithmus Σ_0 ausgehend vom Grundmodell zu beweisen.

Schließlich geben wir ein nebenläufiges Programm an, das dem Agentensystem Σ_0 entspricht. Dabei zeigen wir, einer Variable des Programmes entspricht im Petrinetzmodell eine Invariante.

4.3.1 Korrektheit des Anfangsalgorithmus Σ_0

Im Einführungsbeispiel in Abschnitt 4.1 haben wir schon gesehen, für den Ziel-Algorithmus Σ^* *token-passing-Mutex* (siehe Abb. 4.1.1) ist der Algorithmus Σ_0 in Abb. 4.3.3 schon die stärkste Vereinfachung (Vergrößerung), die wir noch als Agentensystem ansehen wollen und von dem ausgehend wir durch verteilende Verfeinerung den Ziel-Algorithmus Σ^* erreichen können. Ein noch abstrakterer Algorithmus als dieser kann dann unsere Forderungen nicht erfüllen. Ein solcher Algorithmus für Σ^* ist der *Mutex* Algorithmus Σ_a in Abb. 4.3.1.

In diesem Abschnitt werden wir den Beweis der *Mutex*-Eigenschaft des Algorithmus Σ_0 angeben und diesen mit dem Beweis von Σ_a vergleichen. Konkret werden wir wie folgt vorgehen: Zuerst geben wir einen Beweis für Σ_a an, und dann analog einen Beweis für Σ_0 .

Wir beginnen mit dem Beweis des in Abb. 4.3.1 angegebenen abstrakteren *Mutex*-Algorithmus Σ_a .

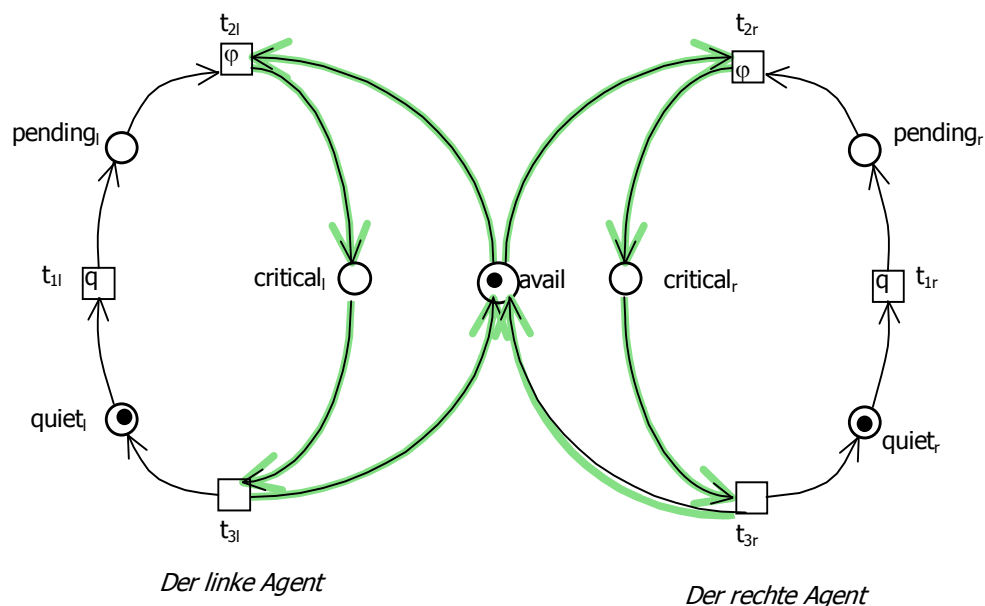


Abb. 4.3.1 System Σ_a

Wenn die Ressource gerade benutzt wird, dann wird die Ressource entweder von dem linken Agenten benutzt (sie ist also bei dem linken Agenten) oder sie wird von dem rechten Agenten benutzt (sie ist also bei dem rechten Agenten). Andernfalls, wenn die Ressource frei ist, gibt es keine Information mehr, bei welchem Agenten die Ressource ist. Das ist der wesentliche Unterschied zwischen den Algorithmen Σ_a und Σ_0 .

Die gewünschten *Mutex*-Eigenschaften

Sicherheitseigenschaft:

$$\Sigma_a \models \square \neg(\text{critical}_l \wedge \text{critical}_r) \quad (4-3-1)$$

d.h. es kommt nie vor, dass critical_l und critical_r gleichzeitig gelten.

Lebendigkeitseigenschaften:

$$\Sigma_a \models \text{pending}_l \triangleright \text{critical}_l \quad (4-3-2)$$

$$\Sigma_a \models \text{pending}_r \triangleright \text{critical}_r \quad (4-3-3)$$

d.h. für den linken Agenten gilt, wenn pending_l gilt, dann wird irgendwann critical_l gelten. Für den rechten Agenten ist es auch ähnlich.

Invarianten

Für den Beweis der Eigenschaften brauchen wir folgende Invarianten:

$$\text{quiet}_l + \text{pending}_l + \text{critical}_l = 1 \quad (4-3-4)$$

$$\text{critical}_l + \text{critical}_r + \text{avail} = 1 \quad (4-3-5)$$

(4-3-4) besagt, der linke Agent befindet sich immer in einem der drei Zustände quiet_l , pending_l und critical_l .

(4-3-5) besagt, die Ressource wird entweder von l benutzt oder von r benutzt oder sie ist frei.

Nachweis der Eigenschaften

Die Sicherheitseigenschaft (4-3-1) erhalten wir direkt mit der Invariante (4-3-5).

Die Lebendigkeitseigenschaft (4-3-2) zeigen wir im Folgenden. (4-3-3) kann man ähnlich beweisen.

Wir zeigen zunächst die folgende Eigenschaft:

$$\Sigma_a \models pending_l \triangleright avail \quad (4-3-6)$$

d.h. in einem beliebigen Zustand gilt: Wenn $pending_l$ gilt, dann wird $avail$ irgendwann gelten.

Beweis von (4-3-6): (siehe Beweisgraph in Abb. 4.3.2)

Knoten 1. Aus (4-3-4) folgt $pending_l \rightarrow \neg critical_l$.

Damit folgt aus (4-3-5) $avail \vee critical_r$.

Knoten 2. Gilt $critical_r$, ist nur t_{3r} aktiviert. Das Schalten belegt $avail$.

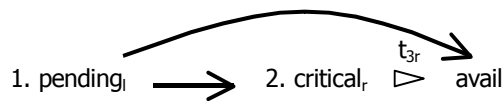


Abb.4.3.2 Beweisgraph zu (4-3-6)

Beweis von (4-3-2):

1. Transition t_{2l} ist konflikt-reduziert mit der Konflikt-Stelle $avail$.
2. (4-3-6) gilt.

Aus 1, 2 gilt (4-3-2) nach der Folgerung 2.3.7 (Fairness-Beweis-Regel).

Wir betrachten jetzt unseren Anfangsalgorithmus Σ_0 , der ein unverteilt Agentensystem ist.

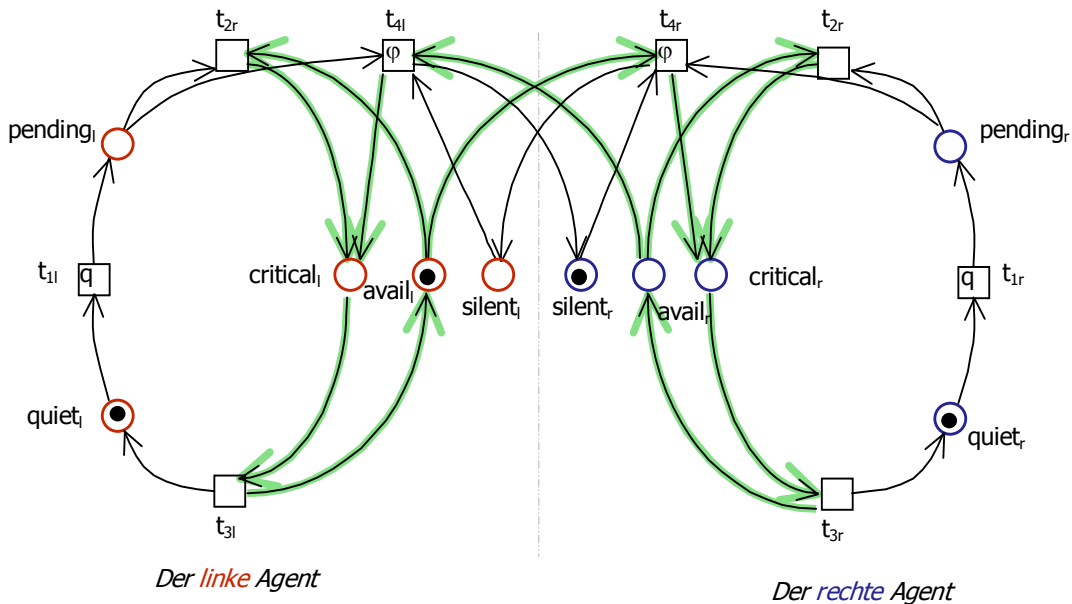


Abb. 4.3.3 System Σ_0

Die gewünschten *Mutex*-Eigenschaften

Analog zu Σ_a sind die gewünschten Eigenschaften wie folgt:
Sicherheitseigenschaft:

$$\Sigma_0 \models \square \neg(\text{critical}_l \wedge \text{critical}_r) \quad (4-3-7)$$

Lebendigkeitseigenschaften:

$$\Sigma_0 \models \text{pending}_l \triangleright \text{critical}_l \quad (4-3-8)$$

$$\Sigma_0 \models \text{pending}_r \triangleright \text{critical}_r \quad (4-3-9)$$

Bevor wir anfangen, die Korrektheit zu beweisen, möchten wir erklären, was sich im Vergleich zu Σ_a am Beweis ändern wird.

Die Beweisidee zu der *Sicherheitseigenschaft* (4-3-7) ist genau dieselbe wie bei Σ_a , also Invarianten zu verwenden. In Σ_0 sind Daten konkreter als in Σ_a . Der Zustand *avail* in Σ_a (d.h. die Ressource ist frei) wird in Σ_0 in zwei Zustände *avail_l* (d.h. die Ressource ist frei und bei *l*) und *avail_r* (d.h. die Ressource ist frei und bei *r*) geteilt. Deshalb hat die Invariante in Σ_0 entsprechend eine kleine Veränderung.

Weil die Daten konkreter sind, ist der Beweis der *Lebendigkeitseigenschaften* entsprechend auch ein bisschen komplizierter. Als ein Beispiel erklären wir die Änderung bei dem Beweis der Eigenschaft (4-3-8) (d.h. *l* wird irgendwann die Ressource benutzen, wenn er sie braucht.) im Vergleich zu Σ_a .

In Σ_a gilt: Wenn die Ressource frei ist, kann *l* sie benutzen. In Σ_0 gibt es dementsprechend zwei Fälle zu beweisen:

Fall 1.: Wenn die Ressource frei und bei ihm (bei *l*) ist, kann er sie benutzen.

Fall 2.: Wenn die Ressource frei und bei seinem Partner ist, kann er sie benutzen.

Für den 2. Fall: Der Beweis ist ganz ähnlich wie bei Σ_a . Weil es auch sein kann, dass sein Partner die Ressource benutzt, verwenden wir die Fairness-Beweis-Regel, um zu beweisen, dass *l* irgendwann die Ressource benutzen wird.

Für den 1. Fall gilt: *l* kann die Ressource benutzen. Wenn *l* die Ressource

benutzt, dann gilt sofort die Aussage (4-3-8). Es kann auch sein, dass sein Partner die Ressource benutzt. Wenn sein Partner die Ressource benutzt, dann wird irgendwann gelten, dass die Ressource frei und bei seinem Partner ist. Das ist dann wieder der 2. Fall.

Der 1. Fall ist das, was in Σ_0 mehr zu beweisen ist. Der 2. Fall ist aber entscheidend beim Beweisen dieser Eigenschaft, und die Beweisidee dafür ist genau dieselbe wie in Σ_a .

Invarianten

Ähnlich wie in Σ_a brauchen wir folgende Invarianten für den Beweis der Eigenschaften:

$$quiet_l + pending_l + critical_l = 1 \quad (4-3-10)$$

$$critical_l + critical_r + avail_l + avail_r = 1 \quad (4-3-11)$$

$$critical_l + avail_l + silent_l = 1 \quad (4-3-12)$$

(4-3-10) ist genau dieselbe wie (4-3-4) in Σ_a .

(4-3-11) ist fast gleich wie (4-3-5) in Σ_a mit einem kleinen Unterschied: statt $avail$ in (4-3-11) steht $avail_l + avail_r$, d.h. die freie Ressource ist entweder bei l oder bei r .

In Σ_0 gibt es eine Invariante (4-3-12) mehr, die besagt, für den linken Agenten gilt: die Ressource ist entweder bei ihm oder nicht bei ihm ($silent_l$ gilt). Wenn die Ressource bei ihm ist, dann benutzt er gerade die Ressource ($critical_l$ gilt), oder er benutzt die Ressource gerade nicht ($avail_l$ gilt).

Nachweis der Eigenschaften

Die Sicherheitseigenschaft (4-3-7) erhalten wir auch direkt mit der Invariante (4-3-11).

Von den Lebendigkeitseigenschaften zeigen wir auch nur (4-3-8). Die Beweislinie ist genau so wie in Σ_a .

Wir zeigen zunächst die folgende Eigenschaft:

$$\Sigma_0 \models pending_l \wedge silent_l \triangleright critical_l \quad (4-3-13)$$

d.h. wenn l die Ressource braucht und sie nicht bei ihm ist, wird l auch

irgendwann die Ressource benutzen.

Dafür zeigen wir zuerst die folgende Eigenschaft:

$$\Sigma_0 \models \text{pending}_l \wedge \text{silent}_l \triangleright \text{avail}_r \quad (4-3-14)$$

Beweis von (4-3-14): (siehe Beweisgraph in Abb. 4.3.4)

Knoten 1. Aus (4-3-12) folgt $\text{silent}_l \rightarrow \neg \text{critical}_l \wedge \neg \text{avail}_l$.

Damit folgt aus (4-3-11) $\text{avail}_r \vee \text{critical}_r$.

Knoten 2. Gilt critical_r , ist nur t_{3r} aktiviert. Das Schalten belegt avail_r .



Abb.4.3.4 Beweisgraph zu (4-3-14)

Beweis von (4-3-13):

1. Wir zeigen: Transition t_{4l} ist abgeschwächt konflikt-reduziert.

Wegen (4-3-12) gilt: $\text{pending}_l \wedge \text{silent}_l \rightarrow \neg \text{avail}_l$.

Dann gilt: t_{2l} ist nicht aktiviert, wenn $\text{pending}_l \wedge \text{silent}_l$ gilt.

Nach Def. 2.3.8 ist t_{4l} abgeschwächt konflikt-reduziert mit der Konflikt-Stelle avail_l .

2. (4-3-14) gilt.

Aus 1, 2 nach der Folgerung 2.3.11 (*Fairness*-Beweis-Regel) gilt (4-3-13).

Beweis von (4-3-8): (siehe Beweisgraph in Abb. 4.3.5)

Knoten 1. Aus (4-3-10) folgt $\text{pending}_l \rightarrow \neg \text{critical}_l$.

Damit folgt aus (4-3-12) $\text{avail}_l \vee \text{silent}_l$.

Knoten 2. avail_l verhindert t_{4l} , da $\neg \text{silent}_l$ gilt (wegen (4-3-12)).

Knoten 3. folgt aus (4-3-13).

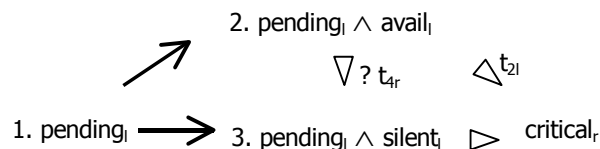


Abb.4.3.5 Beweisgraph zu (4-3-8)

4.3.2 Entwurf des Anfangsalgorithmus

In diesem Abschnitt werden wir einen Entwurfprozess zu dem Anfangsalgorithmus Σ_0 angeben. Wir werden von dem Grundmodell Σ_a (Abb. 4.3.1) ausgehen, und in zwei Schritten ($\Sigma_a \rightarrow \Sigma_b$) und ($\Sigma_b \rightarrow \Sigma_0$) das Agentensystem Σ_0 erreichen. Die Korrektheit jedes Verfeinerungsschritts lässt sich durch Beobachtungverfeinerung (siehe Abschnitt 3.1.2) beweisen. Der Anfangsalgorithmus muss bereits die algorithmische Idee des nachrichtenbasierten *Mutex* modellieren, weil die verteilenden Verfeinerungsschritte im Wesentlichen schematisch vorgenommen werden können. Anders gesagt, durch schematische verteilende Verfeinerung erhält man keine neuen algorithmischen Ideen.

Der erste Schritt

Wir stellen uns vor, der zu entwerfende Algorithmus ist für ein verteiltes System gedacht, in dem es nur zwei Agenten gibt – den linken Agenten und den rechten Agenten. Und der Algorithmus soll am Ende (nach einigen Verfeinerungsschritten) *verteilt implementierbar* sein, d.h. jede Transition, die Aktionen der beiden Agenten modelliert, muss so verfeinert werden, dass jede Transition eine Aktion eines Agenten modelliert.

Weil es außer dem linken Agenten und dem rechten Agenten keinen dritten gibt, kann die Ressource entweder bei dem linken oder bei dem rechten Agenten sein.

Die Ressource ist entweder im Zustand 'gerade benutzt werden' oder im Zustand 'frei sein'. Wenn die Ressource gerade benutzt wird, dann wird sie entweder von dem linken Agenten (Stelle $critical_l$), oder von dem rechten Agenten benutzt (Stelle $critical_r$). Wenn die Ressource frei ist, dann ist sie entweder frei und bei dem linken Agenten (Stelle $avail_l$) oder frei und bei dem rechten Agenten (Stelle $avail_r$), weil das Freigeben der Ressource und das Senden der Ressource an den Partner zwei Aktionen sind.

Deshalb zerlegen wir in diesem Schritt $\Sigma_a \rightarrow \Sigma_b$ die Stelle $avail$ zu den Stellen $avail_l$ und $avail_r$ (Abb. 4.3.6).

Entsprechend wird die Transition t_{2l} in Σ_a zu den Transitionen t_{2l} und t_{4l} in Σ_b geändert. D.h. wenn der linke Agent im Zustand $pending_l$ ist

und die Ressource frei und bei ihm ist (Stelle $avail_l$), kann er die Ressource benutzen (Transition t_{2l}). Andernfalls, wenn er die Ressource braucht und sie bei seinem Partner aber frei ist (Stelle $avail_r$), hat er auch noch das Recht, die Ressource zu benutzen (Transition t_{4l}).

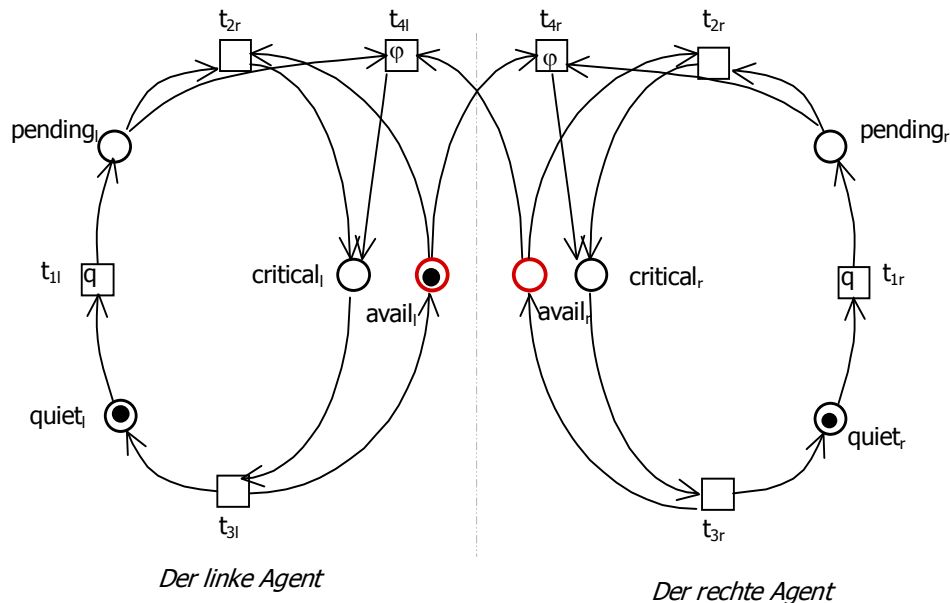


Abb. 4.3.6 Das System Σ_b (Der erste Schritt $\Sigma_a \rightarrow \Sigma_b$)

Wenn der linken Agent die Ressource fertig benutzt, wird die Ressource wieder frei. Jetzt ist noch festzulegen, bei welchem Agenten sich die Ressource befindet, wenn sie frei ist. Wir verwenden folgende *algorithmische Idee: token-passing* (die Ressource dem Partner geben) wird nur stattfinden, wenn der Partner die Ressource braucht. Deshalb steht die Ressource im Zustand $avail_l$ nach der Benutzung von dem linken Agenten (Transition t_{3l}).

Der zweite Schritt

Für den linken Agenten in Σ_b (Abb. 4.3.6) kann es sein, dass die Ressource gerade bei ihm ist: Er benutzt die Ressource (Stelle $critical_l$), oder die Ressource ist frei (Stelle $avail_l$). Es kann auch sein, dass die Ressource nicht gerade bei ihm ist. Diesen Fall werden wir mit einer Stelle $silent_l$ darstellen (Abb. 4.3.7).

Zur Steuerung des Ablaufes benötigt in einem implementierbaren Programm jeder Agent eine Variable, die anzeigt, ob er die Ressource hat ($critical$, $avail$) oder nicht bei ihm ist ($silent$). D.h. solche Variable kann genau drei Werte annehmen.

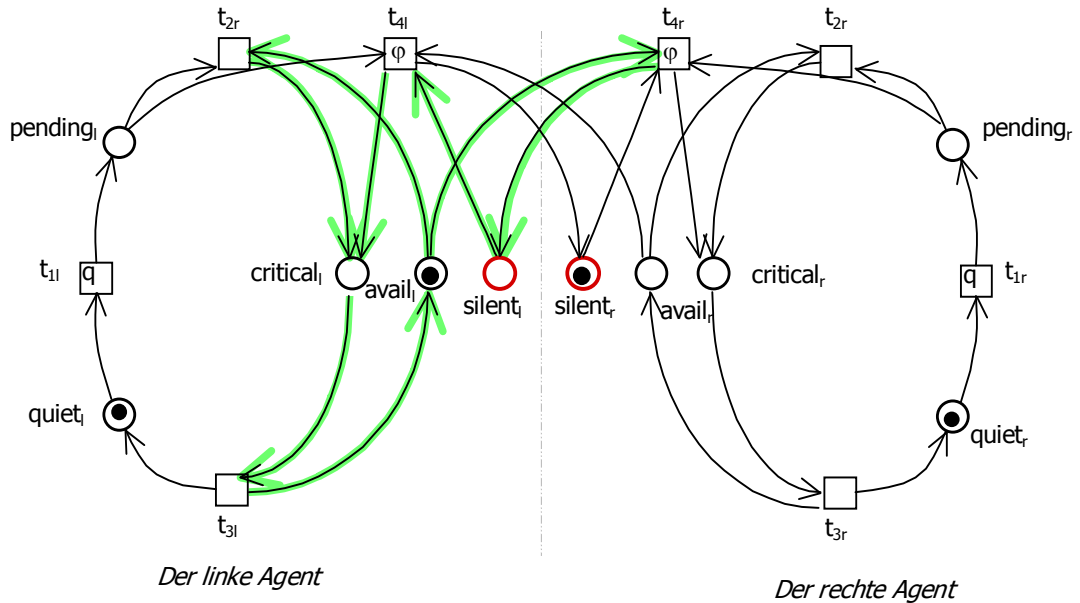


Abb. 4.3.7 Das System Σ_0 (Der zweite Schritt $\Sigma_b \rightarrow \Sigma_0$)

Wann wird die Ressource von ihm (l) zu seinem Partner fließen? Wenn die Transition t_{4r} schaltet. D.h. wenn er die Ressource nicht gerade benutzt (Stelle $avail_l$) und sein Partner sie aber braucht (Stelle $pending_r$), dann kann die Transition t_{4r} schalten. Und wenn die Transition t_{4r} schaltet, dann wird der Partner die Ressource benutzen (Stelle $critical_r$), und die Ressource ist nicht mehr bei ihm (Stelle $silent_l$).

Dann ist es klar, dass

$$avail_l + critical_l + silent_l = 1$$

eine Invariante ist. Wie in Abschnitt 4.1 erwähnt, ist die folgende Stellenmenge der Agent l :

$$\{ quiet_l, pending_l, avail_l, critical_l, silent_l \}.$$

D.h. jede Stelle des Agenten l wird durch eine Stelleninvariante von l überdeckt.

4.3.3 Nebenläufiges Programm des Anfangsalgorithmus

Um den Begriff Agentensysteme verständlicher zu machen, geben wir in diesem Abschnitt ein nebenläufiges Programm an, das dem Anfangsalgorithmus Σ_0 im Petrinetzmodell (einem Agentensystem) entspricht. Dabei implementieren wir die Invarianten $\{ quiet_l, pending_l, critical_l \}$, $\{ critical_l, avail_l, silent_l \}$, $\{ quiet_r, pending_r, critical_r \}$ und $\{ critical_r, avail_r, silent_r \}$ durch die Variablen a_l, r_l, a_r und r_r .

Die Variable a_l mit dem Wertebereich $\{ quiet_l, pending_l, critical_l \}$ beschreibt den Zustand des linken Agenten l . Sie ist eine Variable von l . r_l mit dem Wertebereich $\{ critical_l, avail_l, silent_l \}$ dient zur Steuerung der Ressource. Analog gibt es die zwei Variablen a_r und r_r des rechten Agenten r . a_r beschreibt den Zustand von r , r_r dient zur Steuerung der Ressource.

In Abb. 4.3.8 ist ein nebenläufiges Programm P_0 für den Anfangsalgorithmus Σ_0 im Petrinetzmodell angegeben.

```

parbegin
  if (  $a_l = quiet_l$  )
     $a_l := pending_l$  // Aktion  $t_{1l}$ 
  if (  $a_l = pending_l \wedge r_l = avail_l$  )
     $a_l := critical_l; r_l := critical_l$  // Aktion  $t_{2l}$ 
  if (  $a_l = pending_l \wedge r_l = silent_l \wedge r_r = avail_r$  )
     $a_l := critical_l; r_l := critical_l; r_r := silent_r$  // Aktion  $t_{4l}$ 
  if (  $a_l = critical_l \wedge r_l = critical_l$  )
     $a_l := quiet_l; r_l := avail_l$  // Aktion  $t_{3l}$ 

  if (  $a_r = quiet_r$  )
     $a_r := pending_r$  // Aktion  $t_{1r}$ 
  if (  $a_r = pending_r \wedge r_r = avail_r$  )
     $a_r := critical_r; r_r := critical_r$  // Aktion  $t_{2r}$ 
  if (  $a_r = pending_r \wedge r_r = silent_r \wedge r_l = avail_l$  )
     $a_r := critical_r; r_r := critical_r; r_l := silent_l$  // Aktion  $t_{4r}$ 
  if (  $a_r = critical_r \wedge r_r = critical_r$  )
     $a_r := quiet_r; r_r := avail_r$  // Aktion  $t_{3r}$ 
parend

```

Abb. 4.3.8 Ein nebenläufiges Programm P_0 von Σ_0

Das Programm P_0 besteht aus einigen nebenläufig ausgeführten *guarded commands*. Ein *guarded command* besteht aus einer Aktivierungsbedingung und einem Aktionsteil. Wenn die Aktivierungsbedingung eines *guarded commands* erfüllt ist, dann kann der Aktionsteil dieses *guarded commands* als eine atomare Aktion ausgeführt werden.

Die Variable a_l hat als eine Variable des Programms P_0 in einem beliebigen Zustand einen Wert aus ihrem Wertebereich, d.h. ihr Wert ist entweder $quiet_l$ oder $pending_l$ oder $critical_l$. Im Petrinetzmodell Σ_0 wird dies durch die Stellenmenge $\{ quiet_l, pending_l, critical_l \}$ modelliert, deshalb gilt:

$$quiet_l + pending_l + critical_l = 1$$

ist eine Stelleninvariante.

Analog, die Variable r_l von P_0 hat in einem beliebigen Zustand einen Wert aus ihrem Wertebereich, d.h. ihr Wert ist entweder $critical_l$ oder $avail_l$ oder $silent_l$. Im Petrinetzmodell Σ_0 wird dieses durch die Stellenmenge $\{ critical_l, avail_l, silent_l \}$ modelliert, deshalb gilt:

$$critical_l + avail_l + silent_l = 1$$

ist eine Stelleninvariante.

Wenn alle Stellen einer Invariante zu einem Agenten gehören, dann können diese Stellen im nebenläufigen Programm durch eine Variable des Agenten implementiert werden (z.B. im Programm P_0 werden durch die Variable r_l die Stellen $critical_l$, $avail_l$ und $silent_l$ im Petrinetzmodell Σ_0 implementiert). Damit sichert die Forderung, dass alle Stellen eines Agenten durch Invarianten zu überdecken sind, eine einfache Implementierbarkeit durch ein Programm¹².

¹² Wenn eine Stelle p im Petrinetzmodell nicht durch eine Stelleninvariante, die nur Stellen von einem Agenten enthält, überdeckt werden kann, sondern nur durch eine Stelleninvariante, die Stellen von mehreren Agenten enthält, überdeckt werden kann, dann kann diese Stelle p in einem Programm nur durch eine solche Variable implementiert werden, die nicht zu irgend einem Agenten alleine gehört, sondern zu dem gesamten System gehört.

5. Verteilende Verfeinerung

5.1 Warum verteilende Verfeinerung

In diesem Abschnitt werden wir anhand eines Beispiels, dem *Mutex*-Algorithmus, andeuten, wie man mit Hilfe von verteilender Verfeinerung auch einen komplizierten verteilten Algorithmus leicht verifizieren kann.

Im vorherigen Kapitel haben wir schon erwähnt, wir können den verteilten *Mutex*-Algorithmus durch Verfeinerung verifizieren, und zwar wie folgt: Wir beginnen mit einem unverteilter *Mutex*-Algorithmus Σ_0 (siehe Abb. 5.1.1) und verteilen Schritt für Schritt die unverteilter Transitionen in Σ_0 , bis wir den verteilten Zielalgorithmus Σ^* (siehe Abb. 4.1.1) erreichen. Dabei müssen wir beweisen, dass jeder Verfeinerungsschritt korrekt ist (d.h. alle gewünschten Eigenschaften des Algorithmus bleiben erhalten).

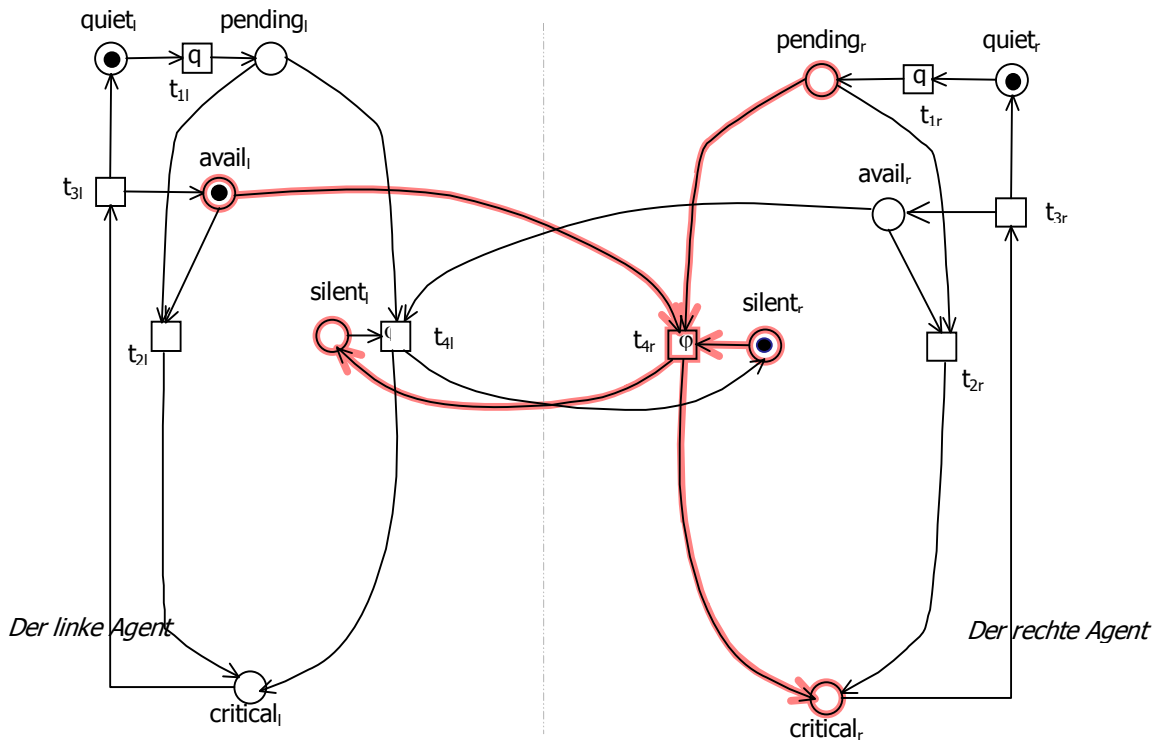


Abb. 5.1.1 Der unverteilter *Mutex*-Algorithmus Σ_0

In Σ_0 sind nur t_{4l} und t_{4r} gemeinsame Transitionen des linken und des rechten Agenten. Als Beispiel verteilen wir die Transition t_{4r} . Diese Transition beschreibt: Wenn die Ressource für den rechten Agenten nicht vorhanden ist (Stelle $silent_r$) und er sie braucht (Stelle $pending_r$), und der

linke Agent die Ressource nicht gerade benutzt (Stelle $avail_l$), dann kann die Transition t_{4r} schalten. Nach dem Schalten ist die Ressource für l nicht mehr vorhanden (Stelle $silent_l$) und r benutzt sie (Stelle $critical_r$).

Diese Transition wird zu einem Netz $N_{t_{4r}}$ verteilt (Abb. 5.1.2) und das erhaltene System ist Σ_1 (siehe Abb. 5.1.3).

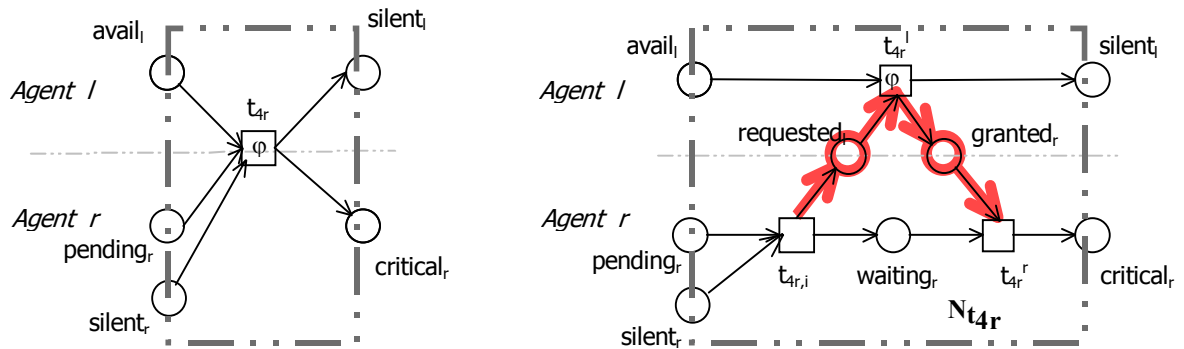


Abb. 5.1.2 Die unverteilte Transition t_{4r} und ihr verteiltes Ersetzungsnetz $N_{t_{4r}}$

Im verteilten Netz $N_{t_{4r}}$ ist es wie folgt: Wenn die Ressource nicht bei r ist und er sie braucht, dann schickt er an l eine Nachricht $requested_l$ (Transition $t_{4r,i}$).

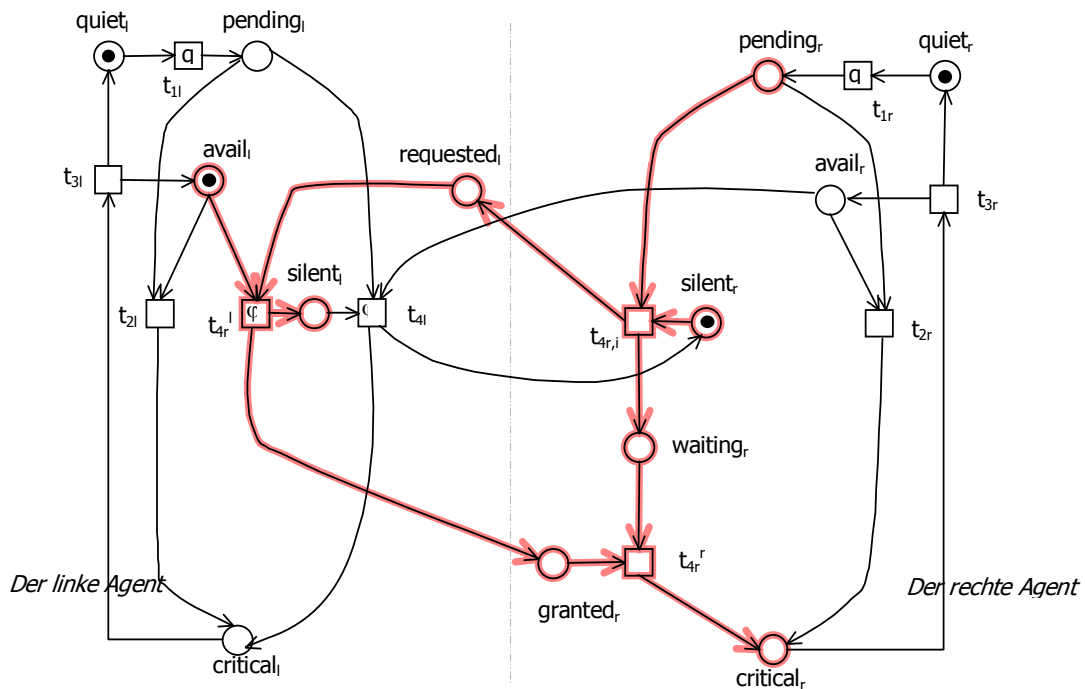


Abb. 5.1.3 Das erhaltene System Σ_1 (nach der Verteilung von t_{4r})

Wenn l diese Nachricht bekommt und er die Ressource nicht gerade benutzt, wird er (irgendwann) die Ressource abgeben (Transition t_{4r}^l). Dabei sendet er an r ein Signal $granted_r$. Wenn Agent r dieses Signal bekommt, dann benutzt er die Ressource (Transition t_{4r}^r).

Beim Verteilen von t_{4r} müssen die folgenden gewünschten Eigenschaften (siehe Abschnitt 4.3.1) erhaltenbleiben:

Lebendigkeitseigenschaft: $pending_r \triangleright critical_r$ (L)

Sicherheitseigenschaft: $\square \neg(critical_l \wedge critical_r)$ (S)

Die Blockbedingung wird durch Kausalitäten $\bullet t_{4r} \blacktriangleleft t_{4r} \bullet$ und $t_{4r,i} \blacktriangleleft t_{4r}^l$ im Ersetzungsnetz (und $fairness$ von t_{4r}^l) garantiert (vgl. Satz 6.4.2), wobei $t_{4r,i}$ und t_{4r}^l die Anfangstransitionen jeweils von r und l im Ersetzungsnetz sind. Die Kausalität $t_{4r,i} \blacktriangleleft t_{4r}^l$ impliziert: Nur wenn r die Ressource braucht, wird l sie abgeben. Dadurch wird gesichert, dass l nicht einseitig die Aktion von t_{4r} ausführt.

Die Blockbedingung und die Kausalität $\bullet t_{4r} \blacktriangleleft t_{4r} \bullet$ garantieren das Erhaltenbleiben von (L) (Lemma 3.5.9) und das Erhaltenbleiben von (S) (Lemma 3.5.12).

Also, wir brauchen nur das Ersetzungsnetz zu betrachten und zu zeigen, dass im Ersetzungsnetz die Kausalitäten $\bullet t_{4r} \blacktriangleleft t_{4r} \bullet$ und $t_{4r,i} \blacktriangleleft t_{4r}^l$ und die im Satz 6.4.2 geforderten Eigenschaften der Umgebung erfüllt sind, um zu beweisen, dass die Verteilung von t_{4r} korrekt ist. Da das Ersetzungsnetz $N_{t_{4r}}$ ein Kausalnetz ist, können entsprechend Ableseregeln Satz 2.4.4 die Kausalitäten in $N_{t_{4r}}$ direkt abgelesen werden.

5.2 Verteilende Verfeinerung

In diesem Abschnitt werden wir den Begriff – *verteilende Verfeinerung* definieren. Wir untersuchen solche Verfeinerungsschritte, in denen eine unverteilte Transition t zu einem Netz N_t verfeinert wird, das die durch die Transition t modellierten Aktionen auf Transitionen der beteiligten Agenten verteilt. Wir werden diskutieren, welche Bedingungen ein solcher Verfeinerungsschritt erfüllen soll, damit die Korrektheit des Systems bei dem Verfeinerungsschritt erhalten bleibt.

Im Kapitel 3 haben wir schon gesehen, dass eine Transitionsverfeinerung die Blockbedingung und kausale Bedingungen im Ersetzungsnetz erfüllen muss, damit gewünschte Eigenschaften des Systems erhaltenbleiben. Bei einer Verfeinerung zur Verteilung einer unverteilten Transition mehrerer Agenten sind diese kausalen Bedingungen im Ersetzungsnetz kausale Abhängigkeiten zwischen Transitionen unterschiedlicher Agenten. Die kausalen Abhängigkeiten der Aktionen der Agenten nennen wir *Synchronisationsbedingung*.

Im Folgenden werden wir zunächst *verteilende Systemtransformation* definieren und danach die Definition für die Synchronisationsbedingungen angeben und schließlich die verteilende Verfeinerung bezüglich einer Synchronisationsbedingung definieren.

5.2.1 Verteilende Systemtransformation

In diesem Abschnitt werden wir definieren, was wir als verteilende Transitionsverfeinerung in einem Agentensystem verstehen wollen.

Sei Σ ein Agentensystem mit Agentenmenge A und Kanalmenge C . Wir substituieren eine Transition t von Σ durch ein Netz N_t und erhalten ein Agentensystem Σ' mit Agentenmenge A' und Kanalmenge C' .

Wir betrachten zunächst, welche Bedingungen das Netz N_t für Σ – ein Agentensystem – noch erfüllen soll, damit wir die Transition t durch N_t ersetzen können. Danach betrachten wir, welches Agentensystem Σ' wir dann erhalten.

In Abb. 5.2.1 wird ein Agentensystem Σ (vgl. Beispiel in Abschnitt 3.1.4) gezeigt. In Σ gibt es zwei Agenten – Agenten a und Agenten b . Die roten

Stellen sind die Stellen von a , die blauen Stellen sind die von b , und die gelben Stellen sind Kanalstellen.

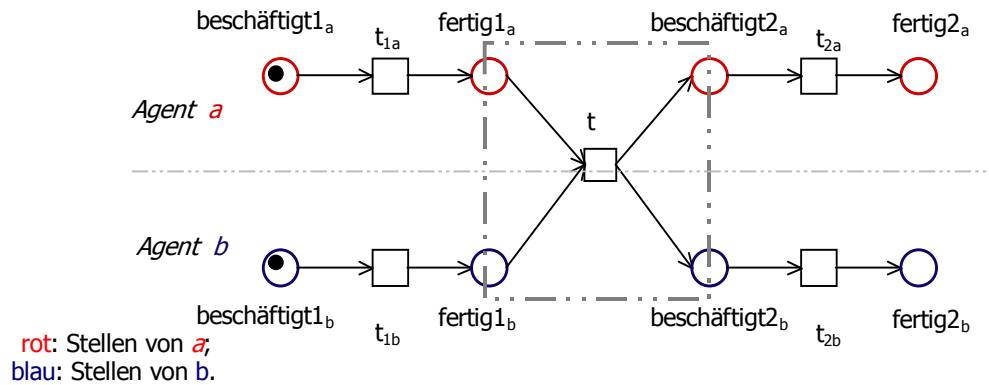


Abb. 5.2.1 Agentensystem Σ

Wir verfeinern die unverteilte Transition t zu einem verteilten Netz N_t (siehe Abb. 5.2.2).

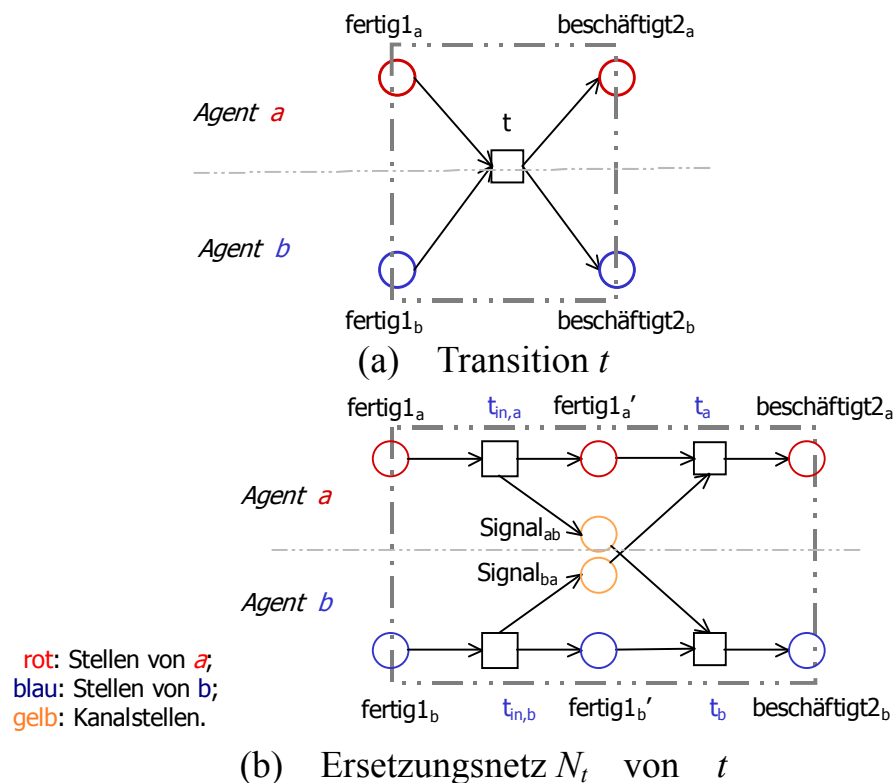


Abb. 5.2.2 Die Transition t in Σ und ihr Ersetzungsnetz

Das Teilsystem (N_b, t^-) (Abb. 5.2.3) soll auch ein Agentensystem sein und die Zuordnung der Stellen aus $\bullet t \cup t \bullet$ zu den Agenten bzw. zur Kanalmenge soll erhalten bleiben. Abb.5.2.3 veranschaulicht, dass N_t diese Bedingung erfüllt.

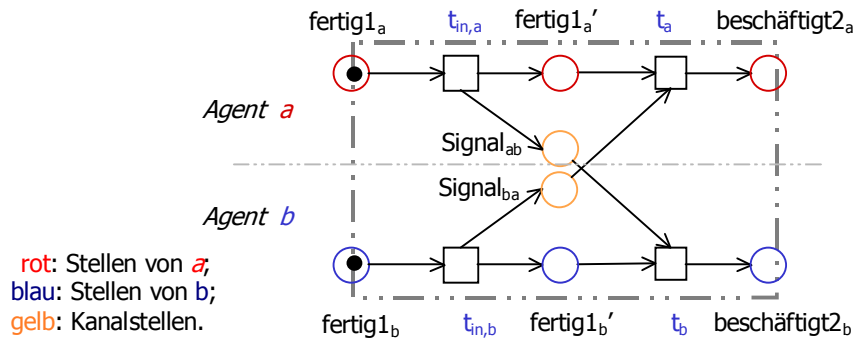


Abb. 5.2.3 System (N_t, t^-) (Verfeinerungsschritt $\Sigma \rightarrow \Sigma'$)

Bei einer formalen Definition der Forderung, dass die Zuordnung der Stellen aus $\bullet t \cup t \bullet$ zu Agenten bei der Verfeinerung erhalten bleibt, ist zu beachten, dass Agenten Stellenmengen sind. Diese Stellenmengen können sich bei der Verfeinerung vergrößern. Was gleich bleibt, ist die Zugehörigkeit zur gleichen Menge oder zu unterschiedlichen Mengen. Gehörten p und q vor der Verfeinerung zum gleichen Agenten, so gehören p und q auch nach der Verfeinerung zum gleichen Agenten.

Eine Agentenmenge A ist eine Zerlegung von Agentenstellen in eine Menge von Klassen. Diese Menge von Klassen A bilden die Äquivalenzklassen einer Äquivalenzrelation \sim_A . Zwei Stellen p, q sind äquivalent, d.h. $p \sim_A q$, gdw. sie beide zu demselben Agenten gehören. Z.B. in Σ (Abb. 5.2.1) gehören die Stellen $fertigl_a, beschäftigt2_a$ zu demselben Agenten a , deshalb gilt $fertigl_a \sim_A beschäftigt2_a$. Die Stellen $fertigl_a, fertigl_b$ gehören nicht zu demselben Agenten, deshalb gilt $\neg (fertigl_a \sim_A fertigl_b)$.

Die Zerlegung der Agentenstellen A_{N_t} von (N_t, t^-) soll der Zerlegung der Agentenstellen A von Σ nicht widersprechen, d.h. wenn zwei Stellen in Σ zu demselben Agenten gehören, dann gehören sie in (N_t, t^-) auch zu demselben Agenten. Z.B. in Σ gehören $fertigl_a$ und $beschäftigt2_a$ zu demselben Agenten, sie gehören in (N_t, t^-) auch zusammen (siehe Abb. 5.2.3), also $fertigl_a \sim_{A_{N_t}} beschäftigt2_a$.

Wenn eine Stelle im alten System vor der Verfeinerung eine Kanalstelle ist, dann ist sie in (N_t, t^-) auch eine Kanalstelle.

Agentensysteme müssen 1-beschränkt sein. Das gilt offensichtlich für die Kanalstellen und die Stellen der Agenten a und b in N_t (siehe Abb. 5.2.3).

Notation (\sim_A)

Sei A eine Zerlegung der Menge S .

Die Relation $\sim_A \subseteq S \times S$ ist wie folgt definiert:

Für $s, t \in S$, $s \sim_A t$ gdw. $\exists a \in A: s, t \in a$.

Definition 5.2.1 (Ersetzbarkeit in einem Agentensystem)

Sei Σ ein Agentensystem mit Agentenmenge A und Kanalmenge C , sei $t \in T$ eine Transition von Σ .

Sei (N_t, t^-) ein Agentensystem mit Agentenmenge A_{N_t} und Kanalmenge C_{N_t} .

t heißt im Agentensystem Σ durch N_t ersetzbar, gdw.

- i. t ist im System Σ durch N_t ersetzbar;
- ii. Für jedes $p, q \in \bullet t \cup t^\bullet$ gilt:
 - $p \in C$ gdw. $p \in C_{N_t}$
(d.h. p ist in Σ eine Kanalstelle gdw. sie in (N_t, t^-) eine Kanalstelle ist);
 - $p \sim_A q$ gdw. $p \sim_{A_{N_t}} q$
(d.h. p, q gehören in Σ zu demselben Agenten gdw. sie in (N_t, t^-) zu demselben Agenten gehören). □

Wir ersetzen die Transition t in Σ durch N_t und erhalten das neue System Σ' (Abb.5.2.4).

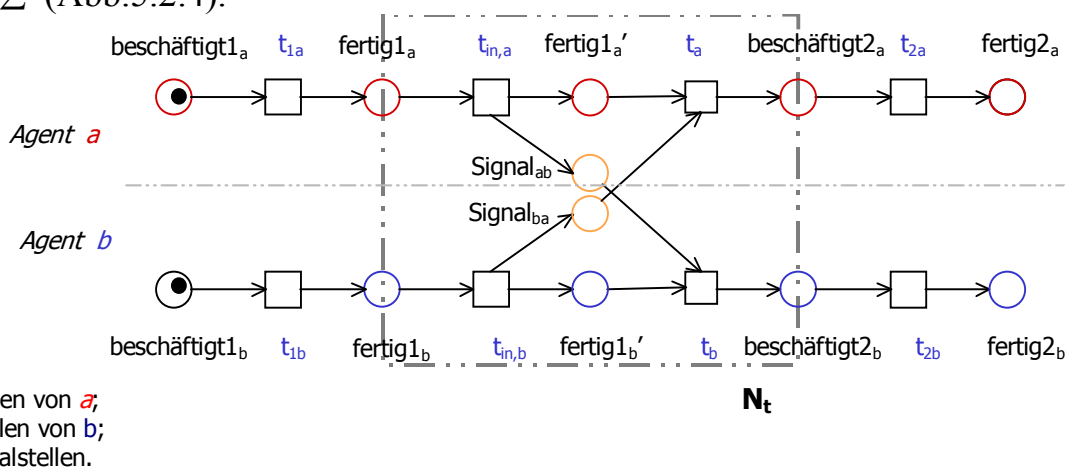


Abb. 5.2.4 Das erhaltene Agentensystem Σ' durch Ersetzen von t im Agentensystem Σ durch N_t

Die Agentenmenge A' von Σ' berechnen wir wie folgt: Zunächst ist eine Äquivalenzrelation \sim' aus der Äquivalenzrelation \sim_A und der Äquivalenzrelation $\sim_{A_{N_t}}$ abzuleiten (und zwar ist $\sim' =$ *die transitive Hülle von* $\sim_A \cup \sim_{A_{N_t}}$). Und dann ist aus dieser Äquivalenzrelation die Zerlegung A' zu definieren. A' ist die Agentenmenge des neuen Systems Σ' .

Die Kanalmenge C' von Σ' besteht aus den Kanalstellen des alten Systems Σ und den Kanalstellen des Ersatznetzes (N_t, t^-) , d.h. $C' = C \cup C_{N_t}$.

Wir sagen, das System Σ' mit Agenten und Kanälen ist eine $t \rightarrow N_t$ Substitution des Agentensystems Σ und $\Sigma \rightarrow \Sigma'$ ist eine Agentensystem-Transformation vom Agentensystem Σ nach System Σ' . Weil t unverteilt und N_t verteilt ist, nennen wir $\Sigma \rightarrow \Sigma'$ eine *verteilende Systemtransformation*.

Σ' ist im Allgemeinen nur ein System mit Agenten und Kanälen, weil Σ' nicht 1-beschränkt sein muss.

Notation (A_\sim)

Sei $\sim \subseteq S \times S$ eine Äquivalenzrelation.

Für $s \in S$, $[s] =_{\text{def}} \{ t \in S \mid s \sim t \}$.

$A_\sim =_{\text{def}} \{ [s] \mid s \in S \}$.

Für $\Sigma(N_t \setminus t)$ sind das neue Netz, die Agentenmenge und die Kanalmenge zu definieren.

Definition 5.2.2 ($t \rightarrow N_t$ Substitution in einem Agentensystem und Verteilende Systemtransformation)

Sei Σ ein Agentensystem mit Agentenmenge A und Kanalmenge C , sei $t \in T$ eine Transition von Σ .

Sei (N_t, t^-) ein Agentensystem mit Agentenmenge A_{N_t} und Kanalmenge C_{N_t} mit: t ist im Agentensystem Σ durch N_t ersetzbar.

Ein Agentensystem Σ' mit Agentenmenge A' und Kanalmenge C' heißt eine $t \rightarrow N_t$ Substitution des Agentensystems Σ gdw.

- System Σ' ist eine $t \rightarrow N_t$ Substitution des Systems Σ ;

- $C' = C \cup C_{N_t}$;
- $A' = A_{\sim'}$, wobei: $\sim' =_{def}$ die transitive Hülle von $(\sim_A \cup \sim_{A_{N_t}})$.

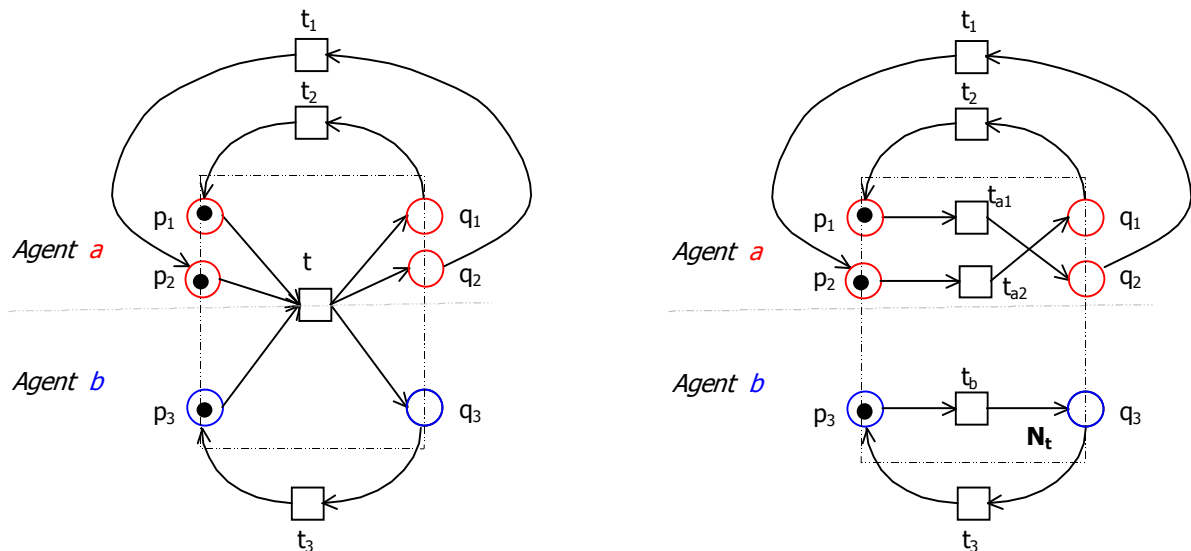
Notation: $\Sigma' = \Sigma(N_t \setminus t)$.

$\Sigma \rightarrow \Sigma'$ heißt die Agentensystem-Transformation von Agentensystem Σ .

$\Sigma \rightarrow \Sigma'$ heißt eine verteilende Systemtransformation, wenn t unverteilt und N_t verteilt ist.

□

Auf den ersten Blick könnte man meinen, dass eine $t \rightarrow N_t$ Ersetzung eines Agentensystems Σ wieder ein Agentensystem ist, wenn N_t ein Agentensystem ist. Das folgende Gegenbeispiel zeigt, dass dies nicht immer gilt



(a) System Σ (vor der Verfeinerung) (b) System Σ' (nach der Verfeinerung)
Abb. 5.2.5 Gegenbeispiel 1

Durch die Ersetzung von t durch N_t entsteht ein System Σ' , das nicht 1-beschränkt ist, also kein Agentensystem sein kann. Nach dem Schalten von t_{a2} und t_2 liegen auf p_1 zwei Marken (siehe Abb. 5.2.6).

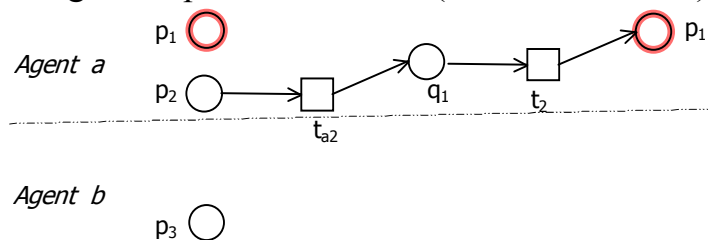


Abb. 5.2.6 Anfang eines Ablaufs von Σ'

Die Ursache dafür, dass Σ' kein Agentensystem ist, liegt darin begründet, dass die Invarianten, die in Σ und (N_b, t^-) die Agenten überdecken, sich nicht ergänzen. In Σ haben wir für den Agenten a die folgenden Invarianten:

$$p_1 + q_1 = 1,$$

$$p_2 + q_2 = 1.$$

In (N_b, t^-) gelten die Invarianten:

$$p_1 + q_2 = 1,$$

$$p_2 + q_1 = 1.$$

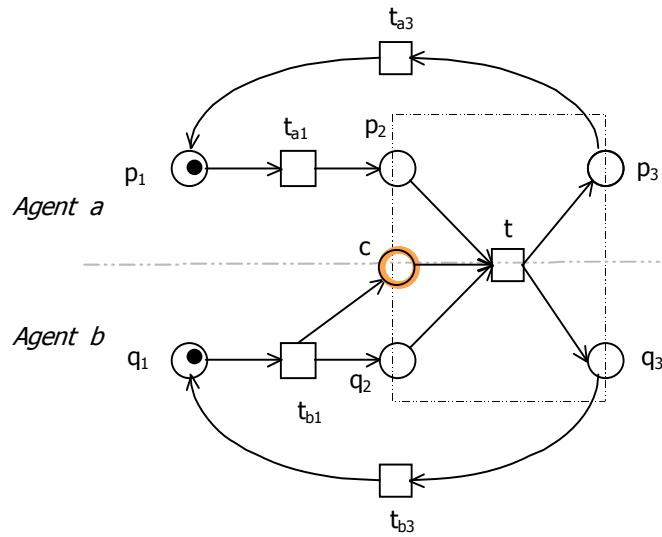
Damit eine Invariante I_1 der Umgebung vereinigt mit einer Invariante I_2 des Verfeinerungsnetzes eine neue Invariante des verfeinerten Netzes mit dem Wert 1 ergibt, muss gelten z.B. nach [Kin95]: Die Invarianten I_1 und I_2 enthalten die gleichen Stellen aus dem Vor- und Nachbereich von t , also

$$(\bullet t \cup t \bullet) \cap I_1 = (\bullet t \cup t \bullet) \cap I_2.$$

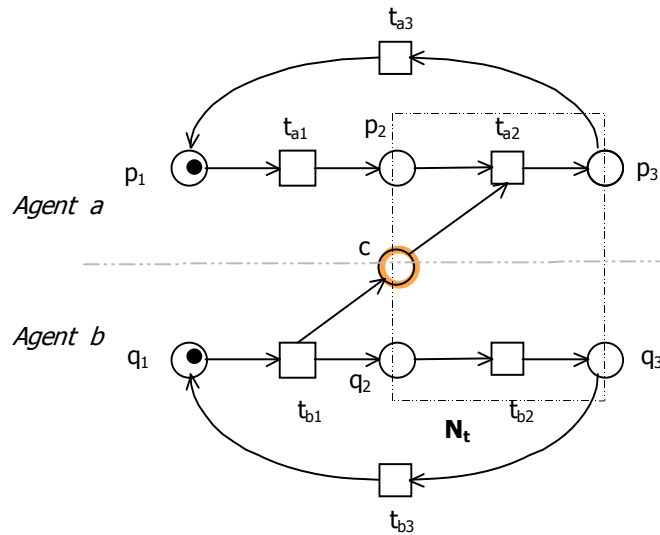
Dieses Problem tritt nicht auf, wenn in (N_b, t^-) zusätzlich $\bullet t \blacktriangleleft t \bullet$ gilt. Dann kann p_1 nicht mehr markiert sein, wenn q_1 markiert wird.

Wenn das verfeinerte System ein Agentensystem sein soll muss weiter gelten, dass auch alle Kanalstellen 1-beschränkt sind. Wenn wir die Voraussetzung, dass die Kanalstellen in Σ und (N_b, t^-) 1-beschränkt sind verschärfen und fordern, dass alle Kanalstellen durch Invarianten mit dem Wert 1 zu überdecken sind, dann reicht das immer noch nicht. Folgendes Beispiel (Abb. 5.2.7) zeigt, dass auch unter dieser verschärfen Voraussetzung ein verfeinertes System erhalten werden kann, in dem Kanalstellen nicht mehr 1-beschränkt sind.

Durch die Ersetzung von t durch N_t entsteht ein System Σ'_6 , das nicht 1-beschränkt ist, also kein Agentensystem sein kann. Nach dem Schalten von t_{b1} , t_{b2} , t_{b3} und t_{b1} liegen auf der Kanalstelle c zwei Marken (siehe Abb. 5.2.8).



(a) System Σ_6 (vor der Verfeinerung)



(b) System Σ'_6 (nach der Verfeinerung)

Abb. 5.2.7 Gegenbeispiel 2

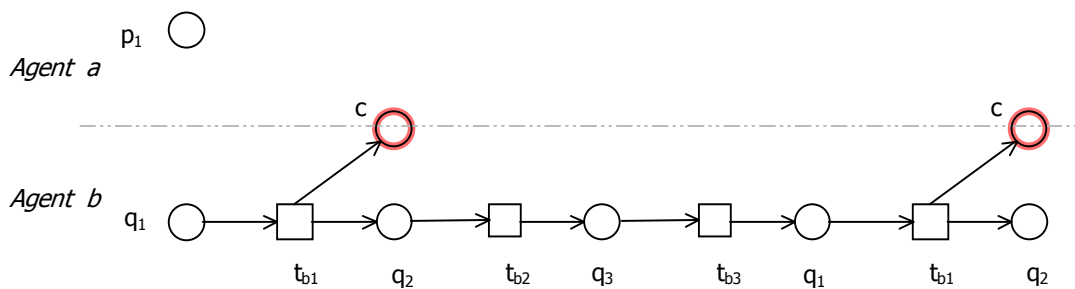


Abb. 5.2.8 Anfang eines Ablaufs von Σ'_6

Der Grund dafür, dass die Kanalstelle c nicht 1-beschränkt ist, ist der Folgende: Die Invarianten, die in Σ_6 und (N_t, t^-) die Kanalstelle c enthalten, passen nicht zusammen.

In Σ_6 gilt nur die folgende Invariante I , die c enthält:

$$q_1 + c + q_3 = 1.$$

Aber in (N_b, t^-) gilt nur die folgende Invariante I' , die c enthält:

$$c + p_3 = 1.$$

Dann gilt offensichtlich wegen $\bullet t \cup t \bullet = \{c, p_2, p_3, q_2, q_3\}$:

$$I \cap (\bullet t \cup t \bullet) \neq I' \cap (\bullet t \cup t \bullet).$$

Satz 5.2.3

Sei $\Sigma' = \Sigma(N_t \setminus t)$ eine $t \rightarrow N_t$ Ersetzung des Agentensystems Σ durch das Netz N_t , dann ist Σ' ein Agentensystem, wenn (N_b, t^-) ein Agentensystem ist und es Invarianten I_1, \dots, I_k gibt, die die Agenten und die Kanalstellen von Σ überdecken und Invarianten I'_1, \dots, I'_l von (N_b, t^-) , die die Agenten und Kanalstellen von N_t überdecken, so dass gilt: Zu jedem $I' \in \{I'_1, \dots, I'_l\}$ mit $I' \cap (\bullet t \cup t \bullet) \neq \emptyset$ gibt es ein $I \in \{I_1, \dots, I_k\}$ mit $I \cap (\bullet t \cup t \bullet) = I' \cap (\bullet t \cup t \bullet)$ und umgekehrt (zu jedem $I \in \{I_1, \dots, I_k\}$ mit $I \cap (\bullet t \cup t \bullet) \neq \emptyset$ gibt es ein $I' \in \{I'_1, \dots, I'_l\}$ mit $I \cap (\bullet t \cup t \bullet) = I' \cap (\bullet t \cup t \bullet)$).

Beweis:

Es ist nachzuweisen: 1. Die Agenten von Σ' lassen sich durch Invarianten mit Wert 1 überdecken; 2. Die Kanalstellen von Σ' sind alle 1-beschränkt. Zu 1.: Gilt wegen der Forderung an die Invarianten.

Zu 2.: Gilt entsprechend. □

Bemerkung 5.2.4 (Erhaltenbleiben der 1-Beschränktheit)

$\Sigma' = \Sigma(N_t \setminus t)$ erfülle die Blockbedingung.

Gilt $(N_b, t^-) \models \bullet t \blacktriangleleft t \bullet$, so bleiben alle Stellen aus Σ 1-beschränkt.

Beweis:

Sei $p \in P$ eine beliebige Stelle von Σ und in Σ 1-beschränkt. (1)

Zu zeigen ist: p ist auch in Σ' 1-beschränkt.

Seien $(K, r), (K', r')$ Abläufe von Σ, Σ' mit: (K', r') ist eine $t \rightarrow N_t$

Substitution von (K, r) , wobei $K = (B, E, \leq), K' = (B', E', \leq')$.

Angenommen, p ist in Σ' nicht 1-beschränkt.

Seien $b_1, b_2 \in B'$ mit $r'(b_1) = r'(b_2) = p$ und $b_1 \text{ co } b_2$. (2)

Dann gilt: $b_1, b_2 \in B$ und $r(b_1) = r(b_2) = p$. (wegen Bem. 3.2.5)

Wegen (2) gilt:

Auch in K gilt $b_1 \underline{co} b_2$, weil:

Wenn in K $b_1 \leq^+ b_2$ gilt, dann gilt auch in K' $b_1 \leq'^+ b_2$. (wegen Satz 3.5.1)

Also, p ist in Σ nicht 1-beschränkt. Das ist ein Widerspruch zu (1).

Daraus folgt, p ist in Σ' 1-beschränkt. \square

Der folgende Satz zeigt den Zusammenhang zwischen Invarianten mit dem Wert 1 und der kausalen Halbordnung in Abläufen.

Sei $p_1 + p_2 + \dots + p_k = 1$ eine Invariante, d.h. in jedem Schnitt eines Ablaufs ist die Anzahl der markierten Stellen aus $\{p_1, \dots, p_k\}$ gleich 1.

Satz 5.2.5

$p_1 + p_2 + \dots + p_k = 1$ ist eine Invariante, gdw. es in jedem Ablauf (K, r) mit (B, E, \leq) genau einen Weg W von ${}^\circ K$ bis K° oder unendlich gibt mit für alle $b \in B$ mit $r(b) \in \{p_1, \dots, p_k\}$ ist $b \in W$. Die Ereignisse $b \leq e$ mit $r(b) \in \{p_1, \dots, p_k\}$ bilden also eine Kausalkette.

Beweis:

In der Anfangsmarkierung von Σ ist genau eine Stelle aus $\{p_1, \dots, p_k\}$ belegt (wegen $p_1 + \dots + p_k = 1$). Also existiert genau ein $b_1 \in {}^\circ K$ mit $r(b_1) \in \{p_1, \dots, p_k\}$. Ein Ablauf entsteht durch das Schalten von Transitionen. Angenommen wir haben schon einen Schnitt C erreicht, zu dem es genau einen Weg mit den geforderten Eigenschaften gibt, der in $b_n \in C$ endet. Schaltet als nächstes eine Transition t mit $\bullet t \cap \{p_1, \dots, p_k\} = \emptyset$ so enthält der Folgeschnitt weiter b_n und die geforderte Eigenschaft gilt weiter. Gilt $\bullet t \cap \{p_1, \dots, p_k\} = r(b_n)$, so gibt es genau ein $p_j \in \bullet t \cap \{p_1, \dots, p_k\}$, sonst wären nach dem Schalten von t keine Stelle oder mindestens zwei Stellen der Invariante belegt. Der gesuchte Weg verlängert sich um das Ereignis e mit $r(e) = t$ und die Bedingung b_{n+1} mit $r(b_{n+1}) = p_j$. \square

5.2.2 Synchronisations-Bedingung und verteilende Verfeinerung

In diesem Abschnitt werden wir den Begriff *Synchronisationsbedingung* einer unverteilter Transition und den Begriff *verteilende Verfeinerung* bzgl. einer Synchronisationsbedingung angeben.

Synchronisationsbedingungen sind spezielle Kausalitäten im System. Wir werden zunächst diese Art der Kausalitäten vorstellen.

1. Kausalitäten zwischen Vor- bzw. Nachbereichen von Stellen im System

Wir werden anhand einfacher Beispiele die Kausalitäten im System erläutern, die wir hier einführen werden.

In Abb. 5.2.9 wird das System *Sicherer Wasserspringer* aus der Einleitung (siehe Abb. 1.10) nochmals gezeigt. In diesem System muss der Hallenwart jedesmal zuerst das Wasser einlassen (Transition *Wasser-einlassen*). Erst danach kann der Wasserspringer Training beginnen (Transition *Training-beginnen*).

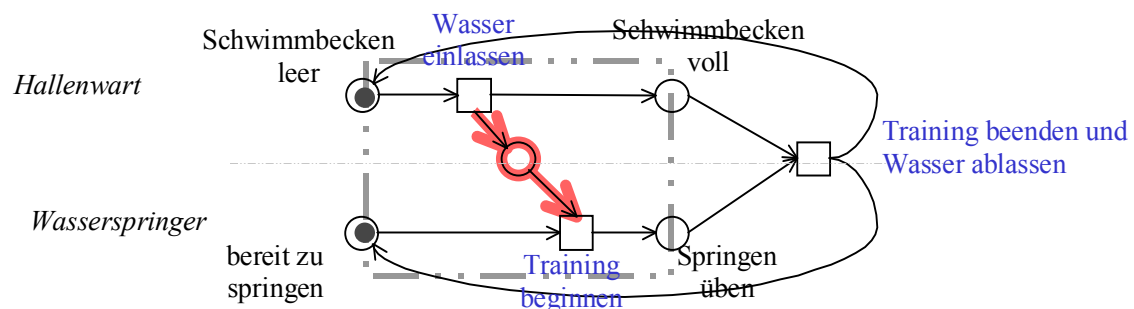


Abb. 5.2.9 *Sicherer Wasserspringer*

In Abb. 5.2.10 ist der Ablauf dieses Systems dargestellt. In diesem Ablauf gibt es zu jedem Auftreten von *Training-beginnen* ein Auftreten von *Wasser-einlassen*. Also in diesem Ablauf gilt

Wasser-einlassen \blacktriangleleft *Training-beginnen* (wegen Def. 2.4.1).

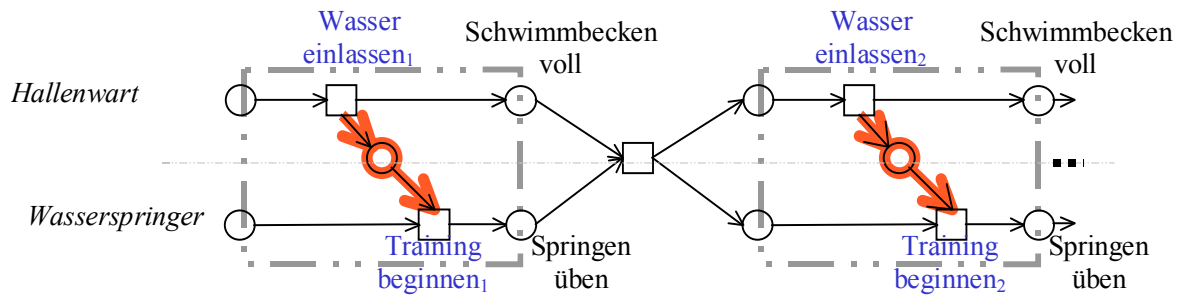


Abb. 5.2.10 Ablauf des Systems *Sicherer Wasserspringer* (Abb. 5.2.9)
 (Illustration für *Wasser-einlassen* \blacktriangleleft *Training-beginnen* im Ablauf)

In diesem System ist die Aktion *Wasser-einlassen* die einzige Aktion, die *Schwimmbecken-voll* macht (d.h.

•*Schwimmbecken-voll* = { *Wasser-einlassen* }). Die Aktion *Training-beginnen* ist die einzige Aktion, die *Springen-üben* zulässt (d.h.

• *Springen-üben* = { *Training-beginnen* }). Deshalb sagen wir auch: In diesem Ablauf gilt •*Schwimmbecken-voll* \blacktriangleleft •*Springen-üben* (siehe Abb. 5.2.11).

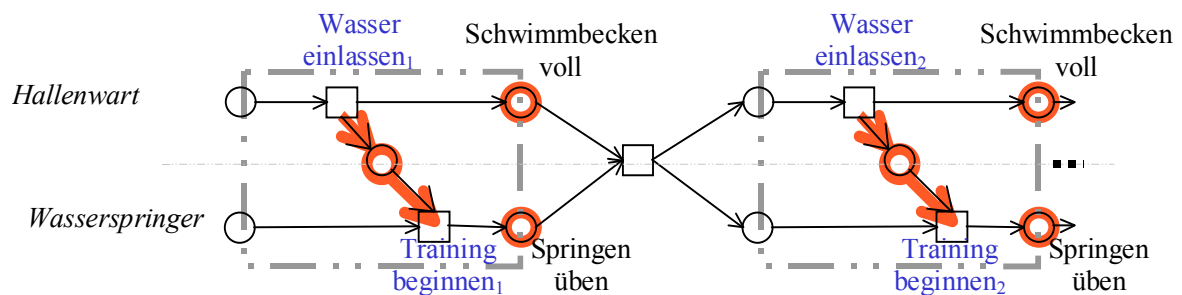


Abb. 5.2.11 Ablauf des Systems *Sicherer Wasserspringer* (Abb. 5.2.9)
 (Illustration für •*Schwimmbecken-voll* \blacktriangleleft •*Springen-üben* im Ablauf)

Im System in Abb. 5.2.12 kann der *Hallenwart* mit einer (roten) Maschine das Wasser einlassen, er kann auch mit einer anderen (gelben) Maschine das Wasser einlassen. Das heisst, es gibt zwei Aktionen, die das *Schwimmbecken voll* machen (also $|\bullet \text{Schwimmbecken-voll}| > 1$).

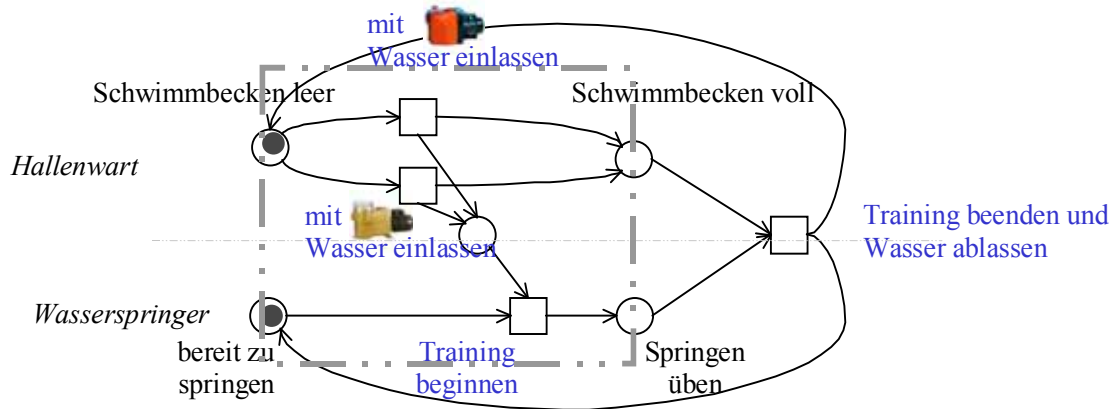


Abb. 5.2.12 *Sicherer Wasserspringer mit zwei Maschinen*

In Abb. 5.2.13 ist ein Ablauf des Systems *Sicherer Wasserspringer mit zwei Maschinen* angegeben. In diesem Ablauf hat der *Hallenwart* beim ersten Training mit der roten Maschine das Wasser eingelassen, beim zweiten Mal mit der gelben.

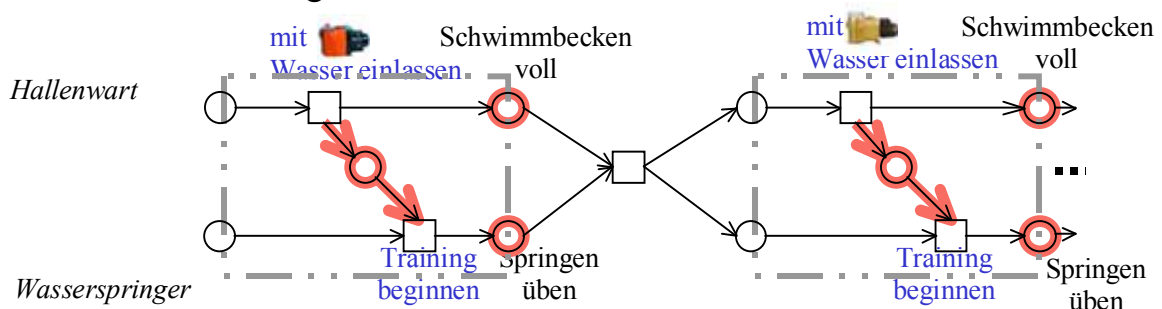


Abb. 5.2.13 Ein Ablauf des Systems *Sicherer Wasserspringer mit zwei Maschinen* (Abb. 5.2.12)

In diesem Ablauf gilt:

$\neg (\text{mit } \text{[red machine icon]} \text{ Wasser einlassen} \blacktriangleleft \text{Training-beginnen})$,
weil der *Hallenwart* beim zweiten Training nicht mit der roten Maschine das Wasser eingelassen hat.

In diesem Ablauf gilt auch:

$\neg (\text{mit } \text{[yellow machine icon]} \text{ Wasser einlassen} \blacktriangleleft \text{Training-beginnen})$,
weil der *Hallenwart* beim ersten Training nicht mit dieser Maschine das Wasser eingelassen hat.

Es ist eigentlich egal, mit welcher Maschine das Wasser eingelassen wurde. Hauptsache, das Schwimmbecken ist voll. Deshalb sagen wir, in diesem Ablauf gilt:

\bullet Schwimmbecken-voll \blacktriangleleft \bullet Springen-üben.

Nun können wir Kausalitäten der Form $\bullet p \blacktriangleleft \bullet q$ des Systems definieren.

Definition 5.2.6 ($K \models \bullet p \blacktriangleleft \bullet q$)

Sei $\Sigma = ((P, T, F), m_0)$ ein System, (K, r) ein Ablauf von Σ .

Weiter seien $p, q \in P$.

$\bullet p \blacktriangleleft \bullet q$ gilt in K (Notation: $K \models \bullet p \blacktriangleleft \bullet q$), gdw. zu jedem $e \in K$ mit

$r(e) \in \bullet q$ gibt es ein $e' \in K$ mit $r(e') \in \bullet p$ mit:

- entweder $e' = e$;
- oder $e' < e$ und für jedes $e'' \in K$ mit $e' < e'' < e$ gilt $r(e'') \notin \bullet q$.

Definition 5.2.7 ($\Sigma \models \bullet p \blacktriangleleft \bullet q$)

Sei $\Sigma = ((P, T, F), m_0)$ ein System, seien $p, q \in P$.

$\bullet p \blacktriangleleft \bullet q$ gilt in Σ (Notation: $\Sigma \models \bullet p \blacktriangleleft \bullet q$), gdw. in jedem Ablauf $\bullet p \blacktriangleleft \bullet q$ gilt.

In beiden Systemen in Abb. 5.2.9 und Abb. 5.2.12 gilt

\bullet Schwimmbecken-voll \blacktriangleleft \bullet Springen-üben.

Allgemein gibt es zwischen zwei Stellen p und q die Relationen $p \bullet \blacktriangleleft q \bullet$,

$\bullet p \blacktriangleleft \bullet q$, $p \bullet \blacktriangleleft \bullet q$ und $\bullet p \blacktriangleleft q \bullet$.

Definition 5.2.8 ($K \models x \blacktriangleleft y$)

Sei $\Sigma = ((P, T, F), m_0)$ ein System, (K, r) ein Ablauf von Σ .

Weiter seien $p, q \in P$ und $x = \bullet p$ oder $x = p \bullet$ und $y = \bullet q$ oder $y = q \bullet$.

$x \blacktriangleleft y$ gilt in K (Notation: $K \models x \blacktriangleleft y$), gdw. zu jedem $e \in K$ mit $r(e) \in y$ gibt

es ein $e' \in K$ mit $r(e') \in x$ mit:

- entweder $e' = e$;
- oder $e' < e$ und für jedes $e'' \in K$ mit $e' < e'' < e$ gilt $r(e'') \notin y$.

Definition 5.2.9 ($\Sigma \models x \blacktriangleleft y$)

Sei $\Sigma = ((P, T, F), m_0)$ ein System.

Weiter seien $p, q \in P$ und $x = \bullet p$ oder $x = p \bullet$ und $y = \bullet q$ oder $y = q \bullet$.

$x \blacktriangleleft y$ gilt in Σ (Notation: $\Sigma \models x \blacktriangleleft y$), gdw. in jedem Ablauf $x \blacktriangleleft y$ gilt.

Im vorherigen Beispiel *Sicherer Wasserspringer* (siehe Abb. 5.2.9) gelten:

$$|\bullet \text{Schwimmbecken-voll}| = |\{\text{Wasser-einlassen}\}| = 1 \text{ und}$$

$$|\bullet \text{Springen-üben}| = |\{\text{Training-beginnen}\}| = 1.$$

Deshalb gilt in diesem System:

$$\bullet \text{Schwimmbecken-voll} \blacktriangleleft \bullet \text{Springen-üben} \text{ gdw.}$$

$$\text{Wasser-einlassen} \blacktriangleleft \text{Training-beginnen}.$$

Bemerkung 5.2.10

Sei $\Sigma = ((P, T, F), m_0)$ ein System.

Weiter seien $p, q \in P$, sei $x = \bullet p$ oder $x = p \bullet$, sei $y = \bullet q$ oder $y = q \bullet$.

Seien $x = \{t\}$, $y = \{t'\}$, wobei $t, t' \in T$.

Dann gilt:

$$\Sigma \models x \blacktriangleleft y \iff (t=t') \vee \Sigma \models t \blacktriangleleft t'.$$

Das System *vollkommen verteilter Wasserspringer* (Abb. 5.2.14) veranschaulicht den Zusammenhang zwischen Kausalitäten im Netz und temporal logischen Eigenschaften des Systems.

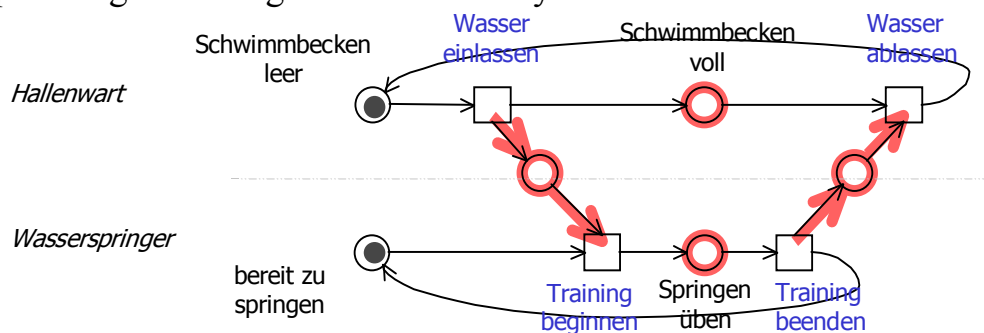


Abb. 5.2.14 System *vollkommen verteilter Wasserspringer*

In diesem System Σ gelten die folgenden Kausalitäten:

$$\Sigma \models \bullet \text{Schwimmbecken-voll} \blacktriangleleft \bullet \text{Springen-üben} \text{ und}$$

$$\Sigma \models \text{Springen-üben} \bullet \blacktriangleleft \text{Schwimmbecken-voll} \bullet.$$

Diese Kausalitäten sichert die folgende Eigenschaft:

$$\Sigma \models \square \quad (\text{Springen-üben} \rightarrow \text{Schwimmbecken-voll}).$$

2. Synchronisations-Bedingung einer unverteilter Transition

Nach dem Verteilen einer unverteilter Transition t soll neben der Blockbedingung auch noch garantiert werden, dass die für die gewünschten Eigenschaften eines Systems nötigen kausalen Anhängigkeiten der Aktionen unterschiedlicher Agenten erhalten bleiben.

Für eine Transition t mit $\bullet t = \{p_a, p_b\}$ und $t \bullet = \{q_a, q_b\}$ (vgl. Abb. 5.2.15) gelten folgende kausale Abhängigkeiten: $p_a \bullet \blacktriangleleft p_b \bullet$, $p_a \bullet \blacktriangleleft \bullet q_b$, $\bullet q_a \blacktriangleleft \bullet q_b$, $p_b \bullet \blacktriangleleft p_a \bullet$, $p_b \bullet \blacktriangleleft \bullet q_a$ und $\bullet q_b \blacktriangleleft \bullet q_a$.

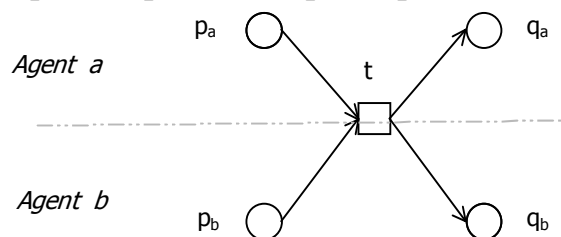


Abb. 5.2.15 Illustration für Synchronisationsbedingungen (eine isoliert dargestellte unverteilter Transition t)

Alle diese Kausalitäten beschreiben Reihenfolgebeziehungen zwischen Aktionen, die Stellen unterschiedlicher Agenten belegen. Wir nennen sie *Synchronisationsbedingungen*. Bei der Verfeinerung einer Transition t durch ein Netz N_t sollen im lokalen Teilsystem (N_t, t^-) einige gewünschte Synchronisationsbedingungen erhalten bleiben andere können wegfallen.

Im Beispiel Σ (siehe Abb. 5.2.1) soll der Agent a erst die nächste Phase seiner Arbeit beginnen, wenn sein Partner – der Agent b – auch schon mit der aktuellen Phase fertig ist.

Im System Σ_1' in Abb. 5.2.16 wird die Transition t im System Σ zu einem verteilten Netz NI_t verteilt. In diesem Ersetzungsnetz NI_t kann der Agent a die zweite Phase beginnen, auch wenn der Agent b noch nicht fertig mit der ersten Phase ist. In Σ_1' kann so ein Zustand vorkommen, dass a in der zweiten Phase aber b erst in der ersten Phase steht.

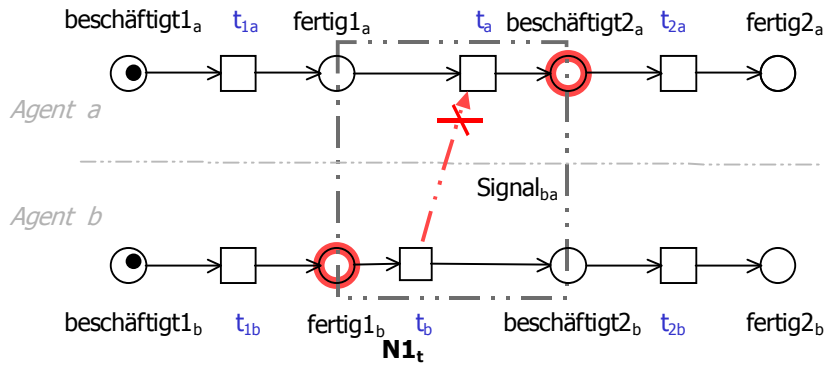


Abb. 5.2.16 Eine weitere Verfeinerung Σ_1' von Σ (siehe Abb. 5.2.1)
 (Der rote Pfeil zeigt die fehlende Kausalität unterschiedlicher Agenten)

Im System Σ_2' in Abb. 5.2.17 wird t zu einem anderen Netz $N2_t$ verteilt. Im Ersetzungsnetz $N2_t$ muss a warten, bis b mit der ersten Phase fertig ist, erst danach kann a die zweite Phase beginnen. Im Teilsystem $(N2_t, t^-)$ gilt die Kausalität $fertig1_b \bullet \blacktriangleleft \bullet$ beschäftigt2_a.

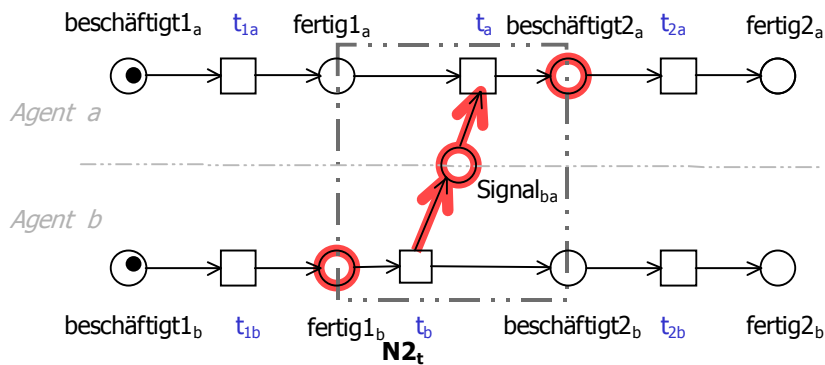


Abb. 5.2.17 Noch eine Verfeinerung Σ_2' von Σ (siehe Abb. 5.2.1)

Für das Ersetzungsnetz N_t (siehe Abb. 5.2.18) gilt in (N_t, t^-) nicht nur die Kausalität $fertig1_b \bullet \blacktriangleleft \bullet$ beschäftigt2_a sondern auch die Kausalität $fertig1_a \bullet \blacktriangleleft \bullet$ beschäftigt2_b. In diesem Ersetzungsnetz müssen die beiden Agenten warten, bis beide mit der ersten Phase fertig sind. Erst danach können die beiden die zweite Phase beginnen.

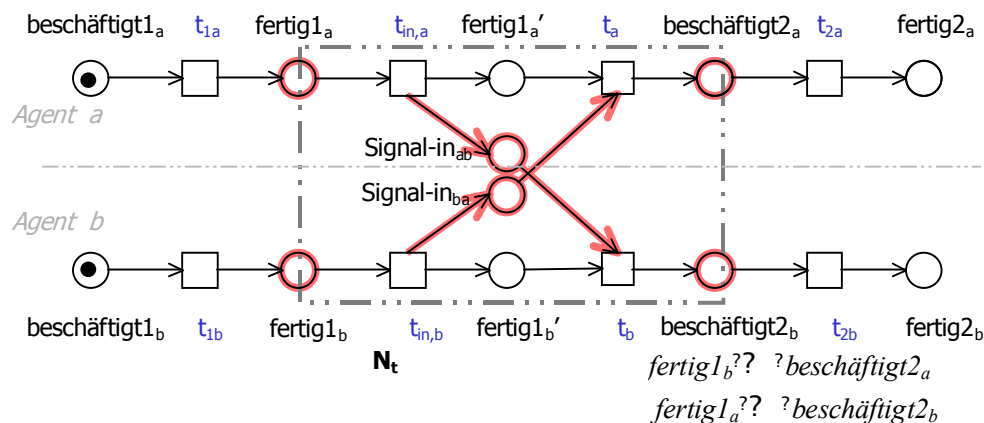


Abb. 5.2.18 Illustration für Synchronisationsbedingungen
(Verfeinerung Σ' von Σ in Abb. 5.2.1)

Im System Σ vor der Verteilung gibt es Synchronisationen zwischen unterschiedlichen Agenten wegen der gemeinsamen Transition der Agenten. Nach der Verteilung dieser Transition werden die Synchronisationen verschwinden, z.B. im Ersetzungsnetz NI_t gibt es gar keine Synchronisation zwischen den Agenten a und b mehr. Die Synchronisationen in einem verteilten Netz können nur durch Nachrichtenaustausch im Ersetzungsnetz wieder vorgenommen werden. Das Erhaltenbleiben der Synchronisationen zwischen unterschiedlichen Agenten lokal in einem Ersetzungsnetz sichert das Erhaltenbleiben der gewünschten Eigenschaften des ganzen Systems. Im Folgenden werden wir zunächst die Definition für Synchronisationsbedingungs-Formeln angeben, und dann definieren wir die Gültigkeit einer Synchronisationsbedingung im Ersetzungsnetz.

Angenommen, p und q sind Stellen aus dem Vor- bzw. Nachbereich einer unverteilter Transition t und gehören zu unterschiedlichen Agenten. Dann ist $\bullet p \blacktriangleleft \bullet q$ eine kausale Bedingung des Teilsystems (N_b, t^-) , wenn $p, q \in t^\bullet$.

Diese kausale Bedingung $\bullet p \blacktriangleleft \bullet q$ ist eine für t gültige Bedingung. Wir nennen sie eine Synchronisationsbedingung von t . Analog, die kausale Bedingung $p^\bullet \blacktriangleleft q^\bullet$ für $p, q \in t^-$ und die kausale Bedingung $p^\bullet \blacktriangleleft \bullet q$ für $p \in t^\bullet, q \in t^-$ vom Teilsystem (N_b, t^-) gelten auch für t und wir nennen sie ebenfalls Synchronisationsbedingungen von t .

Synchronisationsbedingungen von t sind also die Kausalitäten von Σ , deren Gültigkeit beim Verteilen von t möglicherweise erhalten bleiben sollen.

Definition 5.2.11 (*Synchronisationsbedingungs-Formeln*)

Sei Σ ein Agentensystem mit Agentenmenge A und Kanalmenge C , sei $t \in T$ unverteilt.

Die Menge \mathcal{P}_s der Synchronisationsbedingungs-Formeln von t wird wie folgt induktiv definiert:

a) $true \in \mathcal{P}_s$;

b) Seien $p, q \in (\bullet t \cup t \bullet) \setminus C$ mit $\neg (p \sim_A q)$.

$(\bullet p \blacktriangleleft \bullet q) \in \mathcal{P}_s$, wenn $p, q \in t \bullet$;

$(p \bullet \blacktriangleleft q \bullet) \in \mathcal{P}_s$, wenn $p, q \in \bullet t$;

$(p \bullet \blacktriangleleft \bullet q) \in \mathcal{P}_s$, wenn $p \in \bullet t, q \in t \bullet$.

c) $(p_{s1} \wedge p_{s2}) \in \mathcal{P}_s$, wenn $p_{s1}, p_{s2} \in \mathcal{P}_s$. □

Für das System Σ (Abb. 5.2.1) sind z.B. die folgenden Formeln Synchronisationsbedingungs-Formeln von t :

$$p_{s1} = (fertig1_b \bullet \blacktriangleleft \bullet besch\u00e4ftigt2_a) \quad (5-2-1)$$

$$p_{s2} = (fertig1_a \bullet \blacktriangleleft \bullet besch\u00e4ftigt2_b) \quad (5-2-2)$$

$$p_{s3} = sco-input = (fertig1_b \bullet \blacktriangleleft \bullet besch\u00e4ftigt2_a) \wedge (fertig1_a \bullet \blacktriangleleft \bullet besch\u00e4ftigt2_b) \quad (5-2-3)$$

$$p_{s4} \equiv true \quad (5-2-4)$$

sco-input (5-2-3) nennen wir auch *Input-Synchronisation*. Sie sichert, dass die Transitionen t_a und t_b erst schalten k\u00f6nnen, wenn die Bedingungen *fertig1_a* und *fertig1_b* beide erf\u00fcllt sind.

Synchronisationsbedingungen sind Kausalit\u00e4tsaussagen f\u00fcr (N_b, t^-) . Ihre Semantik wird daher wie folgt definiert.

Definition 5.2.12 (*G\u00fcltigkeit einer Synchronisationsbedingungs-Formel*)

Sei $\Sigma \rightarrow \Sigma'$ mit $\Sigma' = \Sigma(N_t \setminus t)$ eine verteilende Systemtransformation, sei $p_s \in \mathcal{P}_s$ eine Synchronisationsbedingung von t .

Die G\u00fcltigkeit von p_s im System (N_b, t^-) wird wie folgt induktiv definiert:

Fall a) $p_s \equiv true$:

Die Synchronisationsbedingung p_s ist in (N_b, t^-) erf\u00fcllt.

Fall b) p_s ist eine kausale Bedingung vom System (N_b, t^-)

(d.h. $p_s = p \blacktriangleleft q$ oder $p_s = p \blacktriangleleft q$ oder $p_s = p \blacktriangleleft q$):

Die Synchronisationsbedingung p_s ist in (N_t, t^-) erfüllt, gdw.

$$(N_t, t^-) \models p_s.$$

Fall c) $p_s = p_{s1} \wedge p_{s2}$, wobei $p_{s1}, p_{s2} \in \mathcal{P}_s$:

Die Synchronisationsbedingung p_s ist in (N_t, t^-) erfüllt, gdw.

p_{s1} und auch p_{s2} in (N_t, t^-) erfüllt sind. \square

Im Beispiel $\Sigma \rightarrow \Sigma'$ (siehe Abb. 5.2.18) sind im Ersetzungsnetz N_t alle Synchronisationsbedingungen p_{s1}, p_{s2}, p_{s3} und p_{s4} (Formeln (5-2-1) bis (5-2-4)) erfüllt. Im Gegensatz dazu ist bei dem anderen Verfeinerungsschritt $\Sigma \rightarrow \Sigma_1'$ (siehe Abb. 5.2.16) mit Ersetzungsnetz N_{t_1} keine dieser Synchronisationsbedingungen erfüllt.

3. Verteilende Verfeinerung bezüglich einer Synchronisationsbedingung

Angenommen, p_s ist eine Synchronisationsbedingungs-Formel einer unverteilter Transition t . Wenn das neue System Σ' nach der Verfeinerung die Blockbedingung erfüllt und die Synchronisationsbedingung p_s im Ersetzungsnetz N_t erfüllt ist, dann ist Σ' eine verteilende Verfeinerung von Σ bzgl. p_s .

Definition 5.2.13 (Verteilende Verfeinerung bzgl. p_s)

Sei $\Sigma \rightarrow \Sigma'$ mit $\Sigma' = \Sigma(N_t \setminus t)$ eine verteilende Systemtransformation, sei $p_s \in \mathcal{P}_s$.

Σ' heißt eine verteilende Verfeinerung bzgl. p_s von Σ , gdw. Σ' die Blockbedingung erfüllt und die Synchronisationsbedingung p_s in (N_t, t^-) erfüllt ist. \square

Im Beispiel $(\Sigma \rightarrow \Sigma')$ (siehe Abb. 5.2.18) erfüllt Σ' die Blockbedingung und die Synchronisationsbedingung *sco-input* ist im Ersetzungsnetz N_t erfüllt, deshalb ist dieser Verfeinerungsschritt eine verteilende Verfeinerung bzgl. *sco-input*.

Im Gegensatz dazu ist beim Verfeinerungsschritt $(\Sigma \rightarrow \Sigma_1')$ (siehe Abb. 5.2.16) die Synchronisationsbedingung *sco-input* im Ersetzungsnetz

NI_t nicht erfüllt, deshalb ist dieser Verfeinerungsschritt keine verteilende Verfeinerung bzgl. *sco-input*. Aber weil Σ_1' die Blockbedingung erfüllt, gilt deshalb: Der Verfeinerungsschritt ist eine verteilende Verfeinerung bzgl. $p_s = true$.

5.3 Kausalitäten im lokalen Teilsystem

In diesem Abschnitt werden wir zunächst eine Ableseregel für Kausalitäten zwischen Vor- bzw. Nachbereichen von Stellen angeben. Danach werden wir zeigen, welche Synchronisationsbedingungen durch $\bullet t \blacktriangleleft t \bullet$ gesichert werden.

Wenn das Netz eines Systems ein Kausalnetz ist, haben wir die folgende Ableseregel für die Gültigkeit einer Kausalität zwischen Vor- und Nachbereichen von Stellen im System.

Satz 5.3.1 (*Ableseregel der Relation \blacktriangleleft zwischen Vor- und Nachbereichen von Plätzen in Kausalnetzen*)

Sei Netz $N=(P,T,F)$ ein Kausalnetz, $\Sigma=(N, m_0)$ ein 1-beschränktes System mit: $m_0(p)=1$ gdw. $p \in {}^\circ N$. Weiter seien $p, q \in P$, sei $x = \bullet p$ oder $x = p \bullet$, sei $y = \bullet q$ oder $y = q \bullet$.

Seien $x = \{t\}$, $y = \{t'\}$, wobei $t, t' \in T$.

Dann gilt: $\Sigma \models x \blacktriangleleft y \leftrightarrow (t=t') \vee (t F^+ t')$.

Beweis:

Nach der Bemerkung 5.2.10 gilt: $\Sigma \models x \blacktriangleleft y \leftrightarrow (t=t') \vee \Sigma \models t \blacktriangleleft t'$.

Aus dem Satz 2.4.4 folgt: $\Sigma \models x \blacktriangleleft y \leftrightarrow (t=t') \vee (t F^+ t')$. \square

Die Kausalität $\bullet t \blacktriangleleft t \bullet$ im lokalen Teilsystem (N, t^-) sichert jede Synchronisationsbedingung $p \bullet \blacktriangleleft \bullet q$ mit $p \in \bullet t$ und $q \in t \bullet$. Diese Kausalität kann aber eine Synchronisationsbedingung $p \bullet \blacktriangleleft q \bullet$ mit $p, q \in \bullet t$ oder eine Synchronisationsbedingung $\bullet p \blacktriangleleft \bullet q$ mit $p, q \in t \bullet$ nicht sichern. D.h. wir können mit Hilfe von Synchronisationsbedingungen schärfere Kausalitäten an (N, t^-) formulieren als $\bullet t \blacktriangleleft t \bullet$.

Bemerkung 5.3.2

Sei $\Sigma \rightarrow \Sigma'$ mit $\Sigma' = \Sigma(N_t \setminus t)$ eine verteilende Systemtransformation.

Sei $p_s = (p \bullet \blacktriangleleft \bullet q) \in \mathcal{P}_s$ mit $p \in \bullet t$ und $q \in t \bullet$ eine Synchronisationsbedingung von t .

Dann gilt:

$$(N_b, t^-) \models \bullet t \blacktriangleleft t \bullet \quad \rightarrow \quad \text{in } (N_b, t^-) \text{ ist } p_s \text{ erfüllt.}$$

Beweis:

Sei (K_{N_p}, r_{N_t}) ein Ablauf von (N_b, t^-) , wobei $K_{N_t} = (B_{N_t}, E_{N_t}, \leq_{N_t})$.

Seien $r_{N_t}(b) = p$, $r_{N_t}(b') = q$ mit $b \in {}^\circ K_{N_t}$, $b' \in K_{N_t}^\circ$.

Seien $e, e' \in E_{N_t}$ mit $b \leq_{N_t} e$, $e' \leq_{N_t} b'$.

$$\text{Zu zeigen ist: } e \leq_{N_t}^+ e' \quad (1)$$

Wegen $(N_b, t^-) \models \bullet t \blacktriangleleft t \bullet$ und $p \in \bullet t$, $q \in t \bullet$ gilt:

$$b \leq_{N_t}^+ b' \quad (2)$$

Angenommen, (1) gilt nicht.

Dann gilt: entweder $e \underline{\text{co}} e'$ oder $e' \leq_{N_t}^+ e$.

Daraus folgt $b \underline{\text{co}} b'$.

Das ist Widerspruch zu (2).

Daraus folgt: (1) gilt. □

5.4 Erhaltenbleiben von Eigenschaften durch Synchronisationsbedingungen

In diesem Abschnitt werden wir darüber diskutieren, welche temporal-logischen Eigenschaften durch Synchronisationsbedingungen erhaltenbleiben.

5.4.1 Erhaltenbleiben von Sicherheitseigenschaften

Für den Verfeinerungsschritt $\Sigma \rightarrow \Sigma''$ im Beispiel im Abschnitt 3.4 (siehe Abb. 3.4.1 (a), Abb. 3.4.2) gilt: In (N_b, t^-) ist die Synchronisationsbedingung $p_s = (p_a \bullet \blacktriangleleft \bullet q_b)$ erfüllt. Deshalb bleibt die Sicherheitseigenschaft $\square \neg(p_a \wedge q_b)$ bei $\Sigma \rightarrow \Sigma''$ erhalten.

In Σ gilt: $p_a' + p_a + q_b \leq 1$ (d.h. in jedem erreichbaren Zustand gilt: Von den Stellen p_a' , p_a und q_b ist höchstens eine Stelle markiert). Deshalb bleibt auch die Sicherheitseigenschaft $\square \neg(p_a' \wedge q_b)$ erhalten.

Im Folgenden werden wir zwei Lemmata über Erhaltenbleiben von Eigenschaften angeben. Vorher geben wir zwei Bemerkungen an, die wir zum Beweis benutzen werden.

Wenn in jedem erreichbaren Zustand von Σ gilt, dass die Stellen p und q nie gleichzeitig markiert sind, dann gibt es für jeden Ablauf (K, r) von Σ eine Folge $\sigma = b_0 b_1 \dots$ mit:

Jedes Element $b_k, k=0,1,\dots$ ist ein Auftreten von p oder q in K (also $b_k \in B$ und $r(b_k) \in \{p, q\}$) und $b_k \leq^+ b_{k+1}$.

Alle Auftreten von p und q in K kommen in σ vor.

Bemerkung 5.4.1

Es gelte: $\Sigma \models \Box \neg(p \wedge q)$.

Sei (K, r) ein Ablauf von Σ , wobei $K = (B, E, \leq)$.

Dann gilt:

Es gibt eine Folge $\sigma = b_0 b_1 \dots$ mit:

- Für $k=0,1,\dots$ gilt: $b_k \in B$ und $r(b_k) \in \{p,q\}$ und $b_k \leq^+ b_{k+1}$.

- Für jedes $b \in B$ mit $r(b) \in \{p,q\}$ gilt:

Es gibt ein $k \in \{0,1,\dots\}$, so dass $b_k = b$.

Beweis:

$B_{pq} := \{ b \in B \mid r(b) \in p \text{ oder } r(b) \in q \}$.

Seien $b, b' \in B_{pq}$ mit $b \neq b'$.

Fall 1. $r(b) = r(b')$:

Es gilt $\neg(b \text{ co } b')$. (wegen 1-Beschränktheit)

Fall 2. $r(b) \neq r(b')$:

Es gilt auch $\neg(b \text{ co } b')$. (wegen $\Box \neg(p \wedge q)$)

Also es gilt: $\neg(b \text{ co } b')$.

Dann gilt: $b \leq^+ b'$ oder $b' \leq^+ b$.

Daraus folgt, es gibt eine Sequenz $\sigma = b_0 b_1 \dots$ mit:

- $\{b_0, b_1, \dots\} = B_{pq}$;

- Für $k=0,1,\dots$ gilt $b_k \leq^+ b_{k+1}$.

Daraus folgt die Behauptung. \square

Bemerkung 5.4.2 ist eine Erweiterung von Bemerkung 5.4.1. Wenn in jedem erreichbaren Zustand von Σ gilt, dass aus einer Stellenmenge $\{p_1, p_2, \dots, p_n\}$ höchstens eine Stelle markiert ist, dann gibt es für jeden Ablauf (K, r) von Σ eine Folge $\sigma = b_0 b_1 \dots$ mit:

Jedes Element $b_k, k=0,1,\dots$ ist ein Auftreten von einer Stelle aus $\{p_1, p_2, \dots, p_n\}$ in K (also $b_k \in B$ und $r(b_k) \in \{p_1, p_2, \dots, p_n\}$) und $b_k \leq^+ b_{k+1}$.

Alle Auftreten von Stellen aus $\{p_1, p_2, \dots, p_n\}$ in K kommen in σ vor.

Bemerkung 5.4.2

Es gelte: $\Sigma \models p_1 + p_2 + \dots + p_n \leq 1$.

Sei (K, r) ein Ablauf von Σ , wobei $K = (B, E, \leq)$.

Dann gilt:

Es gibt eine Folge $\sigma = b_0 b_1 \dots$ mit:

- Für $k=0,1,\dots$ gilt: $b_k \in B$ und $r(b_k) \in \{p_1, p_2, \dots, p_n\}$ und $b_k \leq^+ b_{k+1}$.

- Für jedes $b \in B$ mit $r(b) \in \{p_1, p_2, \dots, p_n\}$ gilt:
 Es gibt ein $k \in \{0, 1, \dots\}$, so dass $b_k = b$.

Lemma 5.4.3

$\Sigma' = \Sigma(N_t \setminus t)$ erfülle die Blockbedingung.

Sei $p_s = (p \bullet \blacktriangleleft \bullet q)$ mit $p \in \bullet t$ und $q \in t \bullet$ eine Synchronisationsbedingung von t .

Ist p_s in (N_t, t^-) erfüllt, so bleibt bei $(\Sigma \rightarrow \Sigma')$ die Sicherheitseigenschaft $\square \neg (p \wedge q)$ erhalten.

Beweis:

Seien $(K, r), (K', r')$ Abläufe von Σ, Σ' mit: (K', r') ist eine $t \rightarrow N_t$

Substitution von (K, r) , wobei $K = (B, E, \leq), K' = (B', E', \leq')$.

Angenommen, in K gilt: $\square \neg (p \wedge q)$ (1)

Zu zeigen ist: In K' gilt: $\square \neg (p \wedge q)$ (2)

Aus (1) folgt: (wegen Bem. 5.4.1)

Es gibt eine Folge $\sigma = b_0 b_1 \dots$ mit:

- Jedes Element $b_k, k=0, 1, \dots$ ist ein Auftreten von p oder q in K

(also $b_k \in B$ und $r(b_k) \in \{p, q\}$) und $b_k \leq^+ b_{k+1}$.

Alle Auftreten von p und q in K kommen in σ vor.

Dann gilt $b_k \in B', k=0, 1, \dots$ (wegen Bem. 3.2.5)

und es gibt in K' sonst keine weitere Auftreten von p oder q .

Fall 1. $b_k \leq e \leq b_{k+1}$ mit $r(e) = t$.

Dann gilt: $r(b_k) = p, r(b_{k+1}) = q$.

Wegen $(N_t, t^-) \models p \bullet \blacktriangleleft \bullet q$ gilt: $b_k \leq'^+ b_{k+1}$.

Fall 2. $b_k \leq'^+ b_{k+1}$ mit: Für jedes $e \in E$ mit $b_k \leq^+ e \leq'^+ b_{k+1}$ gilt $r(e) \neq t$.

Es gilt: $b_k \leq'^+ b_{k+1}$. (wegen Bem. 3.2.5)

Also, $\sigma = b_0 b_1 \dots$ ist ein Weg mit:

- Jedes Element $b_k, k=0, 1, \dots$ ist ein Auftreten von p oder q in K'

(also $b_k \in B'$ und $r'(b_k) \in \{p, q\}$) und $b_k \leq'^+ b_{k+1}$.

Alle Auftreten von p und q in K' kommen in σ vor.

Daraus folgt:

In K' ist ein solcher Zustand nicht erreichbar, in dem p und q markiert sind.

Also, (2) gilt. □

Lemma 5.4.4

$\Sigma' = \Sigma(N_t \setminus t)$ erfülle die Blockbedingung.

Sei $p_s = (p \bullet \blacktriangleleft \bullet q)$ mit $p \in \bullet t$ und $q \in t \bullet$ eine Synchronisationsbedingung von t .

Es gelte $\Sigma \models p+q+r \leq 1$, wobei $r \in P \setminus (\bullet t \cup t \bullet)$. (also keine Stelle des Vor- oder Nachbereichs von t)

Dann gilt:

Wenn p_s in (N_t, t^-) erfüllt ist, dann bleiben bei $(\Sigma \rightarrow \Sigma')$ die Sicherheitseigenschaften $\Box \neg(p \wedge r)$ und $\Box \neg(q \wedge r)$ erhalten.

Beweis:

Beweisidee ist ähnlich wie bei Lemma 5.4.3. Wir beweisen hier nur $\Box \neg(p \wedge r)$.

Seien $(K, r), (K', r')$ Abläufe von Σ, Σ' mit: (K', r') ist eine $t \rightarrow N_t$

Substitution von (K, r) , wobei $K = (B, E, \leq), K' = (B', E', \leq')$.

Angenommen, in K gilt: $\Box \neg(p \wedge r)$ (1)

Zu zeigen ist: In K' gilt: $\Box \neg(p \wedge r)$ (2)

Aus $\Sigma \models p+q+r \leq 1$ folgt: (wegen Bem. 5.4.2)

Es gibt eine Folge $\sigma = b_0 b_1 \dots$ mit:

- Jedes Element $b_k, k=0,1,\dots$ ist ein Auftreten von Stellen aus $\{p,q,r\}$ in K

(also $b_k \in B$ und $r(b_k) \in \{p,q,r\}$) und $b_k \leq^+ b_{k+1}$.

Alle Auftreten von Stellen aus $\{p,q,r\}$ in K kommen in σ vor.

Dann gilt $b_k \in B', k=0,1,\dots$ (wegen Bem. 3.2.5)

und es gibt in K' sonst keine weiteren Auftreten von Stellen aus $\{p,q,r\}$.

Fall 1. $b_k \leq e \leq b_{k+1}$ mit $r(e) = t$.

Dann gilt: $r(b_k) = p, r(b_{k+1}) = q$.

Wegen $(N_t, t^-) \models p \bullet \blacktriangleleft \bullet q$ gilt: $b_k \leq'^+ b_{k+1}$.

Fall 2. $b_k \leq^+ b_{k+1}$ mit: Für jedes $e \in E$ mit $b_k \leq^+ e \leq^+ b_{k+1}$ gilt $r(e) \neq t$.

Es gilt: $b_k \leq'^+ b_{k+1}$. (wegen Bem. 3.2.5)

Also, $\sigma = b_0 b_1 \dots$ ist ein Weg mit:

- Jedes Element $b_k, k=0,1,\dots$ ist ein Auftreten von Stellen aus $\{p,q,r\}$ in

K'

(also $b_k \in B'$ und $r'(b_k) \in \{p, q, r\}$) und $b_k \leq'^+ b_{k+1}$.

Alle Auftreten von Stellen aus $\{p, q, r\}$ in K' kommen in σ vor.

Daraus folgt:

In K' ist ein solcher Zustand nicht erreichbar, in dem p und r markiert sind.

Also gilt (2). □

Das folgende Lemma über das Erhaltenbleiben von Sicherheitseigenschaften werden wir im Kapitel 7 (vgl. Abschnitt 7.1.3) verwenden.

Lemma 5.4.5

$\Sigma' = \Sigma(N_t \setminus t)$ erfülle die Blockbedingung. Sei N_t ein Kausalnetz.

Sei $p_s = (\bullet p \blacktriangleleft \bullet q)$ mit $p, q \in t^\bullet$ eine Synchronisationsbedingung von t .

Ist p_s in (N_t, t^-) erfüllt, so bleibt bei $(\Sigma \rightarrow \Sigma')$ die Sicherheitseigenschaft

$\square(q \rightarrow p)$ erhalten.

Beweis:

Seien $(K, r), (K', r')$ Abläufe von Σ, Σ' mit: (K', r') ist eine $t \rightarrow N_t$

Substitution von (K, r) , wobei $K = (B, E, \leq), K' = (B', E', \leq')$.

Sei K_{N_t} ein Auftreten von N_t in K' , wobei $K_{N_t} = (B_{N_t}, E_{N_t}, \leq_{N_t})$.

Seien $b_p, b_q \in K_{N_t}^\circ$ mit $r'(b_p) = p$ und $r'(b_q) = q$.

Sei C ein Schnitt in K' mit $b_q \in C$.

Zu zeigen ist: $b_p \in C$. (1)

Sei $b_p' \in B'$ mit $b_p' \leq' e_{t_1} \leq' b_p$ für ein $e_{t_1} \in E'$. Sei $e_{t_2} \in E'$ mit $e_{t_2} \leq' b_q$.

Dann gilt: $b_p' \in B_{N_t}, e_{t_1}, e_{t_2} \in E_{N_t}$.

Weil N_t ein Kausalnetz ist, gilt $r'(b_p') \neq p$.

Wegen $(N_t, t^-) \models \bullet p \blacktriangleleft \bullet q$ gilt: $e_{t_1} \leq'^+ e_{t_2}$.

Dann gilt: $b_p' \leq' e_{t_1} \leq'^+ e_{t_2} \leq' b_q$.

Also $b_p' \leq'^+ b_q$.

Daraus folgt, für jedes $b \in B'$ mit $b \leq'^+ b_p$ gilt $b \leq'^+ b_q$.

Daraus folgt: $b_p \in C$. (2)

Sei $b_{p''} \in B'$ mit $b_p \leq' e \leq' b_{p''}$ und $r'(b_{p''}) = p'' \neq p$ und $p'' \in P_{N_t} \setminus (\bullet t \cup t \bullet)$.

Wir zeigen: $b_q \leq'^+ b_{p''}$.

Wir nehmen an, $\neg (b_q \leq'^+ b_{p''})$ gilt.

Dann gilt $b_q \underline{\text{co}} b_{p''}$ oder $b_{p''} \leq'^+ b_q$.

Fall 1. $b_q \underline{\text{co}} b_{p''}$.

Dann gilt: $b_{p''} \in B, b_q \in B$ und $b_q \underline{\text{co}} b_{p''}$ in K . (wegen Bem. 3.2.5)

D.h. in K gibt es einen erreichbaren Zustand, in dem q gilt aber p nicht gilt. Das ist ein Widerspruch zu $\Sigma \models \Box(q \rightarrow p)$.

Daraus folgt, $\neg (b_q \underline{\text{co}} b_{p''})$ gilt. (3)

Fall 2. $b_{p''} \leq'^+ b_q$.

Dann gilt: $b_p \leq'^+ b_{p''} \leq'^+ b_q$.

D.h. $b_p \leq'^+ b_q$. Das ist ein Widerspruch zu $b_p \underline{\text{co}} b_q$.

Daraus folgt, $\neg (b_{p''} \leq'^+ b_q)$ gilt. (4)

Aus (3),(4) folgt, $b_q \leq'^+ b_{p''}$ gilt.

Daraus folgt, für jedes $b \in B'$ mit $b_p \leq' b$ gilt $b_q \leq'^+ b$.

Daraus folgt: $C \text{ [co } b_p$. (5)

Aus (2), (5) folgt: $b_p \in C$.

Daraus folgt die Behauptung. □

5.4.2 Erhaltenbleiben von Lebendigkeitseigenschaften

Für den Verfeinerungsschritt $\Sigma \rightarrow \Sigma''$ im Beispiel im Abschnitt 3.4 (siehe Abb. 3.4.1 (a), Abb. 3.4.2) gilt: Die Stellen $p_{a'}$, p_a gehören zum selben Agenten. Und in Σ gilt: $p_{a'} + p_a \leq 1$. Dann bleibt die Lebendigkeitseigenschaft $p_{a'} \triangleright p_a$ bei $\Sigma \rightarrow \Sigma''$ erhalten.

Im Folgenden werden wir ein Lemma über das Erhaltenbleiben von Lebendigkeitseigenschaften angeben. Vorher geben wir eine Bemerkungen an, die wir zum Beweis benutzen werden. Die Idee bei der Bemerkung ist ähnlich wie die bei Satz 3.5.1.

Bemerkung 5.4.6

Sei Σ ein Agentensystem mit Agentenmenge A .

Erfülle $\Sigma' = \Sigma(N_t \setminus t)$ die Blockbedingung.

Seien $(K, r), (K', r')$ jeweils Abläufe von Σ, Σ' mit: (K', r') ist eine $t \rightarrow N_t$ Substitution von (K, r) , wobei $K = (B, E, \triangleleft), K' = (B', E', \triangleleft')$.

Wenn $(N_t, t^-) \models \bigwedge_{\substack{x \in t, y \in t \\ x \sim_A y}} (x \triangleleft y)$, dann gilt:

Für beliebige $s, s' \in B$ mit $r(s) \sim_A r(s')$ gilt: $s \triangleleft^+ s'$ in $K \rightarrow s \triangleleft'^+ s'$ in K' .

Beweis: ähnlich wie bei Satz 3.5.1, nur dass hier ausschließlich Stellen eines Agenten betrachtet werden.

Lemma 5.4.7

Sei Σ ein Agentensystem mit Agentenmenge A , seien $p, q \in P$ mit $p \sim_A q$.

Erfülle $\Sigma' = \Sigma(N_t \setminus t)$ die Blockbedingung.

Es gelte $\Sigma \models p+q \leq 1$.

Wenn $(N_t, t^-) \models \bigwedge_{\substack{x \in t, y \in t \\ x \sim_A y}} (x \triangleleft y)$, dann bleibt bei $(\Sigma \rightarrow \Sigma')$ die

Lebendigkeitseigenschaft $p \triangleright q$ erhalten.

Beweis:

Angenommen, in Σ gilt: $p \triangleright q$. Zu zeigen ist: In Σ' gilt: $p \triangleright q$.

Seien $(K, r), (K', r')$ Abläufe von Σ, Σ' mit: (K', r') ist eine $t \rightarrow N_t$

Substitution von (K, r) , wobei $K = (B, E, \triangleleft), K' = (B', E', \triangleleft')$.

Sei C ein Zustand von K' , in dem p gilt.

Zu zeigen ist: Es gibt einen von C erreichbaren Zustand C' , in dem q gilt.

Sei $b_p \in C$ mit $r(b_p) = p$.

Dann gilt $b_p \in B$ (wegen Bem. 3.2.5)

Wegen $\Sigma \models p \triangleright q$ und $\Sigma \models p+q \leq 1$ gilt:

Es gibt ein $b_q \in B$ mit $r(b_q) = q$ und $b_p \leq^+ b_q$ in Σ .

Dann gilt $b_q \in B'$ und $r'(b_q) = q$ (wegen Bem. 3.2.5)

Wegen $r(b_p) \sim_A r(b_q)$ und $(N_b, t^-) \models \bigwedge_{\substack{x \in t^-, y \in t^- \\ x \sim_A y}} (x \blacktriangleleft y)$ gilt:

$$b_p \leq'^+ b_q.$$

Also, es gibt einen von C erreichbaren Zustand C' , in dem q gilt. \square

6. Nachweis der Blockbedingung bei der verteilenden Verfeinerung - Grundtypen von Verfeinerungsnetzen

6.1 Einführung

Die Blockbedingung ist eine Voraussetzung für eine korrekte Transitionsverfeinerung. Beim Entwurf fragt man deshalb, wie man ein Verfeinerungsnetz konstruiert, um die Blockbedingung zu garantieren. Und bei der Verifikation fragt man, wann die Blockbedingung erfüllt ist. Dafür brauchen wir Kriterien für die Blockbedingung.

In der Literatur gibt es auch einige Kriterien für die Blockbedingung (vgl. [Peu01]), wir brauchen aber einfache und praktische Kriterien.

Wir betrachten hier die Verfeinerung einer unverteilter Transition von zwei Agenten. In unserem Modell Agentensysteme können wir einen Agenten weiter in zwei Agenten zerlegen. Deshalb brauchen wir im Wesentlichen nur die Verfeinerung von zwei Agenten zu betrachten.

Ob die Blockbedingung bei einem Verfeinerungsschritt erfüllt ist, das hängt von der Umgebung der zu verfeinernden Aktion ab. Wir geben die Kriterien für unterschiedliche Umgebungen an, die besagen, unter welchen Bedingungen ein Ersetzungsnetz die Blockbedingung garantiert.

Wir haben drei Grundtypen von Ersetzungsnetzen herausgearbeitet, die in Abhängigkeit von den vorangegangenen Absprachen der Agenten zu verwenden sind. Bei dem ersten Grundtyp (*abgesprochene gemeinsame Aktion* genannt) haben sich die Agenten vorher gegenseitig informiert. Bei dem zweiten Grundtyp (*erbetene Zusammenarbeit*), hat ein Agent dem anderen Agenten eine Nachricht geschickt und bei dem dritten Grundtyp (*erwartete Zusammenarbeit*) ist ohne vorbereitende Kommunikation durch den Algorithmus gesichert, dass die Aktion gemeinsam ausgeführt wird.

Bei der Verfeinerung einer Transition t muss beachtet werden, dass alle Stellen aus dem Nachbereich von t kausal nach den Stellen aus dem Vorbereich von t belegt werden. Die drei Typen von Ersetzungsnetzen schwächen diese Eigenschaft unterschiedlich stark ab. Die abgesprochene

gemeinsame Aktion benötigt keine Kommunikation im Ersetzungsnetz und schwächt damit die Kausalität zwischen der Belegung des Vorbereichs und Nachbereichs von t am stärksten ab. Die erwartete Zusammenarbeit erfordert im Ersetzungsnetz zwei Nachrichten und erhält vollständig die Kausalitätseigenschaft.

In den Abschnitten 6.2-6.4 werden diese drei Grundtypen diskutiert.

Eine interessante Frage bei der Verfeinerung ist die Vertauschbarkeit von Verfeinerungsschritten. Angenommen, der Anfangsalgorithmus Σ_0 hat n zu verfeinernden Transitionen t_1, t_2, \dots, t_n . Dann ist der Verfeinerungsprozess $\Sigma_0 \rightarrow \Sigma_1 \rightarrow \dots \rightarrow \Sigma_n$, wobei $\Sigma_k = \Sigma_{k-1}(N_{t_k} \setminus t_k)$, $k=1, 2, \dots, n$. Wann ist die Reihenfolge von Verfeinerungsschritten beliebig? Für jeden Verfeinerungsschritt $\Sigma_k = \Sigma_{k-1}(N_{t_k} \setminus t_k)$, $k=1, 2, \dots, n$, müssen wir für die Blockbedingung die Umgebung von t_k in Σ_{k-1} analysieren. Im Verfeinerungsprozess wird Σ_k immer komplizierter und die Analyse für Σ_k wird immer schwerer. Es wäre viel einfacher, wenn wir für jede Transition t_k nur den Verfeinerungsschritt ($\Sigma_0 \rightarrow \Sigma_{k'}$) mit $\Sigma_{k'} = \Sigma_0(N_{t_k} \setminus t_k)$ zu betrachten brauchen. Unter welcher Bedingung geht das? Diese Frage werden wir im Abschnitt 6.5 diskutieren.

Können wir immer durch Kommunikation innerhalb des Ersetzungsnetzes die Blockbedingung garantieren? Die Antwort werden wir im Abschnitt 6.6 geben.

6.2 Abgesprochene gemeinsame Aktion

In diesem Abschnitt betrachten wir den einfachsten Fall einer Umgebung einer gemeinsamen Aktion zweier Agenten a und b . Bevor die gemeinsame Aktion gestartet wird, haben beide Agenten mitgeteilt, dass sie bereit sind, ihre Teilaufgabe auszuführen und das auch sicher tun werden.

Im Petrinetzmodell bedeutet diese Voraussetzung: Wenn alle Stellen aus $\bullet t$ für a belegt sind, dann sind auch alle Stellen für b belegt, und es gibt keine konkurrente Transition zur Aktion t .

Wir sehen uns ein Beispiel an, das diese Voraussetzungen erfüllt. Angenommen a und b möchten zusammen einen *Rechner transportieren*. Sie machen das wie folgt: Jeder wird dem anderen ein Zeichen, z.B. durch Augenkontakt, geben, wenn er *bereit* ist. Wenn die beiden *bereit* sind, dann können sie zusammen *den Rechner transportieren* (Abb. 6.2.1). Nichts anderes hält sie davon ab.

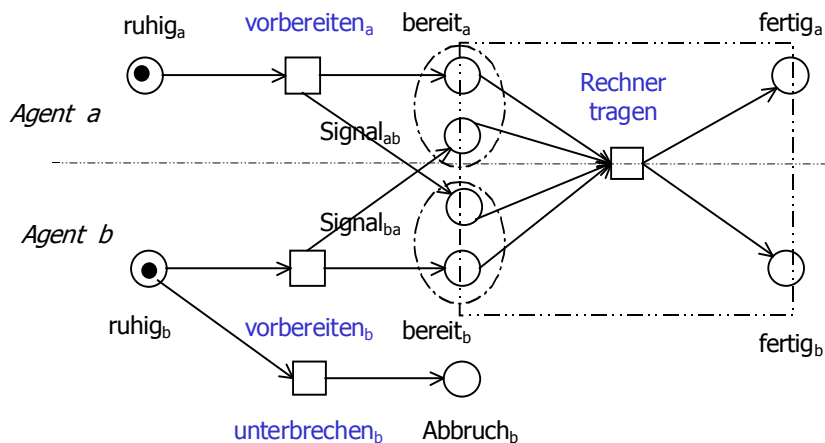


Abb. 6.2.1 System *Rechner transportieren*

In Abb. 6.2.2 wird die Aufgabe *Rechner transportieren* in zwei Teilaufgaben - *das Rechnergehäuse transportieren* und *den Monitor transportieren* geteilt, die jeweils einer der beiden Personen übernimmt. Die beiden machen es auch noch so wie im System *Rechner transportieren*. Durch Augenkontakt weiß jeder Agent, ob der andere Agent schon *bereit* ist. Wenn er weiß, dass sie beide bereit sind, dann kann er seine Teilaufgabe ausführen – also *Rechnergehäuse* bzw. *Monitor tragen*.

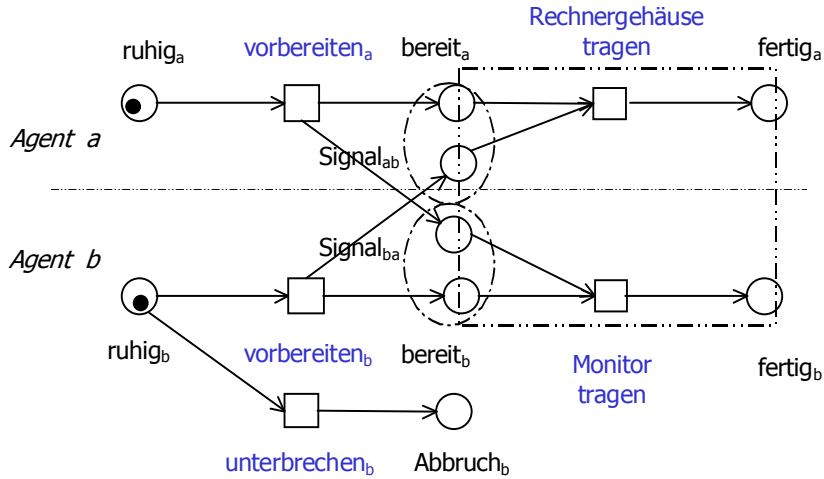


Abb. 6.2.2 Verteiltes System *Rechner transportieren*

Offenbar passiert im verteilten System *Rechner transportieren* nach der Verteilung der Aufgabe das folgende nicht: *a* transportiert das Gehäuse, *b* aber den Monitor nicht, weil *a* vorher schon durch Augenkontakt das Zeichen für das *Bereitsein* von *b* bekommt, d.h. hier gilt die Blockbedingung.

Die gemeinsame Aufgabe der beiden Agenten *a* und *b* wird oft viel komplizierter sein als in diesem Beispiel, d.h. das Verfeinerungsnetz N_t ist nicht immer so einfach wie hier. Trotzdem wird ein Nachweis der Blockbedingung nicht wesentlich schwieriger, als in diesem Beispiel. Die folgende Bemerkung zeigt, dass immer nur die Transitionen von N_t zu betrachten sind, die Stellen aus $\bullet t$ im Vorbereich haben.

Bemerkung 6.1 Sei $\Sigma \rightarrow \Sigma'$ mit $\Sigma' = \Sigma(N_t \setminus t)$ eine verteilende Systemtransformation. K' sei ein Ablauf von Σ' . C sei Schnitt in K' . In C beginnt ein Ablauf von (N_b, t^-) , wenn in C ein Anfangsstück K_A eines Ablaufs von (N_b, t^-) mit ${}^\circ K_A \cap K_A^\circ = \emptyset$ beginnt, d.h. jede Marke aus ${}^\circ K_A$ wurde bereits im Anfangsstück von K_A konsumiert.

Beweis: Für (N_b, t^-) ist das nach Voraussetzung erfüllt. In Σ' gilt das weiter, da keine Transition von $\Sigma \setminus \{t\}$ konkurrent zu einer Transition von N_t sein kann, wegen $P \cap P_{t\text{-intern}} = \emptyset$, wobei $P_{t\text{-intern}} = P_{N_t} \setminus (\bullet t \cup t^\bullet)$. Das Anfangsstück K_A muss also wie in (N_b, t^-) zu einem Ablauf von (N_b, t^-) vervollständigt werden. \square

Von Bemerkung 6.2.1. können wir eine Folgerung erhalten. Diese ist ähnlich wie die Bem. 6.2.1. Der Unterschied ist wie folgt: Hier beginnt in einem Schnitt C noch nicht ein Anfangsstück K_A eines Ablaufs von (N_t, t^-) , sondern nur ein Auftreten e einer Transition im Ersetzungsnetz N_t . Und von C aus ist ein Schnitt C_{nach} erreichbar, der alle Bedingungen im Vorbereich $\bullet e$ von e noch enthält, und in dem ein Anfangsstück K_A eines Ablaufs von (N_t, t^-) beginnt. Dann beginnt in C_{nach} ein vollständiger Ablauf von (N_t, t^-) , der dann auch dieses Ereignis e enthält.

Folgerung 6.2.2 (von Bem. 6.2.1)

Sei $\Sigma \rightarrow \Sigma'$ mit $\Sigma' = \Sigma(N_t \setminus t)$ eine verteilende Systemtransformation. (K', r') sei ein Ablauf von Σ' . C sei Schnitt in K' , in dem ein $t' \in N_t$ mit $\bullet t' \cap \bullet t \neq \emptyset$ schaltet. Sei $C' \subseteq C$ mit $r'(C') = \bullet t'$. Wenn auf C ein Schnitt C_{nach} folgt, der C' enthält und in dem ein Anfangsstück K_A eines Ablaufs von (N_t, t^-) mit ${}^\circ K_A \cap K_A^\circ = \emptyset$ beginnt, dann beginnt in C_{nach} ein Ablauf von (N_t, t^-) .

Beweis: Nach der Voraussetzung gilt: In C_{nach} beginnt ein Anfangsstück K_A eines Ablaufs von (N_t, t^-) mit ${}^\circ K_A \cap K_A^\circ = \emptyset$. Nach der Bem 6.1 folgt: In C_{nach} beginnt ein Ablauf von (N_t, t^-) . □

Jetzt ist noch zu klären, wie die Nachrichtenstellen für einen Agenten zu definieren sind. In unserem Beispiel haben wir Kanalstellen, die Nachrichten für einen Agenten aufnehmen, z.B. $Signal_{ab}$ nimmt eine Nachricht von a für b auf. Im unverteilten System *Rechner transportieren* (siehe Abb. 6.2.1) ist die Kanalstelle $Signal_{ab}$ von der Stelle $bereit_a$ des Agenten a nicht syntaktisch zu unterscheiden. Es ist unsere Entwurfsentscheidung, dass $Signal_{ab}$ bei der Verfeinerung ein Signal von a für b aufnehmen soll.

Wir werden Kanalstellen aus C , die bei der Verfeinerung Nachrichten für den Agenten a aufnehmen sollen, mit C_a bezeichnen. Bei der Festlegung der Menge C_a müssen wir beachten, dass die Nachricht von einer Transition gesendet wird, die nicht lokal zu a ist und von einer Transition empfangen wird, die a betrifft.

Definition 6.2.3 (Nachrichten für einen Agenten a)

Eine Menge $C_a \subseteq C$ der Kanalstellen mit Nachrichten für einen Agenten a ist Teilmenge von C mit:

Für jedes $p \in C_a$ gilt:

- i. Es existiert eine Transition t_e mit $p \in \bullet t_e$ und $t_e \bullet \cap a \neq \emptyset$
(t_e empfängt die Nachricht p und betrifft den Agenten a oder ist lokal zu a)
- ii. und es existiert eine Transition t_s mit $p \in t_s \bullet$ und t_s nicht lokal zu a
(t_s sendet die Nachricht p und betrifft mindestens einen Agenten ungleich a).

Wir können jetzt den ersten Satz formulieren, der sagt, welche Eigenschaften eine Umgebung einer zu verteilenden Transition besitzen muß, damit eine bestimmte Klasse von Verfeinerungsnetzen die Blockbedingung garantiert.

Wir notieren die Menge aller Stellen aus $\bullet t$ für einen Agenten a mit $(\bullet t)_a$, d.h. $(\bullet t)_a = \bullet t \cap (a \cup C_a)$. Wir betrachten hier eine solche unverteilte Transition t , die nur zwei Agenten, z.B. Agenten a und b , betrifft, also gilt $\bullet t = (\bullet t)_a \cup (\bullet t)_b$.

Satz 6.2.4

Gegeben sei das Agentensystem Σ mit Agentenmenge A und Kanalmenge C und die unverteilte Transition t . Sei $(\bullet t)_a := \bullet t \cap (a \cup C_a)$, $(\bullet t)_b := \bullet t \cap (b \cup C_b)$ und $\bullet t = (\bullet t)_a \cup (\bullet t)_b$, wobei $a, b \in A$.

In Σ gelte:

- i. $(\bullet t)_a \leftrightarrow (\bullet t)_b$;
- ii. Es gibt keine konkurrente Transition zu t .

Dann garantiert jedes Verfeinerungsnetz N_t mit folgenden Eigenschaften die Blockbedingung für $\Sigma' = \Sigma(N_t \setminus t)$:

In N_t gibt es Transitionen t_{Start_a} und t_{Start_b} mit:

- iii. $\bullet t_{Start_a} = (\bullet t)_a$, $\bullet t_{Start_b} = (\bullet t)_b$ und
- iv. es gibt keine konkurrente Transition zu t_{Start_a} und t_{Start_b} in N_t .

Beweis:

Zu zeigen ist: Jeder Ablauf von Σ' wird aus einem Ablauf von Σ erhalten, indem jedes Auftreten von t durch einen vollständigen Ablauf

von (N_b, t^-) ersetzt wird.

Wir nehmen an, es gibt einen Ablauf (K', r') von Σ' , für den das nicht der Fall ist. Diese Annahme ist zum Widerspruch zu führen.

Aus der Annahme folgt: (K', r') enthält einen unvollständigen Ablauf von (N_t, t^-) . Es gibt einen Schnitt C , in dem dieser unvollständige Ablauf beginnt, also t_{Start_a} oder t_{Start_b} schaltet. Aus Symmetriegründen können wir annehmen, dass in C t_{Start_a} schaltet, also $C' \subseteq C$ mit $r'(C') = \bullet t_{Start_a}$. Nach Voraussetzung gilt in C $(\bullet t)_a \rightarrow (\bullet t)_b = \bullet t_{Start_b}$. Da es nach Voraussetzung keine konkurrenten Transitionen zu t und t_{Start_b} gibt, schaltet auch t_{Start_b} im Schnitt C . Nach Bem. 6.2.1 ist das ein Widerspruch dazu, dass in C kein vollständiger Ablauf von (N_b, t^-) beginnt.

Daraus folgt die Behauptung. □

Im vorherigen Beispiel *Rechner transportieren* (siehe Abb. 6.2.1, Abb. 6.2.2) gilt offensichtlich:

$$bereit_a \wedge Signal_{ba} \leftrightarrow bereit_b \wedge Signal_{ab}$$

Deshalb erfüllt das System nach der Verteilung von Transition *Rechner tragen* die Blockbedingung.

Die Forderungen *iii* und *iv* im Satz 6.2.4 ist notwendig, wie das folgende Beispiel (Abb. 6.2.3) zeigt:

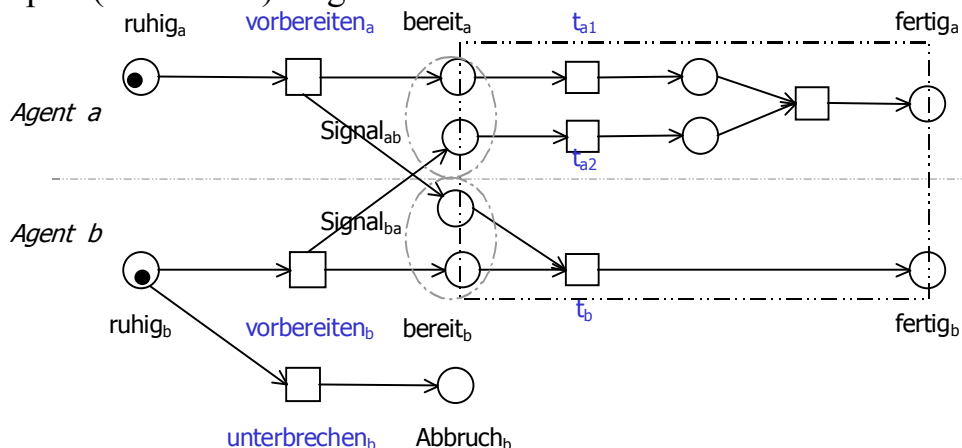


Abb. 6.2.3 Illustration für die Forderungen *iii* und *iv* im Satz 6.2.4

t_{a1} kann schalten auch wenn t_{a2} und t_b nie aktiviert werden, weil *unterbrechen_b* geschaltet hat.

Abschwächung der Voraussetzungen an die Umgebung im Satz 6.2.4

Im Satz 6.2.4 wird gefordert, dass die Agenten a und b keine Konflikt-Transitionen zu t haben. Diese Forderung kann wie folgt abgeschwächt werden: Die Agenten a und b haben zwar Konflikt-Transitionen zu t , diese Konflikt-Transitionen sind aber nicht aktiviert. In diesem abgeschwächten Fall erfüllt das erhaltene System nach der Verfeinerung die Blockbedingung, wenn das Ersetzungsnetz zusätzlich noch die kausale Abhängigkeit ($\bullet t \blacktriangleleft t \bullet$) erfüllt (siehe Abb. 6.2.4).

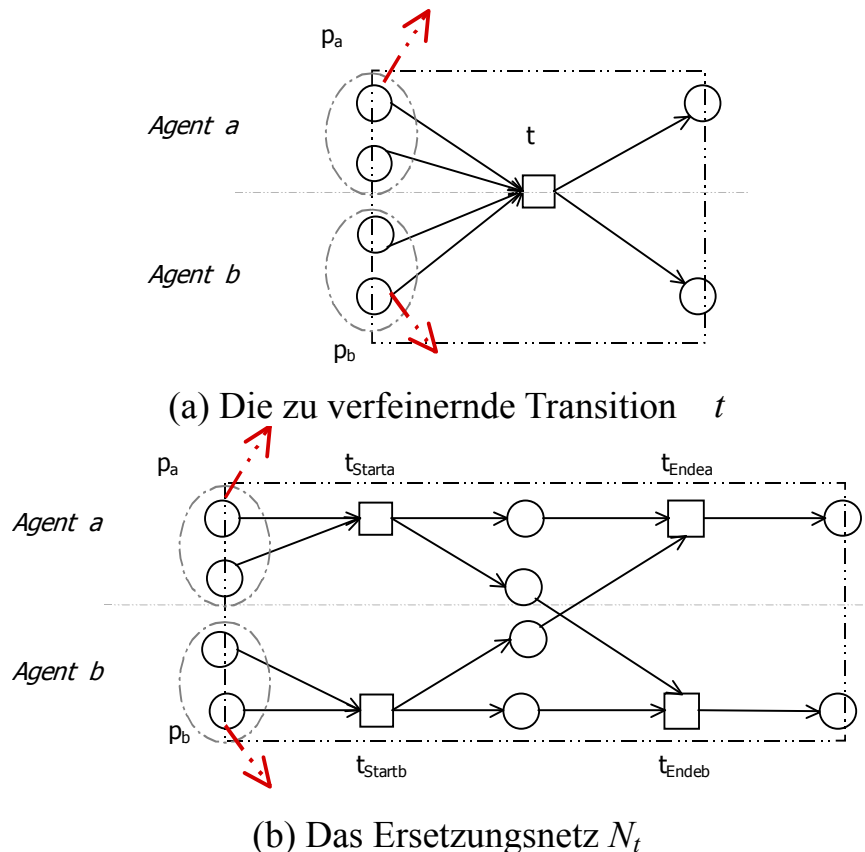


Abb. 6.2.4 Illustration für Satz 6.2.5 (Abschwächung von Satz 6.2.4)
(der rote Pfeil symbolisiert zu t konkurrente Transitionen, die nicht aktiviert sind, wenn t aktiviert ist)

Warum soll das Ersetzungsnetz die kausale Abhängigkeit $\bullet t \blacktriangleleft t \bullet$ erfüllen?

Das erklären wir durch ein Beispiel, für das nicht $\bullet t \blacktriangleleft t \bullet$ gilt und deshalb die Blockbedingung nicht erfüllt. In Abb. 6.2.5 ist ein System Σ_3 angegeben. In Σ_3 gibt es an der Stelle $bereit_b$ eine Konflikt-Transition $unterbrechen_b$. In Σ_3 ist diese Konflikt-Transition nie aktiviert.

In Abb. 6.2.6 ist ein Verfeinerungssystem Σ_3' mit einem Ersetzungsnetz N_{t_1} angegeben, das die kausale Abhängigkeit ($\bullet t \blacktriangleleft t \bullet$) nicht erfüllt. In Σ_3'

kann b diese Konflikt-Transition $unterbrechen_b$ ausführen, so dass die Blockbedingung nicht erfüllt ist. Wenn das Ersetzungsnetz die kausale Abhängigkeit ($\bullet t \blacktriangleleft t^*$) erfüllt, so wie N_t in Abb. 6.2.7, dann ist das nicht möglich, weil diese kausale Abhängigkeit sichert, dass a erst in den Zustand $fertig_a$ übergehen kann, wenn b schon mit einer Aktion in N_t angefangen hat.

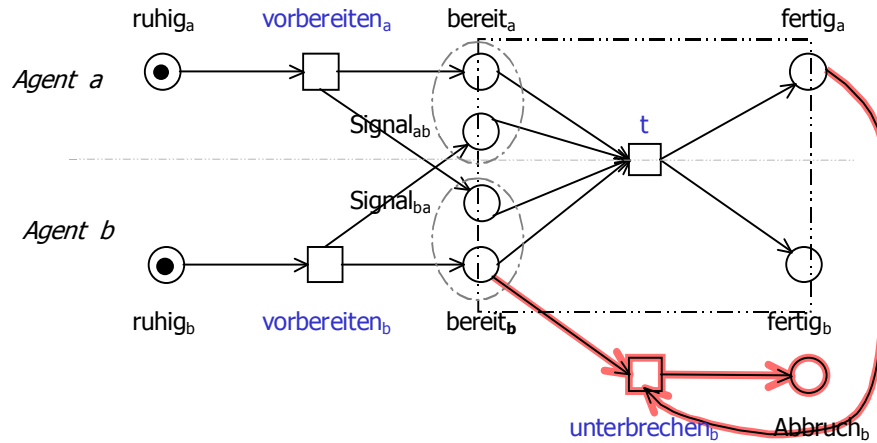


Abb. 6.2.5 System Σ_3 (Es gilt $\bullet t \blacktriangleleft t^*$)

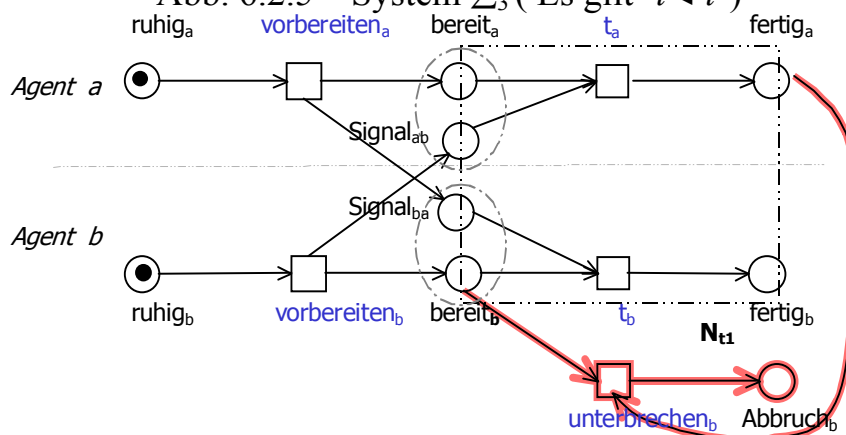


Abb. 6.2.6 Verfeinertes System Σ_3' (Es gilt nicht $\bullet t \blacktriangleleft t^*$)

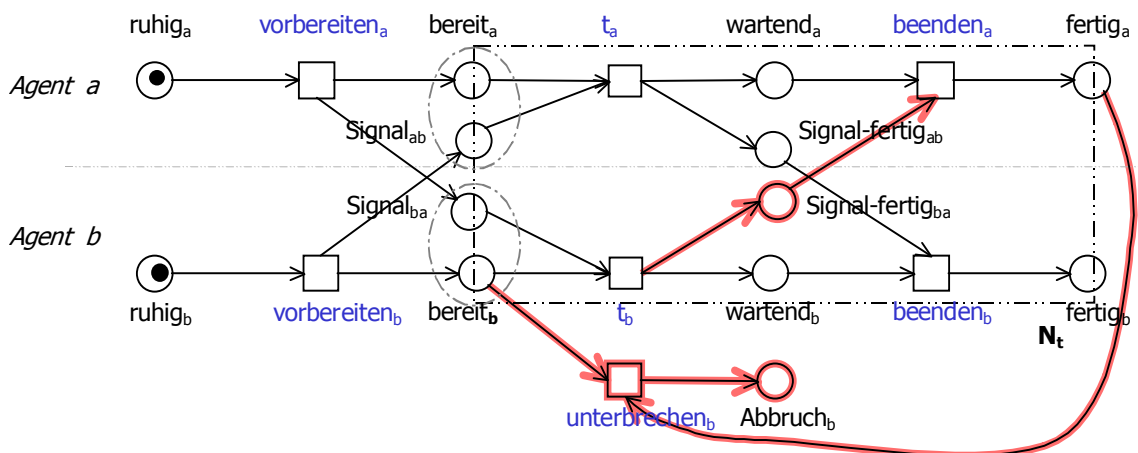


Abb. 6.2.7 Verfeinertes System Σ_3'' (Es gilt $\bullet t \blacktriangleleft t^*$)

Satz 6.2.5 (Abschwächung von Satz 6.2.4)

Gegeben sei das Agentensystem Σ mit Agentenmenge A und Kanalmenge C und die unverteilte Transition t . Sei $(\bullet t)_a := \bullet t \cap (a \cup C_a)$, $(\bullet t)_b := \bullet t \cap (b \cup C_b)$ und $\bullet t = (\bullet t)_a \cup (\bullet t)_b$, wobei $a, b \in A$.

In Σ gelte:

- i. $(\bullet t)_a \triangleright (\bullet t)_b$ und $(\bullet t)_b \triangleright (\bullet t)_a$;
- ii. Es gibt keine aktivierte konkurrente Transition zu $(\bullet t)_a$ und $(\bullet t)_b$.

Dann garantiert jedes Verfeinerungsnetz N_t mit folgenden Eigenschaften die Blockbedingung für $\Sigma' = \Sigma(N_t \setminus t)$:

In N_t gibt es Transitionen t_{Start_a} und t_{Start_b} mit:

- iii. $\bullet t_{Start_a} = (\bullet t)_a$, $\bullet t_{Start_b} = (\bullet t)_b$;
- iv. Es gibt keine konkurrenten Transitionen zu t_{Start_a} und t_{Start_b} in N_t ;
- v. $\bullet t \triangleleft t$ gilt in (N_t, t^-) .

Beweis:

Entsprechend dem Beweis zu Satz 6.2.4 können wir wieder davon ausgehen, dass, falls der Satz nicht gilt, es in einem Ablauf (K', r') einen Schnitt C gibt, in dem ein unvollständiger Ablauf beginnt. Also in C schaltet t_{Start_a} , $C' \subseteq C$ mit $r'(C') = \bullet t_{Start_a}$. Wegen der Folgerung von Bem. 6.2.1 folgt, auf diesen Schnitt folgt kein Schnitt, der C' enthält und in dem auch t_{Start_b} schaltet.

Nach Voraussetzung i gilt $(\bullet t)_a \triangleright (\bullet t)_b$ in Σ . Wegen der Voraussetzungen v und ii (also $\bullet t \triangleleft t$ und es gibt keine aktivierte konkurrente Transition zu $(\bullet t)_a$) gilt im Ablauf K' weiter $(\bullet t)_a \triangleright (\bullet t)_b$. Daraus folgt: Auf C folgt ein Schnitt D , in dem $(\bullet t)_a$ noch gilt, und auch $(\bullet t)_b$ gilt, d.h. $C' \subseteq D$ und $C'' \subseteq D$ mit $r'(C'') = (\bullet t)_b$.

In D ist nach Voraussetzung ii keine zu t_{Start_b} konkurrente Transition von Σ aktiviert. Auch in N_t gibt es keine konkurrenten Transitionen zu t_{Start_b} . Weiter kann keine der zu t_{Start_b} konkurrenten Transitionen von Σ aktiviert werden (wegen Voraussetzung v). Wegen Progressforderung für Abläufe muß also t_{Start_b} schalten. Das ist ein Widerspruch dazu, dass auf C kein Schnitt folgt, der C' enthält und in dem auch t_{Start_b} schaltet.

Daraus folgt die Behauptung. □

Im Folgenden werden wir noch eine Abschwächung 6.2.6 von Satz 6.2.4 angeben, die wir im Kapitel 7 beim Anwendungsbeispiel (vgl. Abschnitt 7.2.2) direkt verwenden können. Weiter werden wir ein Lemma über Erhaltenbleiben von Eigenschaften für die Verfeinerungsschritte angeben, bei denen die Bedingungen von Satz 6.2.6 erfüllt sind.

Satz 6.2.6 (*Abschwächung von Satz 6.2.4*)

Gegeben sei das Agentensystem Σ mit Agentenmenge A und Kanalmenge C und die unverteilte Transition t . Sei $(\bullet t)_a := \bullet t \cap (a \cup C_a)$, $(\bullet t)_b := \bullet t \cap (b \cup C_b)$ und $\bullet t = (\bullet t)_a \cup (\bullet t)_b$, wobei $a, b \in A$.

In Σ gelte:

- i. $(\bullet t)_a \leftrightarrow (\bullet t)_b$;
- ii. Es gibt keine aktivierte konkurrente Transition zu $(\bullet t)_a$ und $(\bullet t)_b$.

Dann garantiert jedes Verfeinerungsnetz N_t mit folgenden Eigenschaften die Blockbedingung für $\Sigma' = \Sigma(N_t \setminus t)$:

In N_t gibt es Transitionen t_{Start_a} und t_{Start_b} mit:

- iii. $\bullet t_{Start_a} = (\bullet t)_a$, $\bullet t_{Start_b} = (\bullet t)_b$;
- iv. Es gibt keine konkurrenten Transitionen zu t_{Start_a} und t_{Start_b} in N_t ;
- v. $\bullet t \blacktriangleleft t^\bullet$ gilt in (N_t, t^-) .

Beweis:

Entsprechend dem Beweis zu Satz 6.2.4 können wir wieder davon ausgehen, dass, falls der Satz nicht gilt, es in einem Ablauf (K', r') einen Schnitt C gibt, in dem t_{Start_a} schaltet aber nicht t_{Start_b} , obwohl nach Voraussetzung iii auch t_{Start_b} aktiviert ist.

In C ist nach Voraussetzung ii keine zu t_{Start_b} konkurrente Transition von Σ aktiviert. Auch in N_t gibt es keine konkurrenten Transitionen zu t_{Start_b} . Weiter kann keine der zu t_{Start_b} konkurrenten Transitionen von Σ aktiviert werden (wegen Voraussetzung v). Wegen Progressforderung für Abläufe muß also t_{Start_b} schalten. Wegen Bem. 6.2.1 ist das ein Widerspruch dazu, dass in C kein vollständiger Ablauf von (N_t, t^-) beginnt.

Daraus folgt die Behauptung. □

Nun geben wir ein Lemma über Erhaltenbleiben von Eigenschaften für die Verfeinerungsschritte, bei denen die Bedingungen von Satz 6.2.6 erfüllt sind. Vorher brauchen wir noch eine Bemerkung für den Beweis von diesem Lemma.

Für das System Σ_3'' im vorherigen Beispiel (vgl. 6.2.7) kann nicht sein, dass die Stelle $ruhig_a$ und die Stelle $wartend_b$ gleichzeitig markiert sind.

Bemerkung 6.2.7

Erfülle $(\Sigma \rightarrow \Sigma')$ mit $\Sigma' = \Sigma(N_t \setminus t)$ die Bedingungen von Satz 6.2.6.

Sei (K', r') mit $K' = (B', E', \leq')$ ein Ablauf von Σ' , sei K_{N_t} ein Auftreten von N_t in K' .

Seien $b_1, b_2 \in {}^\circ K_{N_t}$.

Dann gilt:

Für beliebige $b_1', b_2' \in B'$ mit $b_1' \leq'^+ b_1, b_2 \leq'^+ b_2'$ gilt: $b_1' \leq'^+ b_2'$.

Beweis:

Sei (K, r) mit $K = (B, E, \leq)$ ein Ablauf von Σ mit: (K', r') ist eine $t \rightarrow N_t$ Substitution von (K, r) .

Seien $p_a := (\bullet t)_a, p_b := (\bullet t)_b$.

Dann gilt: $B_{p_a} \subseteq {}^\circ K_{N_t}$ und $B_{p_b} \subseteq {}^\circ K_{N_t}$, wobei $r'(B_{p_a}) = p_a, r'(B_{p_b}) = p_b$.

Angenommen, $b_1 \in B_{p_a}$ und $b_2 \in B_{p_b}$.

Weiter nehmen wir an: $b_1' \leq' e_1' \leq' b_1, b_2 \leq' e_2' \leq' b_2'$, wobei $e_1', e_2' \in E'$.

Dann gilt: (wegen Bem. 3.2.5)

${}^\circ K_{N_t} \subseteq B$;

$B_{p_a} \subseteq B$ und $r(B_{p_a}) = p_a$;

$B_{p_b} \subseteq B$ und $r(B_{p_b}) = p_b$;

$b_1' \in B$ und $b_1' \leq^+ b_1, b_1 \in B_{p_a}$.

Zu zeigen ist: $b_1' \leq'^+ b_2'$ (1)

Wir führen den Beweis indirekt, angenommen, $\neg (b_1' \leq'^+ b_2')$.

Dann gilt: $b_2' \leq'^+ b_1'$ oder $b_1' \underline{co} b_2'$.

Fall 1. $b_2' \leq'^+ b_1'$.

Dann gilt $b_2 \leq'^+ b_2' \leq'^+ b_1' \leq'^+ b_1$.

Also $b_2 \leq'^+ b_1$.

Das ist ein Widerspruch zu $b_2 \underline{\text{co}} b_1$ ($b_2 \underline{\text{co}} b_1$ gilt wegen $b_1, b_2 \in {}^\circ K_{N_t}$).

Fall 2. $b_1' \underline{\text{co}} b_2'$.

Dann gilt: $b_1' \underline{\text{co}} b$ für jedes $b \in B_{p_b} \subseteq {}^\circ K_{N_t}$,

(weil: - Wenn $b_1' \leq'^+ b$ gilt, so gilt $b_1' \leq'^+ b_2'$.

- Wenn $b \leq'^+ b_1'$ gilt, so gilt $b \leq'^+ b_1$).

Dann gilt auch in K : $b_1' \underline{\text{co}} b$ für jedes $b \in B_{p_b}$, (wegen Satz 3.5.1)

(weil für jedes $b \in B_{p_b}$ gilt: $b_1' \leq^+ b \rightarrow b_1' \leq'^+ b$).

Daraus folgt:

Es gibt einen Schnitt C in K , der b_1' und B_{p_b} enthält,

d.h. $b_1' \in C$ und $B_{p_b} \subseteq C$. (2)

Wegen $b_1' \leq^+ b_1$ und $b_1 \in B_{p_a}$ gilt:

C enthält nicht B_{p_a} ,

d.h. $\neg (B_{p_a} \subseteq C)$ (3)

Aus (2), (3) folgt:

Es gibt in K einen Zustand, in dem p_b gilt, aber p_a nicht gilt.

Das ist ein Widerspruch zu $\Sigma \models \Box (p_b \rightarrow p_a)$.

Daraus folgt: (1) gilt. □

Satz 6.2.8

Sei $\Sigma = ((P, T, F), m_0)$ ein System.

Erfülle $(\Sigma \rightarrow \Sigma')$ mit $\Sigma' = \Sigma(N_t \setminus t)$ die Bedingungen von Satz 6.2.6.

Sei $q \in \bullet t$, $Q \subseteq P$ mit: $\Sigma \models \Box (Q \rightarrow \neg p)$ für ein $p \in \bullet t$.

Dann gilt:

Beim Verfeinerungsschritt $(\Sigma \rightarrow \Sigma')$ bleibt die Lebendigkeitseigenschaft

$Q \triangleright q$ erhalten.

Beweis:

Seien $(K, r), (K', r')$ Abläufe von Σ, Σ' mit: (K', r') ist eine $t \rightarrow N_t$

Substitution von (K, r) , wobei $K = (B, E, \leq), K' = (B', E', \leq')$.

Angenommen, in K gilt: $Q \triangleright q$ (1)

Zu zeigen ist: In K' gilt $Q \triangleright q$ weiter. (2)

Sei $B_Q \subseteq B$ mit $r(B_Q) = Q$

und $b_q \in B$ mit $r(b_q) = q$ und $b_q \leq e$ wobei $r(e) = t$.

Sei C ein Schnitt in K mit $B_Q \subseteq C$.

Wegen (1) gilt:

Es gibt ein Auftreten b von q (d.h. $r(b) = q$) mit $C \text{ [co } b$.

Weil der Unterschied von K' zu K nur bei Auftreten von t ist, betrachten wir in K nur solche Auftreten von q , auf die Auftreten von t folgen.

Angenommen, $C \text{ [co } b_q$ gilt in K .

Dann gilt für jedes $b \in B_Q$ und $b' \in \bullet e$:

$$b \leq^+ b' \text{ oder } b \underline{\text{co}} b' \quad (3)$$

Wegen Bem. 3.2.5 gilt:

$$B_Q \subseteq B' \text{ und } r'(B_Q) = Q$$

$$b_q \in B' \text{ und } r'(b_q) = q$$

und $b_q \in {}^\circ K_{N_t}$ wobei K_{N_t} ein Auftreten von N_t in K' ist.

wegen $\Sigma \models \square (Q \rightarrow \neg p)$ gilt:

Es gibt ein $b_l \in B_Q$ so dass $\neg (b_l \underline{\text{co}} b_p)$, wobei $b_p \in \bullet e$ und $r(b_p) = p$.

Aus (3) folgt:

$$b_l \leq^+ b_p \quad (4)$$

Wegen Bem. 3.2.5 gilt:

$$b_p \in B' \text{ und } b_p \in {}^\circ K_{N_t}$$

Aus Satz 3.5.1 folgt:

$$b_l \leq'^+ b_p$$

Für ein beliebiges $b' \in B'$ mit $b_q \leq^+ b'$ gilt:

$$b_l \leq^+ b' \quad (\text{wegen Bem. 6.2.7})$$

Daraus folgt:

Für jeden Schnitt C' in K' mit $B_Q \subseteq C'$ gilt: $C' \text{ [co } b_q$.

Daraus folgt:

In K' gilt $Q \triangleright q$ weiter. \square

6.3 Erbetene Mithilfe

Im vorherigen Abschnitt war die Umgebung der zu verteilenden Transition t wie folgt: die beiden Agenten geben vorher dem Partner ein Signal dafür, dass sie für die gemeinsame Aktion bereit sind. In diesem Abschnitt betrachten wir die folgende Umgebung: Nur ein Agent von den beiden, z.B. Agent b , gibt vorher seinem Partner, also Agent a , ein Signal dafür, dass er für die gemeinsame Aktion t bereit ist. Ob diese Aktion dann tatsächlich ausgeführt wird, das wird dann der Agent a entscheiden. Ein Agent bittet also den anderen um Mithilfe, weiß aber nicht, ob ihm der andere helfen wird.

Welche Bedingungen müssen für das Ersetzungsnetz gelten, damit das erhaltene System nach der Verteilung in einer solchen Umgebung die Blockbedingung erfüllt? Weil a dafür entscheidend ist, ob sie überhaupt die gemeinsame Aktion t ausführen werden, soll Agent b seine Teilaufgabe von t nicht ausführen, bevor a sich für t entschieden hat. a muss anschließend im Verfeinerungsnetz mit Hilfe einer Nachricht seine Entscheidung dem Agenten b mitteilen.

Wir sehen uns ein Beispiel an. In Abb.6.3.1 ist ein System, in dem es zwei Agenten – ein *Kind* und seine *Mama* – gibt. Wenn das Kind *Schokolade essen möchte*, dann guckt er ganz liebevoll seine Mama an. Das ist ein *Zeichen* für Schokolade. Wenn die Mama dieses Zeichen sieht und sie *Schokolade gekauft* hat, dann *gibt* sie ihrem Kind *Schokolade*, und es isst dann die Schokolade. Aber wenn die Mama nicht Schokolade sondern *Kaugummi gekauft* hat, dann kann sie ihm nicht Schokolade geben und er kann keine Schokolade essen, also die Transition *Schokolade geben bzw. nehmen* kann nicht schalten.

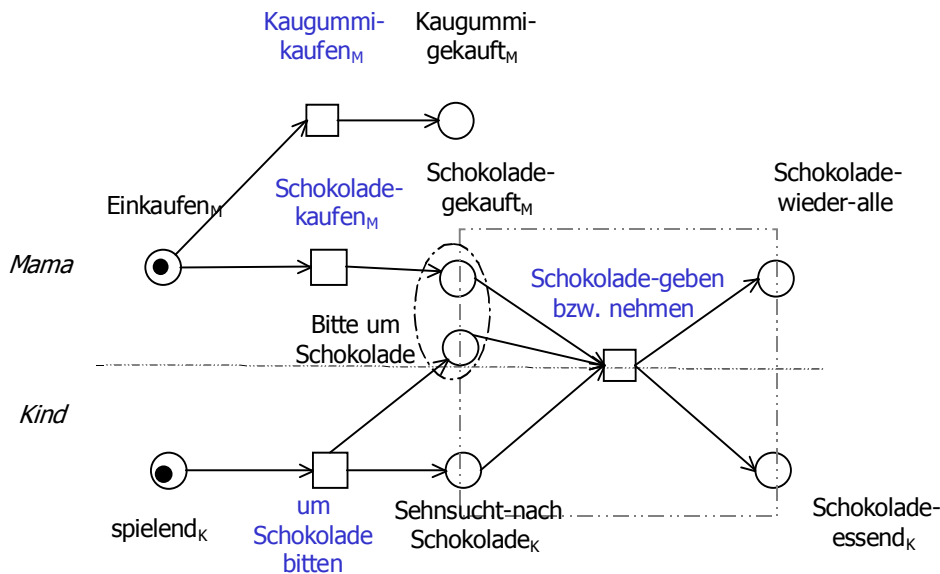


Abb. 6.3.1 Das System *Schokolade*

In Abb. 6.3.2 wird diese Transition in zwei Transitionen – die Transition *Schokolade geben* vom Agenten *Mama* und die Transition *Schokolade nehmen* vom Agenten *Kind* geteilt.

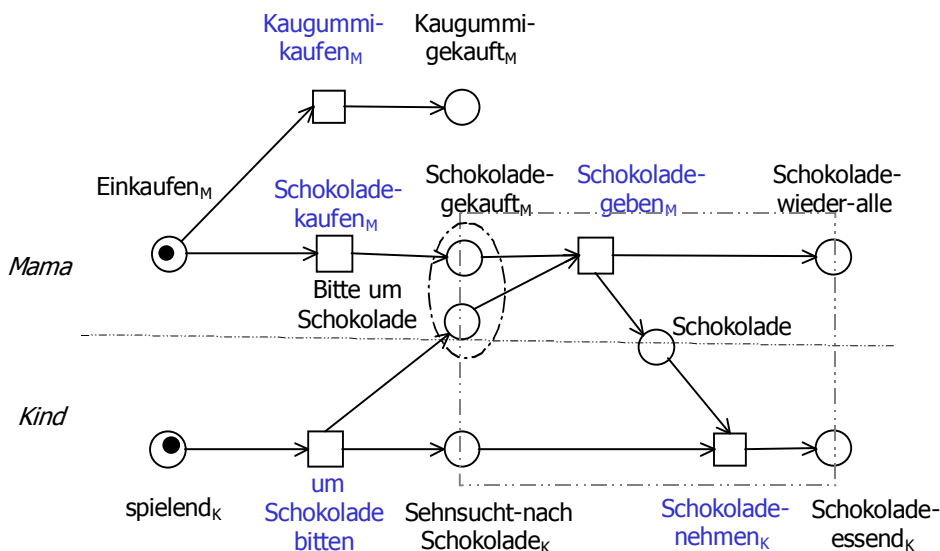


Abb. 6.3.2 Verteiltes System *Schokolade*

Auch im verteilten System *Schokolade* ist es so: Wenn das *Kind* die *Mama* liebevoll anguckt, dann weiß die *Mama*, dass ihr *Kind* *Schokolade essen möchte*. Wenn sie *Schokolade* hat, dann gibt sie ihm *Schokolade*. Es nimmt und isst dann die *Schokolade*.

Das verteilte System *Schokolade* erfüllt offenbar die Blockbedingung, d.h. es kann nicht passieren, dass die *Mama* dem *Kind* *Schokolade gibt*, es sie aber gar nicht möchte, oder dass die *Mama* ihm nicht *Schokoladen*

gegeben hat, er aber Schokolade isst.

Satz 6.3.1

Gegeben sei das Agentensystem Σ mit Agentenmenge A und Kanalmenge C und die unverteilte Transition t . Sei $(\bullet t)_a := \bullet t \cap (a \cup C_a)$, $(\bullet t)_b := \bullet t \cap (b \cup C_b)$ und $\bullet t = (\bullet t)_a \cup (\bullet t)_b$, wobei $a, b \in A$.

In Σ gelte:

- i. $(\bullet t)_a \rightarrow (\bullet t)_b$;
- ii. Es gibt keine konkurrente Transition t .

Dann garantiert jedes Verfeinerungsnetz N_t mit folgenden Eigenschaften die Blockbedingung für $\Sigma' = \Sigma(N_t \setminus t)$:

In N_t gibt es Transitionen t_{Start_a} und t_{Start_b} mit:

- iii. $\bullet t_{Start_a} = (\bullet t)_a$, $\bullet t_{Start_b} \supset (\bullet t)_b$ und $\bullet t_{Start_b} \subseteq (\bullet t)_b \cup \bullet t_{Start_a}$;
- iv. Es gibt keine konkurrenten Transitionen zu t_{Start_a} und t_{Start_b} in N_t ;
- v. $t_{Start_a} \blacktriangleleft t_{Start_b}$ gilt in (N_t, t^-) .

Beweis:

Zu zeigen ist: Jeder Ablauf von Σ' wird aus einem Ablauf von Σ erhalten, indem jedes Auftreten von t durch einen vollständigen Ablauf von (N_t, t^-) ersetzt wird.

Wir nehmen an, es gibt einen Ablauf (K', r') von Σ' , für den das nicht der Fall ist. Diese Annahme ist zum Widerspruch zu führen.

Aus der Annahme folgt: (K', r') enthält einen unvollständigen Ablauf von (N_t, t^-) . Es gibt einen Schnitt C , in dem t_{Start_a} schaltet (da t_{Start_a} wegen der Voraussetzung $t_{Start_a} \blacktriangleleft t_{Start_b}$ die erste aktivierte Transition in N_t ist), also $C' \subseteq C$ mit $r'(C') = \bullet t_{Start_a}$. Nach Voraussetzung $(\bullet t)_a \rightarrow (\bullet t)_b$ gilt auch $(\bullet t)_b$, also $C'' \subseteq C$ mit $r'(C'') = (\bullet t)_b$.

t_{Start_b} ist aktiviert, wenn t_{Start_a} geschaltet hat. Es ist daher jetzt der Schnitt zu betrachten, der aus C durch Schalten von t_{Start_a} entsteht.

Angenommen, E ist der Schnitt mit $C \xrightarrow{e_{Start_a}} E$, wobei $r'(e_{Start_a}) = t_{Start_a}$.

Also gilt: $E = (C \setminus \bullet e_{Start_a}) \cup e_{Start_a}$.

Dann gilt $C'' \subseteq E$ (wegen $(\bullet t)_b \cap (\bullet t_{Start_a} \cup t_{Start_a} \bullet) = \emptyset$).

Im Zustand $r'(E)$ ist die Transition t_{Start_b} aktiviert (wegen $\bullet t_{Start_b} \subseteq (\bullet t)_b \cup t_{Start_a} \bullet$). Da es keine konkurrenten Transitionen zu $(\bullet t)_b$ von Σ und t_{Start_b} in N_t gibt, schaltet auch t_{Start_b} im Zustand $r'(E)$.

In C schalten also t_{Start_a} und dann t_{Start_b} . Nach Bem. 6.2.1 beginnt damit in C ein vollständiger Ablauf von (N_t, t^-) . Das ist ein Widerspruch dazu, dass in C kein vollständiger Ablauf von (N_t, t^-) beginnt.

Daraus folgt die Behauptung. □

Im vorherigen Beispiel *Schokolade* (siehe Abb. 6.3.1 und Abb. 6.3.2) gilt offensichtlich:

$$\text{Schokolade-gekauft}_M \wedge \text{Bitte-um-Schokolade}_M \rightarrow \text{Sehnsucht-nach-Schokolade}_K$$

und es gibt keine konkurrenten Transitionen zu *Sehnsucht-nach-Schokolade*_K und in der Verfeinerung zu *Schokolade-geben*_M und *Schokolade-nehmen*_K.

Abschwächung der Voraussetzungen an die Umgebung im Satz 6.3.1

Im Satz 6.3.1 wird gefordert, dass der Agent b keine Konflikt-Transitionen zu t hat. Diese Forderung kann wie folgt abgeschwächt werden: Der Agent b hat zwar Konflikt-Transitionen zu t , diese Konflikt-Transitionen sind aber nicht aktiviert. In diesem abgeschwächten Fall erfüllt das erhaltene System nach der Verfeinerung die Blockbedingung, wenn das Ersetzungsnetz zusätzlich noch die kausale Abhängigkeit ($\bullet t \blacktriangleleft t \bullet$) erfüllt (siehe Abb. 6.3.5).

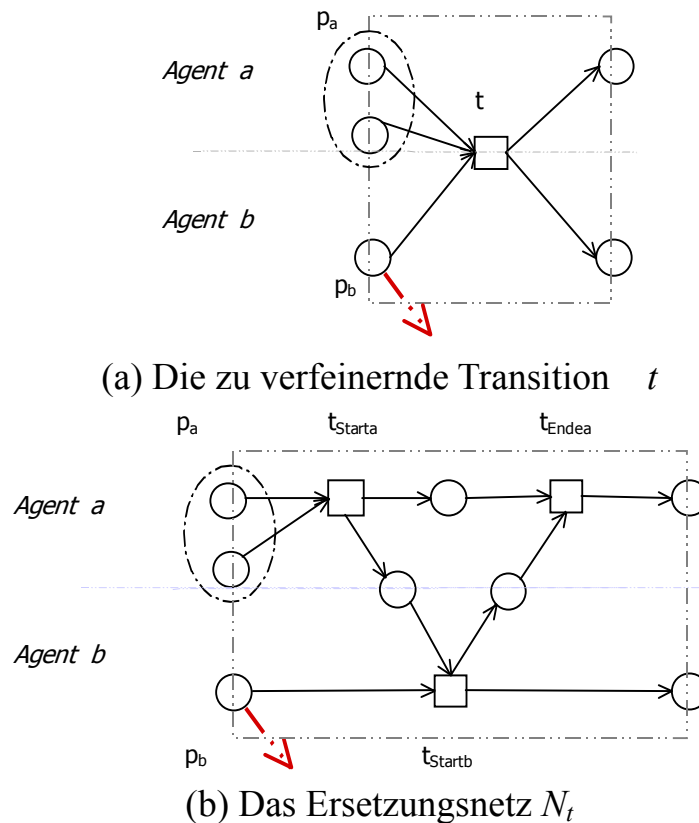


Abb. 6.3.5 Illustration für Satz 6.3.2 (Abschwächung von Satz 6.3.1) (der rote Pfeil symbolisiert zu t konkurrente Transitionen, die nicht aktiviert sind, wenn t aktiviert ist)

Satz 6.3.2 (Abschwächung von Satz 6.3.1)

Gegeben sei das Agentensystem Σ mit Agentenmenge A und Kanalmenge C und die unverteilte Transition t . Sei $(\bullet t)_a := \bullet t \cap (a \cup C_a)$, $(\bullet t)_b := \bullet t \cap (b \cup C_b)$ und $\bullet t = (\bullet t)_a \cup (\bullet t)_b$, wobei $a, b \in A$.

In Σ gelte:

- i. $(\bullet t)_a \triangleright (\bullet t)_b$;
- ii. Es gibt keine aktivierten konkurrenten Transitionen zu $(\bullet t)_a$ und $(\bullet t)_b$.

Dann garantiert jedes Verfeinerungsnetz N_t mit folgenden Eigenschaften die Blockbedingung für $\Sigma' = \Sigma(N_t \setminus t)$:

In N_t gibt es Transitionen t_{Start_a} und t_{Start_b} mit:

- iii. $\bullet t_{Start_a} = (\bullet t)_a$, $\bullet t_{Start_b} \supset (\bullet t)_b$ und $\bullet t_{Start_b} \subseteq (\bullet t)_b \cup t_{Start_a} \bullet$;
- iv. Es gibt keine konkurrenten Transitionen zu t_{Start_a} und t_{Start_b} in N_t ;
- v. $t_{Start_a} \blacktriangleleft t_{Start_b}$ gilt in (N_t, t^-) ;
- vi. $\bullet t \blacktriangleleft t \bullet$ gilt in (N_t, t^-) .

Beweis:

Entsprechend dem Beweis zu Satz 6.3.1 können wir wieder davon ausgehen, dass, falls der Satz nicht gilt, es in einem Ablauf (K', r') einen Schnitt C gibt, in dem ein unvollständiger Ablauf beginnt. Also in C schaltet t_{Start_a} (da t_{Start_a} wegen der Voraussetzung $t_{Start_a} \blacktriangleleft t_{Start_b}$ die erste aktivierte Transition in N_t ist), also $C' \subseteq C$ mit $r'(C') = \bullet t_{Start_a}$. Wegen der Folgerung von Bem. 6.2.1 folgt, auf diesen Schnitt folgt kein Schnitt, der C' enthält und in dem auch t_{Start_b} schaltet.

Nach Voraussetzung i gilt $(\bullet t)_a \triangleright (\bullet t)_b$ in Σ . Wegen den Voraussetzungen vi und ii (also $\bullet t \blacktriangleleft t \bullet$ und es gibt keine aktivierte konkurrente Transition zu $(\bullet t)_a$) gilt weiter $(\bullet t)_a \triangleright (\bullet t)_b$ in K' . Daraus folgt: Auf C folgt ein Schnitt D , in dem $(\bullet t)_a$ noch gilt, und auch $(\bullet t)_b$ gilt, d.h. $C' \subseteq D$ und $C'' \subseteq D$ mit $r'(C'') = (\bullet t)_b$.

Angenommen, E ist ein Schnitt mit $D \xrightarrow{e_{Start_a}} E$, wobei $r'(e_{Start_a}) = t_{Start_a}$. (d.h. E ist der Schnitt nach dem Auftreten von t_{Start_a} in D).

Also gilt: $E = (D \setminus \bullet e_{Start_a}) \cup e_{Start_a} \bullet$.

Dann gilt $C'' \subseteq E$ (wegen $(\bullet t)_b \cap (\bullet t_{Start_a} \cup t_{Start_A} \bullet) = \emptyset$).

Im Zustand $r'(E)$ ist die Transition t_{Start_b} aktiviert (wegen $\bullet t_{Start_b} \subseteq (\bullet t)_b \cup t_{Start_a} \bullet$).

Nach Voraussetzung *ii* ist keine zu $(\bullet t)_b$ konkurrente Transition von Σ aktiviert. Auch in N_i gibt es keine konkurrenten Transitionen zu t_{Start_b} . Weiter kann wegen Voraussetzung *vi* keine der zu t_{Start_b} konkurrenten Transitionen von Σ im Schnitt E aktiviert werden. Wegen Progressforderung für Abläufe muß also t_{Start_b} schalten. Das ist ein Widerspruch dazu, dass auf C kein Schnitt folgt, der C' enthält und in dem auch t_{Start_b} schaltet.

Daraus folgt die Behauptung. □

6.4 Erwartete Mithilfe

Im vorherigen Abschnitt hatte die zu verfeinernde Transition folgende Umgebung: Ein Agent, z.B. Agent b gibt dem Agenten a ein Signal, wenn er bereit ist, die gemeinsame Aktion auszuführen. Das System hatte damit die Eigenschaft, dass alle Stellen von $(\bullet t)_b$ belegt waren, wenn alle Stellen von $(\bullet t)_a$ belegt waren. Für eine solche Umgebung erfüllte das erhaltene System nach der Verfeinerung die Blockbedingung, wenn das Ersetzungsnetz die kausale Abhängigkeit $(t_{Start_A} \blacktriangleleft t_{Start_b})$ erfüllte.

In diesem Abschnitt betrachten wir folgende Umgebung: Keiner von den beiden gibt vorher dem anderen Agenten ein Signal. Aber das System hat folgende Eigenschaft: Wenn der eine, z.B. Agent a , bereit (für die gemeinsame Aktion t) ist, dann wird der andere, also Agent b , irgendwann auch bereit sein.

Für eine solche Umgebung, erfüllt das erhaltene System nach der Verfeinerung die Blockbedingung, wenn das Ersetzungsnetz neben der kausalen Abhängigkeit $(t_{Start_A} \blacktriangleleft t_{Start_b})$, auch noch die kausale Abhängigkeit $(\bullet t \blacktriangleleft t \bullet)$ erfüllt.

Wir sehen uns wieder ein Beispiel an. In Abb. 6.4.1 ist ein System Σ_1 angegeben. Agent a kann für t bereit sein, kann aber auch *unterbrechen* und in den Zustand $Abbruch_a$ übergehen. Aber wenn a für t bereit ist, dann wird b irgendwann auch für t bereit sein. Wenn die beiden bereit sind, dann werden sie gemeinsam die Transition t ausführen.

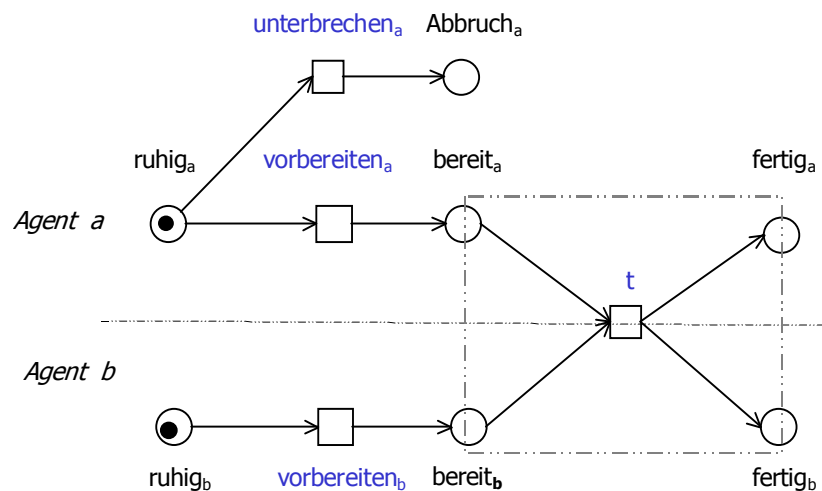


Abb. 6.4.1 System Σ_1

In Abb. 6.4.2 ist das System Σ_1' nach der Verteilung von t angegeben.

Wenn a für t bereit ist, erwartet er, dass b irgendwann auch dafür bereit sein wird. Er beginnt gleich seine Teilaufgabe von t und sendet dabei ein Signal an b , um Bescheid zu geben, dass er schon mit t angefangen hat. Wenn b bereit ist und dieses Signal bekommt, dann beginnt er auch seine Teilaufgabe von t , dabei sendet er auch ein Signal an a , um ihm mitzuteilen, dass auch er schon mit t angefangen hat. Wenn a fertig ist und dieses Signal bekommt, dann kann er in den Zustand $fertig_a$ übergehen. Das erhaltene System Σ_1' erfüllt die Blockbedingung.

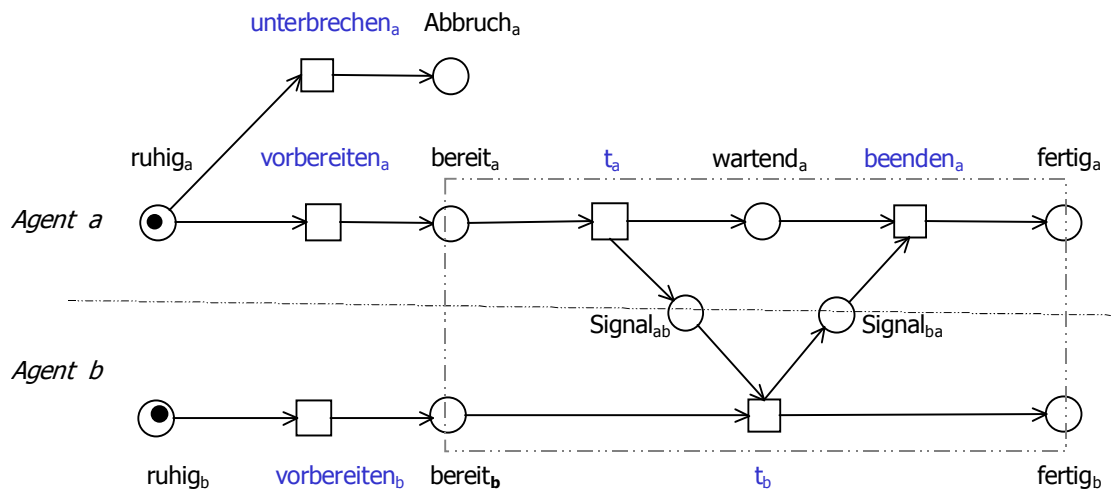


Abb. 6.4.2 System Σ_1'

Warum soll das Ersetzungsnetz die kausale Abhängigkeit $t_a \blacktriangleleft t_b$ erfüllen? Wenn das Ersetzungsnetz diese kausale Abhängigkeit nicht erfüllen würde, dann könnte Folgendes passieren: a unterbricht (Transition $unterbrechen_a$), b führt aber einseitig seine Teilaufgabe t_b aus (Transition t_b).

Warum soll das Ersetzungsnetz die kausale Abhängigkeit ($\bullet t \blacktriangleleft t \bullet$) erfüllen? Das erklären wir durch ein weiteres Beispiel. In Abb. 6.4.3 ist ein System Σ_2 angegeben. Σ_2 hat eine sogenannte Rückkopplung: An der Stelle $ruhig_b$ gibt es eine Konflikt-Transition $unterbrechen_b$. In Σ_2 ist diese

Konflikt-Transition nie aktiviert, deshalb gilt $bereit_a \triangleright bereit_b$.

In Abb. 6.4.4 ist ein Verfeinerungssystem Σ_2' mit einem Ersetzungsnetz N_{t_t} angegeben, das die kausale Abhängigkeit ($\bullet t \blacktriangleleft t \bullet$) nicht erfüllt. In Σ_2' kann passieren, dass a die Teilaufgabe t_a von t ausgeführt hat, b aber unterbricht, so dass die Blockbedingung nicht erfüllt ist. Wenn das Ersetzungsnetz die kausale Abhängigkeit ($\bullet t \blacktriangleleft t \bullet$) erfüllt, so wie N_t in Abb. 6.4.5, dann kann man das vermeiden, weil diese kausale Abhängigkeit sichert, dass a erst in den Zustand $fertig_a$ übergehen kann, wenn b schon mit einer Aktion in N_t angefangen hat.

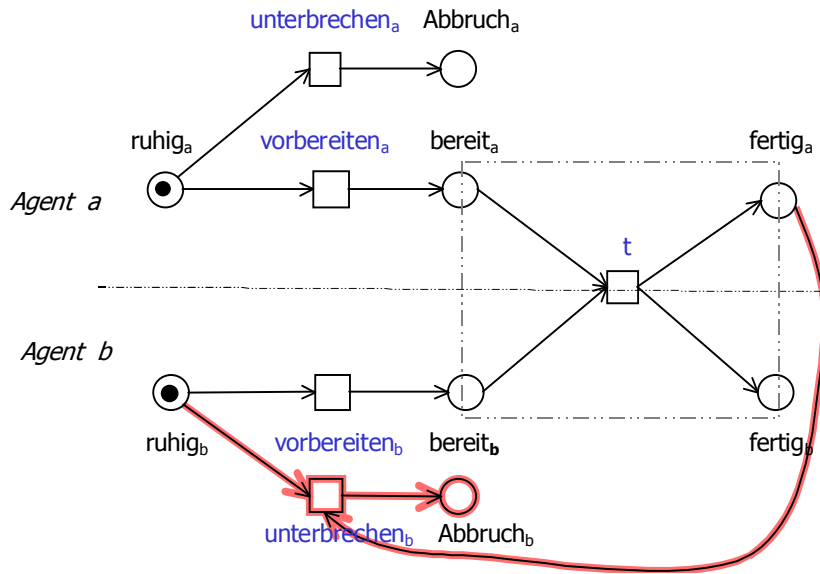


Abb. 6.4.3 System Σ_2 (Illustration für $\bullet t \blacktriangleleft t \bullet$ zu Satz 6.4.2)

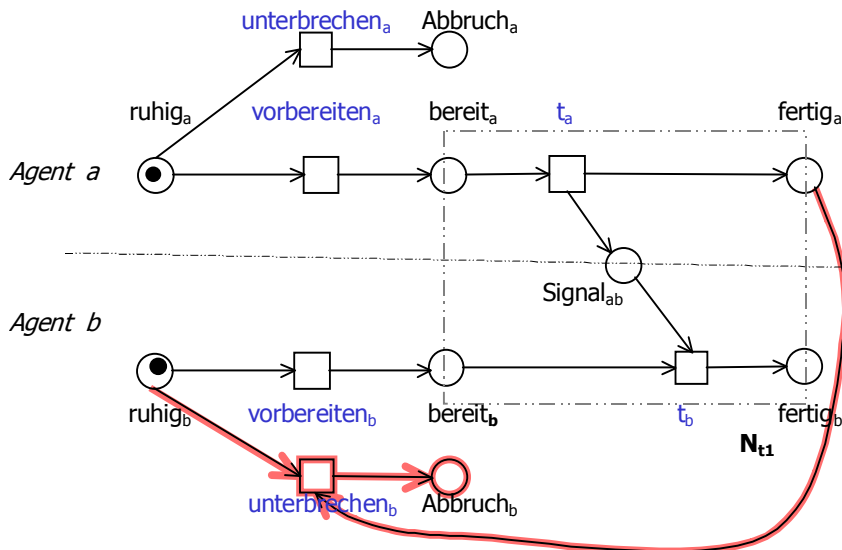


Abb. 6.4.4 System Σ_2' (Illustration für $\bullet t \blacktriangleleft t \bullet$ zu Satz 6.4.2)

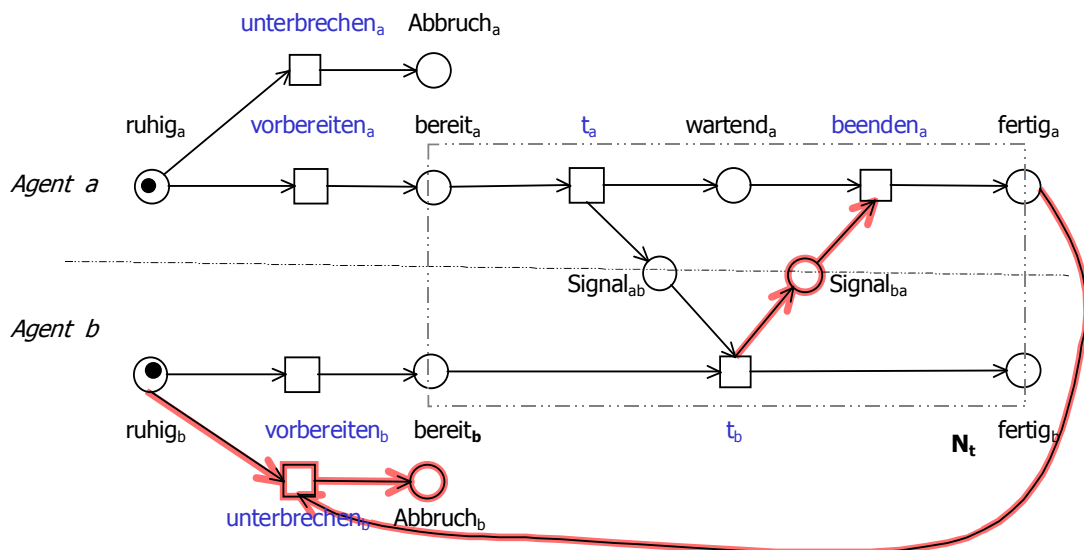


Abb. 6.4.5 System Σ_2'' (Illustration für $\bullet t \blacktriangleleft t^\bullet$ zu Satz 6.4.2)

Nun können wir den entsprechenden Satz angeben.

Satz 6.4.1

Gegeben sei das Agentensystem Σ mit Agentenmenge A und Kanalmenge C und die unverteilter Transition t . Sei $(\bullet t)_a := \bullet t \cap (a \cup C_a)$, $(\bullet t)_b := \bullet t \cap (b \cup C_b)$ und $\bullet t = (\bullet t)_a \cup (\bullet t)_b$, wobei $a, b \in A$.

In Σ gelte:

- i. $(\bullet t)_a \triangleright (\bullet t)_b$;
- ii. Es gibt keine konkurrente Transition zu t .

Dann garantiert jedes Verfeinerungsnetz N_t mit folgenden Eigenschaften die Blockbedingung für $\Sigma' = \Sigma(N_t \setminus t)$:

In N_t gibt es Transitionen t_{Start_a} und t_{Start_b} mit:

- iii. $\bullet t_{Start_a} = (\bullet t)_a$, $\bullet t_{Start_b} \supset (\bullet t)_b$ und $\bullet t_{Start_b} \subseteq (\bullet t)_b \cup t_{Start_a}^\bullet$;
- iv. Es gibt keine konkurrenten Transitionen zu t_{Start_a} und t_{Start_b} in N_t ;
- v. $t_{Start_a} \blacktriangleleft t_{Start_b}$ gilt in (N_b, t^-) ;
- vi. $\bullet t \blacktriangleleft t^\bullet$ gilt in (N_b, t^-) .

Beweis:

Zu zeigen ist: Jeder Ablauf von Σ' wird aus einem Ablauf von Σ erhalten, in dem jedes Auftreten von t durch einen vollständigen Ablauf von (N_b, t^-) ersetzt wird.

Wir nehmen an, es gibt einen Ablauf (K', r') von Σ' , für den das nicht der Fall ist. Diese Annahme ist zum Widerspruch zu führen.

Aus der Annahme folgt: (K', r') enthält einen unvollständigen Ablauf von (N_b, t^-) . Es gibt einen Schnitt C , in dem t_{Start_a} schaltet (da t_{Start_a} wegen der Voraussetzung $t_{Start_a} \blacktriangleleft t_{Start_b}$ die erste aktivierte Transition in N_t ist), also $C' \subseteq C$ mit $r'(C') = \bullet t_{Start_a}$. Wegen der Folgerung von Bem. 6.2.1 folgt, auf diesen Schnitt folgt kein Schnitt, der C' enthält und in dem auch t_{Start_b} schaltet.

Nach Voraussetzung i gilt $(\bullet t)_a \triangleright (\bullet t)_b$ in Σ . Wegen der

Voraussetzungen *vi* und *ii* (also $\bullet t \triangleleft t^\bullet$ und es gibt keine konkurrente Transition zu $(\bullet t)_a$) gilt im Ablauf K' weiter $(\bullet t)_a \triangleright (\bullet t)_b$. Daraus folgt: Auf C folgt ein Schnitt D , in dem $(\bullet t)_a$ noch gilt, und auch $(\bullet t)_b$ gilt, d.h. $C' \subseteq D$ und $C'' \subseteq D$ mit $r'(C'') = (\bullet t)_b$.

t_{Start_b} ist aktiviert, wenn t_{Start_a} geschaltet hat. Es ist daher jetzt der Schnitt zu betrachten, der aus D durch Schalten von t_{Start_a} entsteht.

Angenommen, E ist der Schnitt mit $D \xrightarrow{e_{Start_a}} E$, wobei $r'(e_{Start_a}) = t_{Start_a}$. Also gilt: $E = (D \setminus \bullet e_{Start_a}) \cup e_{Start_a}$.

Dann gilt $C'' \subseteq E$ (wegen $(\bullet t)_b \cap (\bullet t_{Start_a} \cup t_{Start_a}^\bullet) = \emptyset$).

Im Zustand $r'(E)$ ist die Transition t_{Start_b} aktiviert (wegen $\bullet t_{Start_b} \subseteq (\bullet t)_b \cup t_{Start_a}^\bullet$). Da es keine konkurrente Transition zu $(\bullet t)_b$ von Σ und t_{Start_b} in N_i gibt, schaltet auch t_{Start_b} im Zustand $r'(E)$.

In D schaltet also auch t_{Start_b} . Das ist ein Widerspruch dazu, dass auf C kein Schnitt folgt, der C' enthält und in dem auch t_{Start_b} schaltet.

Daraus folgt die Behauptung. □

Abschwächung der Voraussetzungen an die Umgebung im Satz 6.4.1

Im Satz 6.4.1 wird gefordert, dass Agenten a und b keine Konflikt-Transitionen zu t haben. Diese Forderung kann wie folgt abgeschwächt werden: Agent b hat zwar Konflikt-Transitionen zu t , die Transition t ist aber *fair*. Und Agent a hat zwar Konflikt-Transitionen, diese Konflikt-Transitionen sind aber nicht aktiviert. In diesem abgeschwächten Fall erfüllt das erhaltene System nach der Verfeinerung die Blockbedingung, wenn das Ersetzungsnetz zusätzlich noch die Bedingung erfüllt, dass die Anfangstransition vom Agenten b *fair* ist (siehe Abb. 6.4.6).

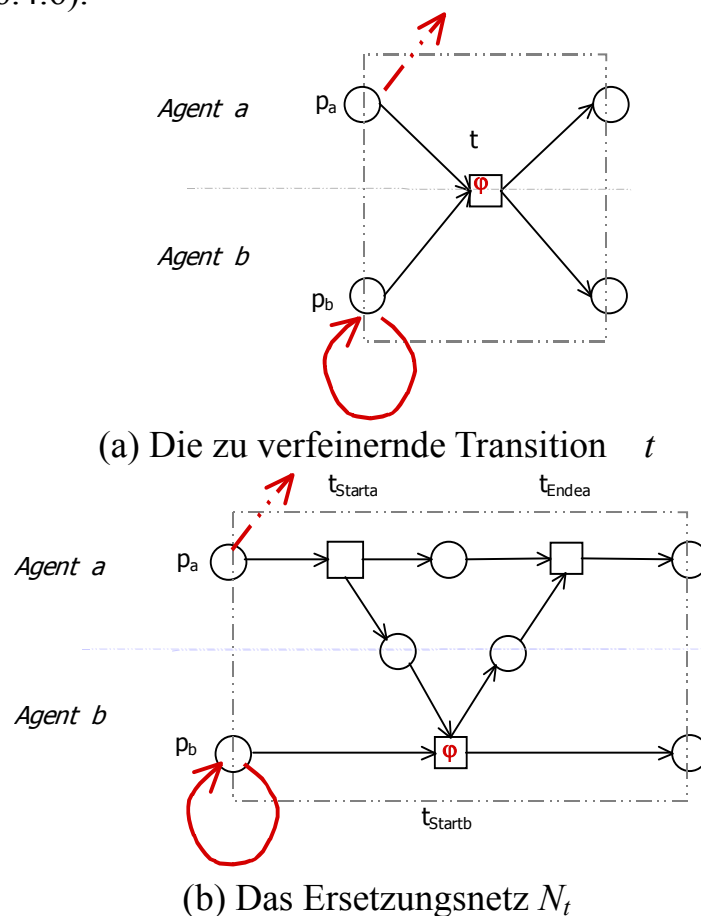


Abb. 6.4.6 Illustration für Satz 6.4.2 (Abschwächung von Satz 6.4.1) (die roten Pfeile symbolisieren konkurrente Transitionen zu t . Die Schleife symbolisiert, dass falls p_a belegt ist, p_b immer wieder markiert wird.)

Satz 6.4.2 (Abschwächung von Satz 6.4.1)

Gegeben sei das Agentensystem Σ mit Agentenmenge A und Kanalmenge C und die unverteilte Transition t . Sei $(\bullet t)_a := \bullet t \cap (a \cup C_a)$, $(\bullet t)_b := \bullet t \cap (b \cup C_b)$ und $\bullet t = (\bullet t)_a \cup (\bullet t)_b$, wobei $a, b \in A$.

In Σ gelte:

- i. $(\bullet t)_a \triangleright (\bullet t)_b$;
- ii. Es gibt keine aktivierte konkurrenten Transitionen zu $(\bullet t)_a$;
- iii. t ist fair.

Dann garantiert jedes Verfeinerungsnetz N_t mit folgenden Eigenschaften die Blockbedingung für $\Sigma' = \Sigma(N_t \setminus t)$:

In N_t gibt es Transitionen t_{Start_a} und t_{Start_b} mit:

- iv. $\bullet t_{Start_a} = (\bullet t)_a$, $\bullet t_{Start_b} \supset (\bullet t)_b$ und $\bullet t_{Start_b} \subseteq (\bullet t)_b \cup \bullet t_{Start_a}$;
- v. Es gibt keine konkurrenten Transitionen zu t_{Start_a} und t_{Start_b} in N_t ;
- vi. $t_{Start_a} \blacktriangleleft t_{Start_b}$ gilt in (N_b, t^-) ;
- vii. $\bullet t \blacktriangleleft t^\bullet$ gilt in (N_b, t^-) ;
- viii. t_{Start_b} ist fair.

Beweis:

Zu zeigen ist: Jeder Ablauf von Σ' wird aus einem Ablauf von Σ erhalten, in dem jedes Auftreten von t durch einen vollständigen Ablauf von (N_b, t^-) ersetzt wird.

Wir nehmen an, es gibt einen Ablauf (K', r') von Σ' , für den das nicht der Fall ist. Diese Annahme ist zum Widerspruch zu führen.

Aus der Annahme folgt: (K', r') enthält einen unvollständigen Ablauf von (N_b, t^-) . Es gibt einen Schnitt C , in dem t_{Start_a} schaltet (da t_{Start_a} wegen der Voraussetzung $t_{Start_a} \blacktriangleleft t_{Start_b}$ die erste aktivierte Transition in N_t ist), also $C' \subseteq C$ mit $r'(C') = \bullet t_{Start_a}$. Wegen der Folgerung von Bem. 6.2.1 folgt, auf diesen Schnitt folgt kein Schnitt, der C' enthält und in dem auch t_{Start_b} schaltet.

Nach Voraussetzung i gilt $(\bullet t)_a \triangleright (\bullet t)_b$ in Σ . Wegen den

Voraussetzungen *vii* und *ii* (also $\bullet t \triangleleft t \bullet$ und es gibt keine aktivierte konkurrente Transition zu $(\bullet t)_a$) gilt weiter $(\bullet t)_a \triangleright (\bullet t)_b$ in K' . Daraus folgt: Auf C folgt ein Schnitt D , in dem $(\bullet t)_a$ noch gilt, und auch $(\bullet t)_b$ gilt, d.h. $C' \subseteq D$ und $C'' \subseteq D$ mit $r'(C'') = (\bullet t)_b$.

Angenommen, E ist ein Schnitt mit $D \xrightarrow{e_{Start_a}} E$, wobei $r'(e_{Start_a}) = t_{Start_a}$ (d.h. E ist der Schnitt nach dem Auftreten von t_{Start_a} in D).

Also gilt: $E = (D \setminus \bullet e_{Start_a}) \cup e_{Start_a} \bullet$.

Dann gilt $C'' \subseteq E$ (wegen $(\bullet t)_b \cap (\bullet t_{Start_a} \cup t_{Start_a} \bullet) = \emptyset$).

Im Zustand $r'(E)$ ist die Transition t_{Start_b} aktiviert (wegen $\bullet t_{Start_b} \subseteq (\bullet t)_b \cup t_{Start_a} \bullet$).

Angenommen, eine konkurrente Transition t_{Start_b}' von t_{Start_b} schaltet,

und C_1 ist ein Schnitt mit $D \xrightarrow{e_{Start_b}' } C_1$, wobei $r'(e_{Start_b}') = t_{Start_b}'$ (d.h. C_1 ist der Schnitt nach dem Auftreten von t_{Start_b}' in D).

Dann gilt $C' \subseteq C_1$ (weil es keine konkurrente Transition zu $(\bullet t)_a$ gibt).

Weil $(\bullet t)_a \triangleright (\bullet t)_b$ gilt und es keine aktivierte konkurrente Transition zu $(\bullet t)_a$ gibt, gilt: Auf C_1 folgt ein Schnitt D_1 , in dem $(\bullet t)_a$ noch gilt, und auch $(\bullet t)_b$ gilt, d.h. $C' \subseteq D_1$ und $C_1'' \subseteq D_1$ mit $r'(C_1'') = (\bullet t)_b$.

Angenommen, E_1 ist ein Schnitt mit $D_1 \xrightarrow{e_{Start_a}} E_1$, wobei $r'(e_{Start_a}) = t_{Start_a}$.

Dann gilt $C_1'' \subseteq E_1$. Im Zustand $r'(E_1)$ ist die Transition t_{Start_b} wieder aktiviert.

Also, die Transition t_{Start_b} wird immer wieder aktiviert. Nach der *Fairness*-Annahme wird sie irgendwann schalten. Also, es gibt einen Schnitt D_n , der C' enthält und in dem t_{Start_b} schaltet. Das ist ein Widerspruch dazu, dass auf C kein Schnitt folgt, der C' enthält und in dem auch t_{Start_b} schaltet.

Daraus folgt die Behauptung. □

6.5 Vertauschbarkeit von Verfeinerungsschritten

Gibt es in einem unverteilter System zwei zu verteilende Transitionen, so stellt sich die Frage, ob beide gleichzeitig verteilt werden dürfen. Wenn das der Fall ist, dann darf für beide Transitionen die Gültigkeit der Blockbedingung im unverteilter System nachgewiesen werden.

Definition 6.5.1 (*Halbordnung vollständig erhaltend*)

Ein Verfeinerungsschritt $(\Sigma \rightarrow \Sigma')$ mit $\Sigma' = \Sigma(N_t \setminus t)$ heißt Halbordnung vollständig erhaltend, falls Σ' die Blockbedingung erfüllt und

$(N_b, t^-) \models \bullet t \blacktriangleleft t \bullet$ gilt.

Eine Folge von Verfeinerungsschritten $\Sigma_0 \rightarrow \Sigma_1 \rightarrow \dots \rightarrow \Sigma_n$ heißt Halbordnung vollständig erhaltend, falls jeder Schritt $(\Sigma_{k-1} \rightarrow \Sigma_k)$, $k=1, \dots, n$, Halbordnung vollständig erhaltend ist.

Satz 6.5.2 (*Gleichzeitige Verfeinerung nicht konkurrierender Transitionen*)

Sei Σ ein System, seien $t_1, t_2 \in T$ mit $\bullet t_1 \cap \bullet t_2 = \emptyset$.

Für $k=1, 2$ gelte:

$(\Sigma \rightarrow \Sigma')$ mit $\Sigma' = \Sigma(N_{t_k} \setminus t_k)$ ist Halbordnung vollständig erhaltend mit:

$(\Sigma \rightarrow \Sigma')$ erfüllt die Bedingungen von einem der Sätze 6.2.5, 6.3.2, 6.4.1, 6.4.2.

Dann ist die Folge der Verfeinerungsschritte $\Sigma_0 \rightarrow \Sigma_1 \rightarrow \Sigma_2$ Halbordnung vollständig erhaltend, wobei $\Sigma_0 = \Sigma$, $\Sigma_k = \Sigma_{k-1}(N_{t_k} \setminus t_k)$, $k=1, 2$.

Beweis:

Angenommen, Verfeinerungsschritt $(\Sigma_0 \rightarrow \Sigma_2')$ mit $\Sigma_2' = \Sigma_0(N_{t_2} \setminus t_2)$ erfüllt die Bedingung von Satz 6.3.2 (Beweis für die anderen Fälle ist ähnlich¹³), d.h. es gilt in Σ_0 :

i. $(\bullet t_2)_a \triangleright (\bullet t_2)_b$ (d.h. t_2 ist eine unverteilte Transition von Agenten a und b).

ii. Jede Konflikt-Transition u_2 von t_2 an Stellen $(\bullet t_2)_b$ und $(\bullet t_2)_a$ ist nicht aktiviert

(d.h. $(\bullet t_2)_b \rightarrow \neg p$ für ein $p \in \bullet u_2 \setminus (\bullet t_2)_b$).

¹³ Für die anderen Fälle sind die Forderungen an die Umgebung ähnlich wie bei Satz 6.3.2 die folgenden: i. *Leadsto*-Eigenschaften der Form $R \triangleright Q$; ii. keine aktivierte konkurrente Transition.

Wir zeigen: Beim Verfeinerungsschritt $(\Sigma_0 \rightarrow \Sigma_1)$ mit $\Sigma_1 = \Sigma_0(N_{t_1} \setminus t_1)$ bleiben (i), (ii) erhalten und damit die Voraussetzungen für den Nachweis der Blockbedingung mit Hilfe des Satzes 6.3.2.

zu i.: Weil $(\bullet t_2)_b \cap \bullet t_1 = \emptyset$ gilt, (wegen $\bullet t_1 \cap \bullet t_2 = \emptyset$)
 und $(\Sigma_0 \rightarrow \Sigma_1)$ Halbordnung vollständig erhaltend ist,
 bleibt (i) erhalten. (wegen Lemma 3.5.11)
 Zu ii.: bleibt erhalten. (wegen Bemerkung 3.5.3)
 Daraus folgt: (i), (ii) gelten in Σ_1 .

Es gilt $(N_{t_2}, t_2^-) \models \bullet t_2 \blacktriangleleft t_2^\bullet$. (wegen: $(\Sigma_0 \rightarrow \Sigma_2')$ ist
 Halbordnung vollständig erhaltend)
 Der Verfeinerungsschritt $(\Sigma_1 \rightarrow \Sigma_2)$ mit $\Sigma_2 = \Sigma_1(N_{t_2} \setminus t_2)$ erfüllt die
 Blockbedingung. (wegen Satz 6.3.2)

Daraus folgt: $(\Sigma_1 \rightarrow \Sigma_2)$ ist Halbordnung vollständig erhaltend.
 Daraus folgt: $\Sigma_0 \rightarrow \Sigma_1 \rightarrow \Sigma_2$ ist Halbordnung vollständig erhaltend. \square

Folgerung 6.5.3

Sei Σ ein System, seien $t_1, t_2, \dots, t_n \in T$ mit $\bullet t_1 \cap \bullet t_2 \cap \dots \cap \bullet t_n = \emptyset$.

Für $k=1, 2, \dots, n$ gelte:

$(\Sigma \rightarrow \Sigma')$ mit $\Sigma' = \Sigma(N_{t_k} \setminus t_k)$ ist Halbordnung vollständig erhaltend
 mit:

$(\Sigma \rightarrow \Sigma')$ erfüllt die Bedingungen von einem der Sätze 6.2.5,
 6.3.2, 6.4.1, 6.4.2.

Dann ist die Folge der Verfeinerungsschritte $\Sigma_0 \rightarrow \Sigma_1 \rightarrow \dots \rightarrow \Sigma_n$
 Halbordnung vollständig erhaltend, wobei $\Sigma_0 = \Sigma$, $\Sigma_k = \Sigma_{k-1}(N_{t_k} \setminus t_k)$,
 $k=1, 2, \dots, n$.

Beweis: durch Induktion über die Anzahl n der zu verfeinernden
 Transitionen t_1, t_2, \dots, t_n .

Für $n=1$: Verfeinerungsschritt $(\Sigma_0 \rightarrow \Sigma_1)$ mit $\Sigma_1 = \Sigma_0(N_{t_1} \setminus t_1)$ ist
 Halbordnung vollständig erhaltend.

Wir nehmen an, für $n=m$ gilt die Aussage.
 Zu zeigen ist: Für $n=m+1$ gilt die Aussage.

Angenommen, Verfeinerungsschritt $(\Sigma_0 \rightarrow \Sigma_{m+1,0})$ mit $\Sigma_{m+1,0} = \Sigma_0(N_{t_{m+1}} \setminus t_{m+1})$

t_{m+1}) erfüllt die Bedingung von Satz 6.3.2. d.h. in Σ_0 gelten:

$$(\bullet t_{m+1})_a \triangleright (\bullet t_{m+1})_b;$$

Jede Konflikt-Transition u an Stellen $(\bullet t_{m+1})_a$ und $(\bullet t_{m+1})_b$ ist nicht aktiviert.

Die Folge von Verfeinerungsschritten $\Sigma_0 \rightarrow \Sigma_1 \rightarrow \dots \rightarrow \Sigma_m$ mit $\Sigma_k = \Sigma_{k-1} (N_{t_k} \setminus t_k)$, $k=1,2,\dots,m$ ist Halbordnung vollständig erhaltend. (wegen Induktionsannahme)

Bei jedem Verfeinerungsschritt $(\Sigma_{k-1} \rightarrow \Sigma_k)$, $k=1,2,\dots,m$, gilt:

(i) bleibt erhalten, d.h. $(\bullet t_{m+1})_a \triangleright (\bullet t_{m+1})_b$ bleibt beim Verfeinerungsschritt

$(\Sigma_{k-1} \rightarrow \Sigma_k)$ erhalten, weil: (wegen Lemma 3.5.11)

- $(\Sigma_{k-1} \rightarrow \Sigma_k)$ ist Halbordnung vollständig erhaltend und

$$- \bullet t_{m+1} \cap \bullet t_k = \emptyset.$$

(ii) bleibt erhalten. (wegen Bemerkung 3.5.3)

Daraus folgt: (i), (ii) gelten weiter in Σ_m .

Der Verfeinerungsschritt $(\Sigma_m \rightarrow \Sigma_{m+1})$ mit $\Sigma_{m+1} = \Sigma_m (N_{t_{m+1}} \setminus t_{m+1})$ erfüllt die Blockbedingung. (wegen Satz 6.3.2)

Daraus folgt: $(\Sigma_m \rightarrow \Sigma_{m+1})$ ist Halbordnung vollständig erhaltend.

Daraus folgt: $\Sigma_0 \rightarrow \Sigma_1 \rightarrow \dots \rightarrow \Sigma_m \rightarrow \Sigma_{m+1}$ ist Halbordnung vollständig erhaltend. □

Folgerung 6.5.4 (Gleichzeitige Verfeinerung konkurrierender Transitionen)

Sei Σ_0 ein System, seien $t_1, t_2, t \in T$ mit $\bullet t_1 \cap \bullet t_2 = \emptyset$.

Für $k=1,2$ gelte:

$(\Sigma_0 \rightarrow \Sigma_k)$ mit $\Sigma_k = \Sigma_0 (N_{t_k} \setminus t_k)$ ist Halbordnung vollständig erhaltend mit:

$(\Sigma_0 \rightarrow \Sigma_k)$ erfüllt die Bedingungen ϕ_k von einem der Sätze 6.2.5, 6.3.2, 6.4.1, 6.4.2.

Weiter sei $(\Sigma_0 \rightarrow \Sigma_0')$ mit $\Sigma_0' = \Sigma_0 (N_t \setminus t)$ Halbordnung vollständig erhaltend mit:

Bei $(\Sigma_0 \rightarrow \Sigma_0')$ bleiben die oben genannten Bedingungen ϕ_k , $k=1,2$ erfüllt.

Dann ist die Folge der Verfeinerungsschritte

$\Sigma_0 \rightarrow \Sigma_0' \rightarrow \Sigma_1' \rightarrow \Sigma_2'$ mit $\Sigma_k' = \Sigma_{k-1}' (N_{t_k} \setminus t_k)$, $k=1,2$, Halbordnung vollständig erhaltend.

Beweis:

Angenommen, Verfeinerungsschritt $(\Sigma_0 \rightarrow \Sigma_1)$ mit $\Sigma_1 = \Sigma_0(N_{t_1} \setminus t_1)$ erfüllt die Bedingung an die Umgebung ϕ_1 von Satz 6.3.2, d.h. $\Sigma_0 \models \phi_1$. Und beim Verfeinerungsschritt $(\Sigma_0 \rightarrow \Sigma_0')$ mit $\Sigma_0' = \Sigma_0(N_t \setminus t)$ bleibt ϕ_1 erhalten.

Dann gilt $\Sigma_0' \models \phi_1$.

Dann gilt: Verfeinerungsschritt $(\Sigma_0' \rightarrow \Sigma_1')$ mit $\Sigma_1' = \Sigma_0'(N_{t_1} \setminus t_1)$

- erfüllt die Blockbedingung und (wegen Satz 6.3.2)

- ist Halbordnung vollständig erhaltend. (wegen $(N_{t_1}, t_1^-) \models \bullet t_1 \blacktriangleleft$

$t_1 \bullet$)

Analog erfüllt der Verfeinerungsschritt $(\Sigma_0' \rightarrow \Sigma_{20}')$ mit $\Sigma_{20}' = \Sigma_0'(N_{t_2} \setminus t_2)$ die Blockbedingung und ist Halbordnung vollständig erhaltend.

Wegen $\bullet t_1 \cap \bullet t_2 = \emptyset$ gilt:

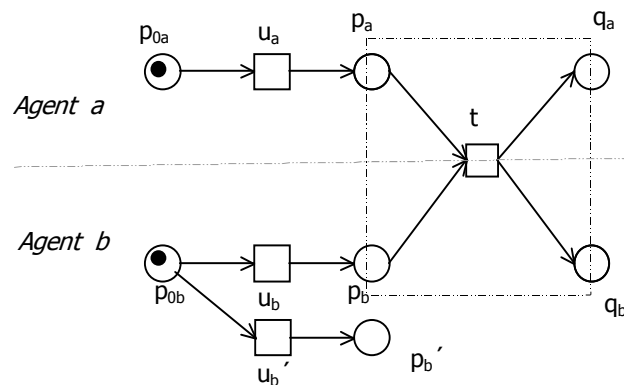
Die Folge von Verfeinerungsschritten $\Sigma_0' \rightarrow \Sigma_1' \rightarrow \Sigma_2'$ ist Halbordnung vollständig erhaltend. (wegen Satz 6.5.2)

Daraus folgt: $\Sigma_0 \rightarrow \Sigma_0' \rightarrow \Sigma_1' \rightarrow \Sigma_2'$ ist Halbordnung vollständig erhaltend. □

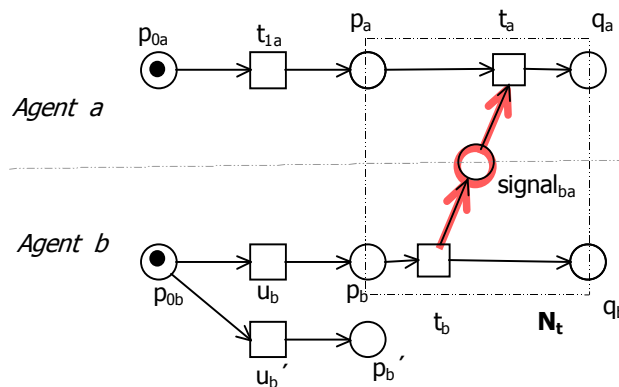
Den in diesem Abschnitt angegebenen Satz und die Folgerungen werden wir im Kapitel 7 verwenden.

6.6 Umgebung, für die es keine korrekte verteilende Verfeinerung gibt

In Abb. 6.6.1 (a) ist ein System Σ_1 dargestellt. In Σ_1 hat Agent b eine Konflikt-Transition u_b' an der Stelle p_{0b} . Wenn b diese Konflikt-Transition ausführt, dann kann es an der Stelle p_b kein Token geben, und die Transition t kann nicht schalten. Um die Blockbedingung zu garantieren, muss eine verteilende Verfeinerung von t die Kausalität $(p_b \bullet \blacktriangleleft p_a \bullet)$ erfüllen (siehe Abb. 6.6.1(b)).



(a) Σ_1 (vor der Verfeinerung von t)



(b) Σ_1' (nach der Verfeinerung von t)

Abb. 6.6.1 System Σ_1 und Verfeinerungsschritt $(\Sigma_1 \rightarrow \Sigma_1')$

Wir ergänzen jetzt Σ_1 um eine zu u_a Konflikt-Transition u_a' und erhalten das System Σ_2 (Abb. 6.6.2). Eine verteilende Verfeinerung von t des Systems Σ_2 muss jetzt aus Symmetriegründen auch die Bedingung

$$(p_a \bullet \blacktriangleleft p_b \bullet)$$

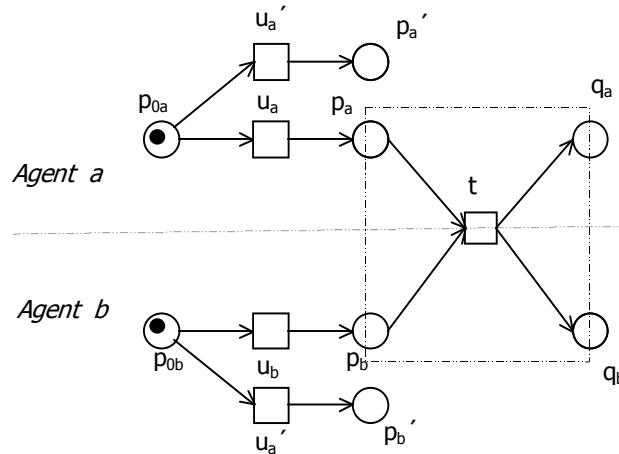


Abb. 6.6.2 System Σ_2 (vor der Verfeinerung von t)

Angenommen, $p_a^\bullet = \{t_a\}$, $p_b^\bullet = \{t_b\}$. Aus der Definition von $t_a \blacktriangleleft t_b$ folgt: In jedem Ablauf tritt vor t_b ein t_a auf und aus der Definition von $t_b \blacktriangleleft t_a$ folgt, vor t_a tritt t_b auf. Das ist ein Widerspruch zur Zyklensfreiheit der Kausalordnung. Damit gilt der Fall $p_a^\bullet = p_b^\bullet$. Also nur eine nicht-verteilende Verfeinerung kann t verfeinern, so dass die Blockbedingung erhalten bleibt, z.B. die in Abb. 6.6.3 dargestellte Verfeinerung.

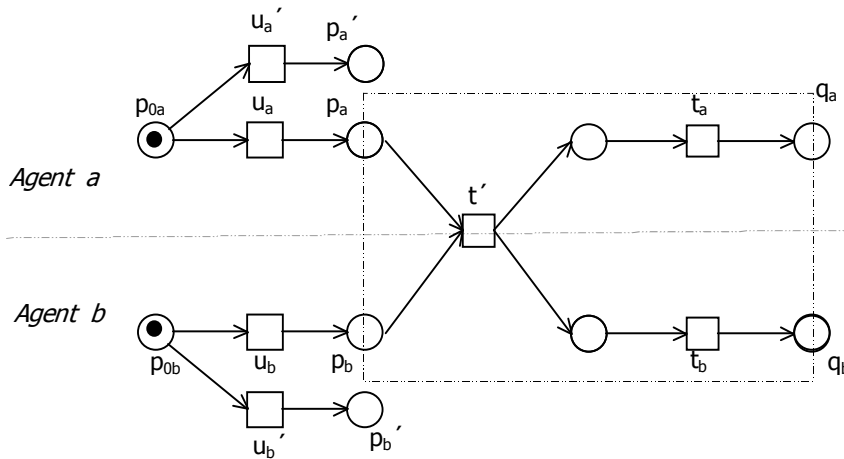


Abb. 6.6.3 System Σ_2' (nach der Verfeinerung von t)

Damit erhalten wir den folgenden Satz.

Satz 6.6.1

Gegeben sei das Agentensystem Σ mit Agentenmenge A und Kanalmenge C und die unverteilte Transition t . Sei $(\bullet t)_a := \bullet t \cap (a \cup C_a)$,

$(\bullet t)_b := \bullet t \cap (b \cup C_b)$ und $\bullet t = (\bullet t)_a \cup (\bullet t)_b$, wobei $a, b \in A$.

Gilt - weder $(\bullet t)_a \triangleright (\bullet t)_b$ und zu $(\bullet t)_b$ gibt es keine aktivierte Konflikt-Transition

- noch $(\bullet t)_b \triangleright (\bullet t)_a$ und zu $(\bullet t)_a$ gibt es keine aktivierte Konflikt-Transition,

so gibt es kein Verfeinerungsnetz N_t , das die Blockbedingung für $\Sigma' = \Sigma(N_t \setminus t)$ garantiert.

7. Entwurf und Verifikation verteilter Algorithmen durch verteilende Verfeinerung und Anwendungsbeispiele

In diesem Kapitel werden wir unsere Methode zum Entwurf und zur Verifikation durch verteilende Verfeinerung vorstellen. Wir werden anhand einiger nicht trivialer dennoch verständlicher und anschaulicher Anwendungsbeispiele unsere Methode demonstrieren.

Als Anwendungsbeispiele für die Verifikation werden wir zwei verteilte *Mutex*-Algorithmen mit unterschiedlichen algorithmischen Ideen – *Token-passing Mutex* und *asymmetrischer verteilter Mutex* – verifizieren. Als Anwendungsbeispiel für den Entwurf werden wir den verteilten *crosstalk* Algorithmus durch verteilende Verfeinerung entwerfen.

Bei *Mutex* betreffen die gewünschten Eigenschaften die Stellen von Agenten. Daher ist es unwichtig, was die zu verteilenden Transitionen für Teilaufgaben modellieren. Die Eigenschaften des *crosstalk* beziehen sich dagegen auf die Teilaufgaben der zu verteilenden Transitionen. Daher ist es wichtig, was diese Transitionen für Teilaufgaben modellieren. Das ist der schwerere Fall. Die gewünschten Eigenschaften werden durch Reihenfolge-Relationen der Aktionen (Kausalitäten im System) formuliert.

7.1 Verifikation verteilter Algorithmen durch verteilende Verfeinerung

7.1.1 Verifikation durch verteilende Verfeinerung

Wie bereits beschrieben, begannen die Untersuchungen zu dieser Arbeit mit dem Versuch, verteilte Algorithmen zu beweisen, indem wir mehrere verteilte Aktionen zu einer unverteiltern Aktion zusammenfassten. Dadurch entstanden einfachere Algorithmen, die sich leichter beweisen ließen. Es ging also um Beweisen durch eine schrittweise korrekte Vergrößerung. Im weiteren Verlauf haben wir als Umkehrung dieser Vergrößerungsschritte den in dieser Arbeit vorgestellten Verfeinerungsbegriff gefunden. Es hat sich herausgestellt, dass man bei der Verifikation ähnlich wie beim Entwerfen eines Algorithmus vorgehen kann, was wir an zwei verteilten *Mutex* Algorithmen (aus [Rei98]) in diesem Abschnitt zeigen wollen.

Als erstes ist die algorithmische Idee des gegebenen Algorithmus zu bestimmen. Danach sucht man nach einem Modell, das die Grundidee des Algorithmus darstellt. Dieses Modell wird zu einem unverteiltern Agentensystem erweitert, das jetzt schon die algorithmische Idee veranschaulicht. Von diesem Netz ist zu beweisen, dass es die geforderten Eigenschaften des gegebenen Algorithmus erfüllt. Jetzt sind nur noch die im Netz enthaltenen unverteiltern Transitionen korrekt zu verteilen. Es wird ein verteilter korrekter Algorithmus erhalten, der mit dem gegebenen Algorithmus übereinstimmt. In einigen Fällen wurden im gegebenen zu beweisenden Algorithmus einige Optimierungen (Zusammenfassen von Stellen und Transitionen mit gleichen Funktionen) vorgenommen, die den Algorithmus einfacher erscheinen lassen, aber meist schwerer zu verstehen machen. Solche Optimierungen müssen nicht unbedingt nachvollzogen werden, da bei Bedarf entsprechende Optimierungen automatisch durchgeführt werden können.

Diese Verifikationsmethode hat noch den Vorteil, dass neben dem Beweis des gegebenen Algorithmus auch gut nachvollziehbare Entwurfsschritte für den Algorithmus erhalten werden.

7.1.2 Anwendungsbeispiel – *Token-Passing-Mutex* Algorithmus

1. Grundmodell und Anfangsalgorithmus für verteilende Verfeinerung

Grundmodell

Im Abschnitt 4.1 haben wir bereits die algorithmische Idee des verteilten *Mutex* als Beispiel betrachtet. Wir wollen hier die wichtigsten Ergebnisse wiederholen. Die Grundidee eines *Mutex*-Algorithmus veranschaulicht das Netz Σ_a (siehe Abb. 7.1.1).

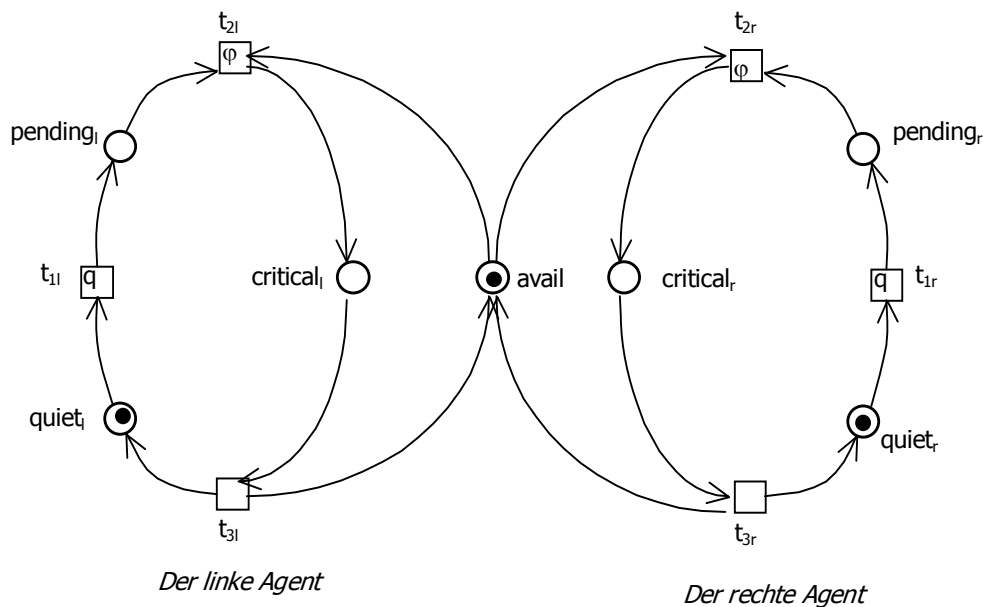


Abb. 7.1.1 Grundmodell Σ_a

Zwei Agenten a und b benutzen eine gemeinsame Ressource. Jeder Agent hat die gleichen Chancen, die Ressource bei Bedarf zu bekommen.

Gewünschte Eigenschaften

Sicherheitseigenschaft: $\square \neg(\text{critical}_l \wedge \text{critical}_r)$ (S)

Lebendigkeitseigenschaften: $\text{pending}_l \triangleright \text{critical}_l$ (L1)

$\text{pending}_r \triangleright \text{critical}_r$ (L2)

Anfangsalgorithmus

Die Analyse des verteilten *Mutex*-Algorithmus zeigt, dass er folgende algorithmische Idee verwirklicht. Die Agenten sind abwechselnd Besitzer der Ressource. Wenn ein Agent vom anderen Agenten die Ressource haben will, dann sendet er eine Nachricht und erhält dann irgendwann die Ressource. Das Netz in Abb. 7.1.2 zeigt diesen Algorithmus Σ_0 , der ein Agentensystem ist.

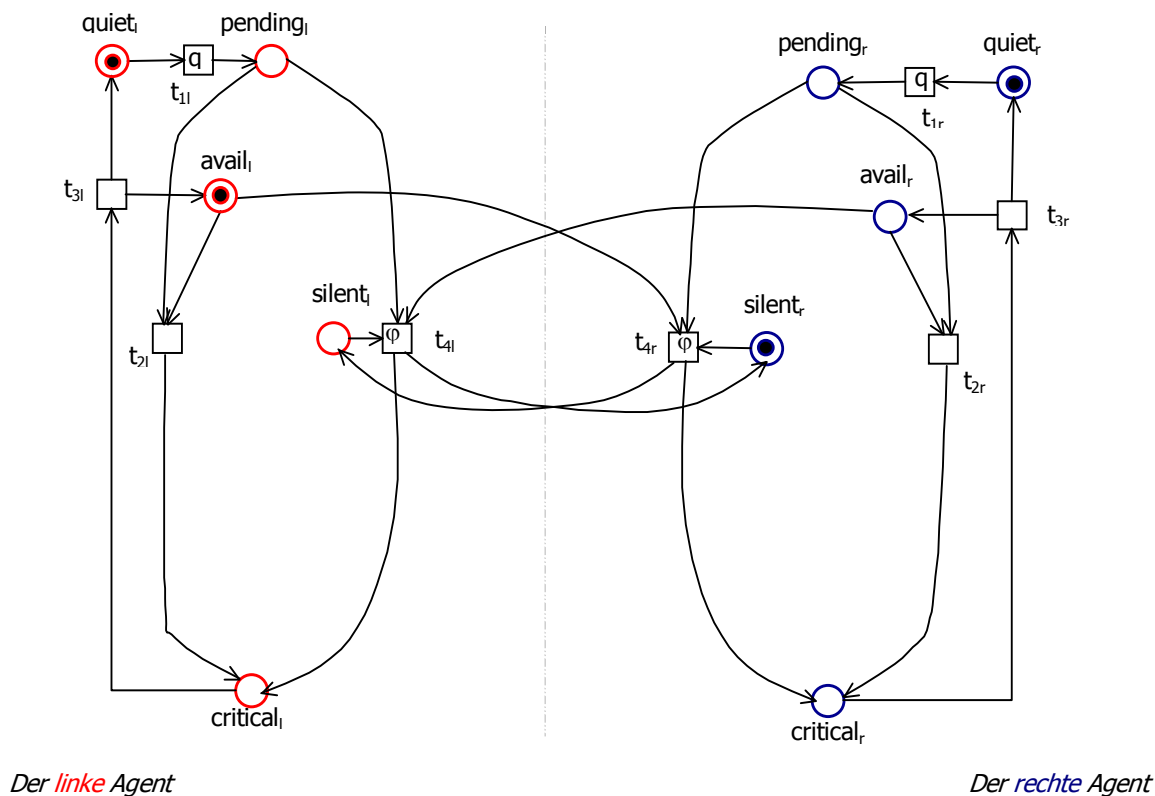


Abb. 7.1.2 Anfangsalgorithmus Σ_0

Wie schon erwähnt, in Σ_0 sind die Transitionen t_{4r} und t_{4l} unverteilt. Wir werden in zwei Schritten jeweils diese unverteilt Transitionen verteilen. Schließlich erhalten wir den verteilten *Mutex*-Algorithmus Σ^* .

2. Die verteilende Verfeinerung

Die Verteilung der unverteilt Transition t_{4r}

Im ersten Schritt verteilen wir die unverteilt Transition t_{4r} (Abb. 7.1.3) zu einem verteilten Ersetzungsnetz $N_{t_{4r}}$ (Abb. 7.1.4).

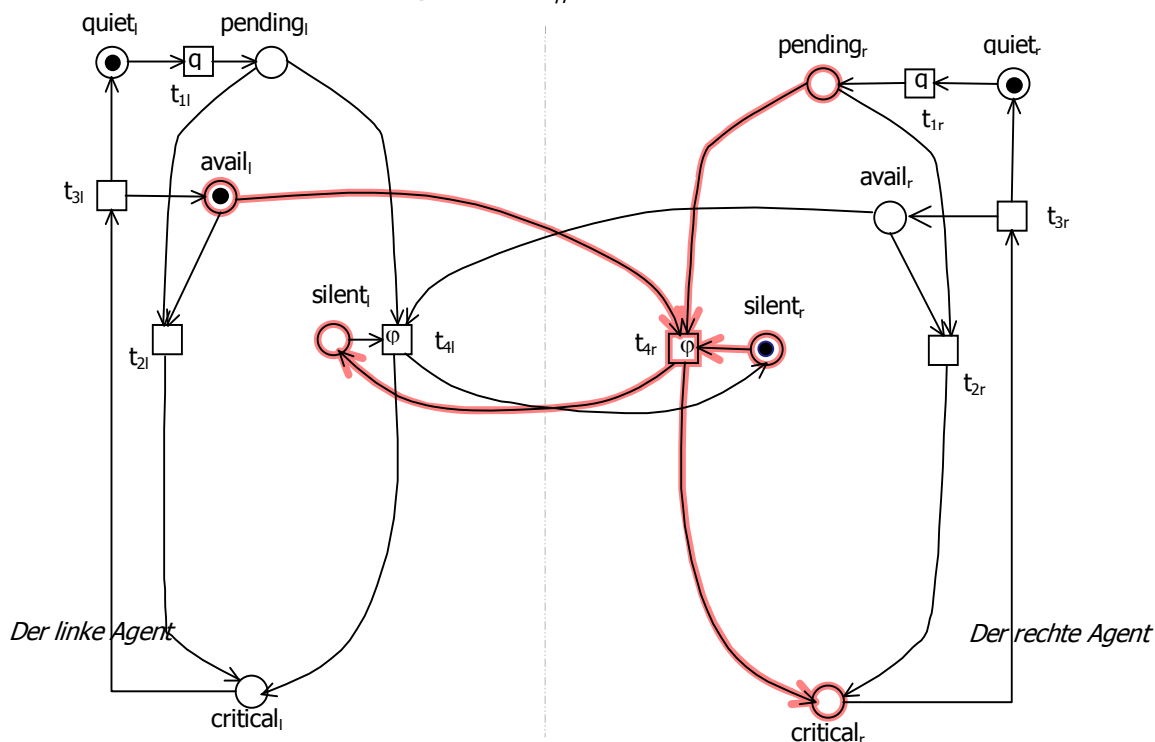
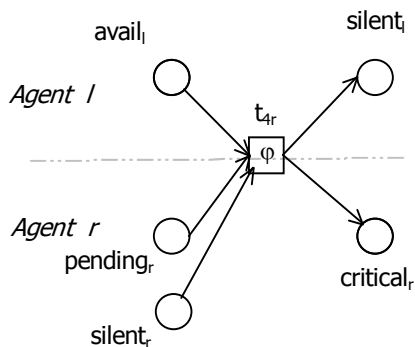
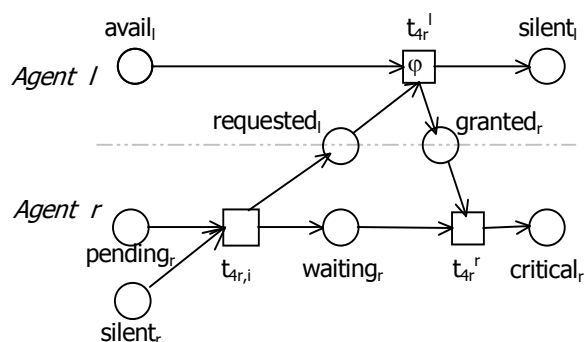


Abb. 7.1.3 System Σ_0 (vor der Verteilung von t_{4r} , die Kanten von und nach t_{4r} sind rot hervorgehoben.)



(a) Transition t_{4r}



(b) Ersetzungsnetz $N_{t_{4r}}$

Abb. 7.1.4 Die unverteilt Transition t_{4r} und ihr verteiltes Ersetzungsnetz $N_{t_{4r}}$

Kausalität

Im lokalen Teilsystem $(N_{t_{4r}}, t_{4r}^-)$ gilt offensichtlich die folgende Kausalität:

$$\bullet t_{4r} \blacktriangleleft t_{4r} \bullet$$

Blockbedingung

Die Umgebung von t_{4r} erfüllt die folgenden Bedingungen:

- In Σ_0 gilt: $pending_r \wedge silent_r \triangleright avail_l$.
- t_{4r} hat keine aktivierte Konflikt-Transition

Das Ersetzungsnetz $N_{t_{4r}}$ für t_{4r} erfüllt die folgenden Bedingungen:

- In $(N_{t_{4r}}, t_{4r}^-)$ gelten:

Anfangstransition $t_{4r,i}$ von $r \blacktriangleleft$ Anfangstransition t_{4r}^l von l
 und $\bullet t_{4r} \blacktriangleleft t_{4r} \bullet$.

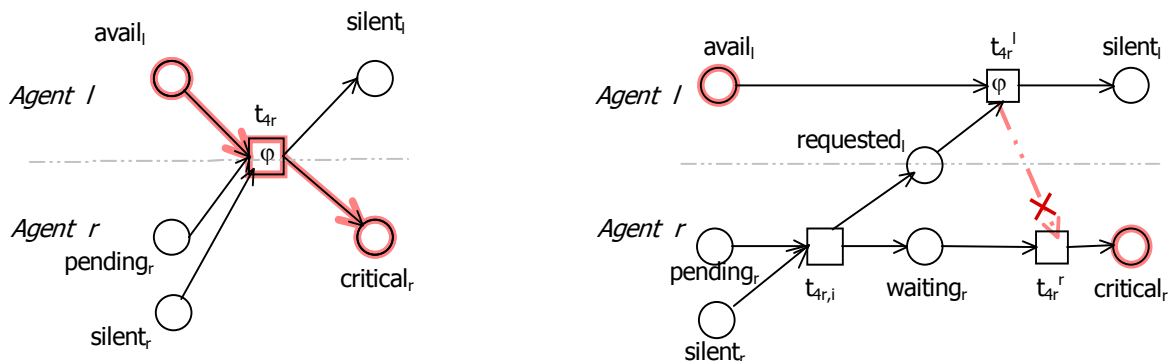
- Anfangstransition t_{4r}^l von l ist *fair*.

Wir benutzen den dritten Grundtyp (vgl. Satz 6.4.2 in Abschnitt 6.4) zum Nachweisen, dass die Blockbedingung erfüllt ist.

Erhaltenbleiben von Eigenschaften

Wir benutzen die Sätze über Erhaltenbleiben der Eigenschaften durch Verfeinerung mit Kausalitäten aus Kapitel 3 (vgl. Lemma 3.5.12, Lemma 3.5.9 in Abschnitt 3.5) zum Nachweisen, dass alle gewünschten Eigenschaften erhalten bleiben.

Ohne Kausalität $t_{4r}^l \blacktriangleleft t_{4r}^r$ im Teilsystem $(N_{t_{4r}}, t_{4r}^-)$ (Abb. 7.1.5(b)) kann im neuen System nach der Verteilung folgendes passieren: An der Stelle $avail_l$ und an der Stelle $critical_r$ können gleichzeitig Token liegen, d.h. die Eigenschaft (S) wird beim Verteilen nicht erhalten bleiben.

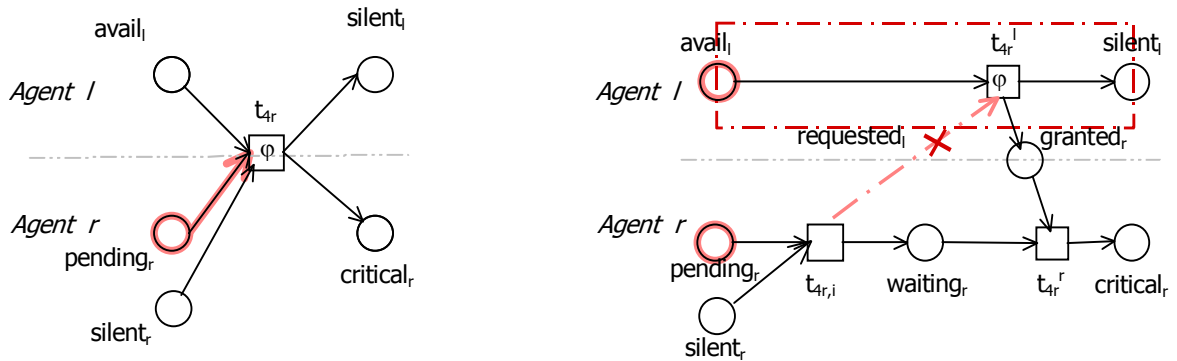


(a) Transition t_{4l}

(b) Ein Ersetzungsnetz ohne $t_{4r}^l \blacktriangleleft t_{4r}^r$

Abb. 7.1.5 *Mutex*-Eigenschaft verletzt ohne $t_{4r}^l \blacktriangleleft t_{4r}^r$

Ohne Kausalität $t_{4r,i} \blacktriangleleft t_{4r}^l$ im Teilsystem $(N_{t_{4r}}, t_{4r}^-)$ (Abb. 7.1.6(b)) kann nach der Verteilung folgendes passieren: Agent r braucht keine Ressource (also $pending_r$ gilt nicht), aber Agent l führt die Transition t_{4r}^l im Ersetzungsnetz $N_{t_{4r}}$ alleine seinerseits aus. D.h. die Blockbedingung wird nicht erfüllt.



(a) Transition t_{4l}

(b) Ein Ersetzungsnetz ohne $t_{4r,i} \blacktriangleleft t_{4r}^l$

Abb. 7.1.6 Blockbedingung unerfüllt ohne $t_{4r,i} \blacktriangleleft t_{4r}^l$

In Abb. 7.1.7 ist das erhaltene System Σ_1 nach der Verteilung von t_{4r} angegeben.

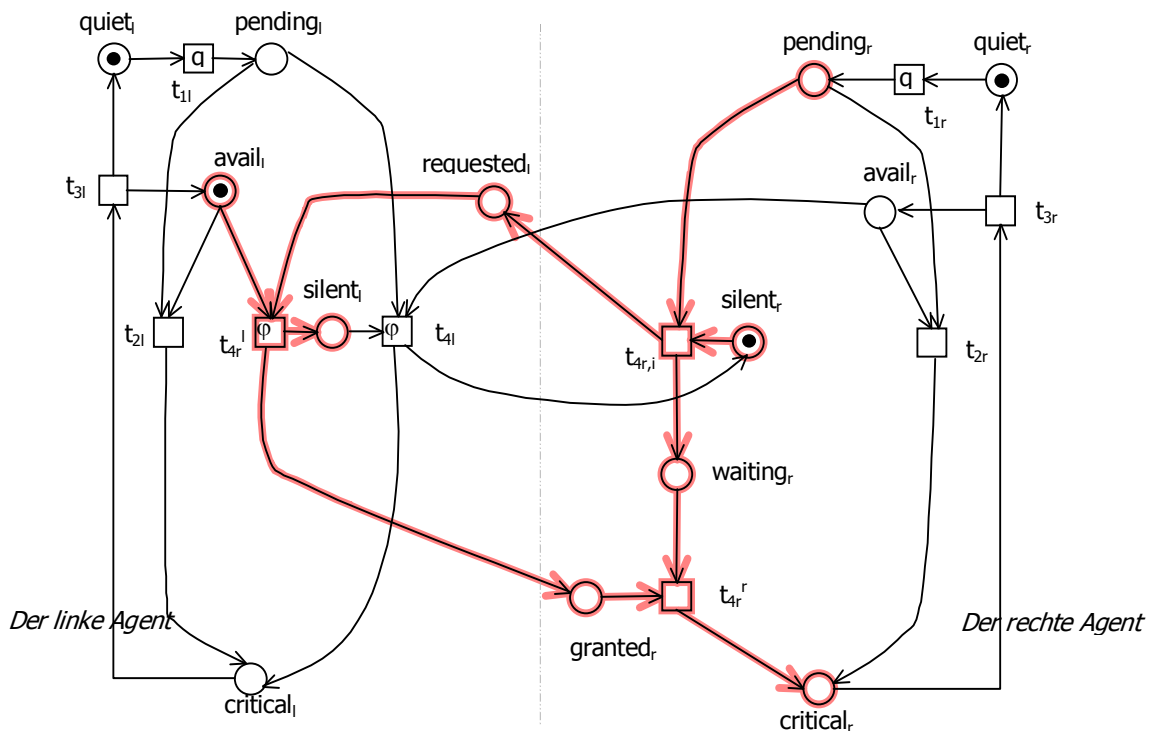


Abb. 7.1.7 System Σ_1 (nach der Verteilung von t_{4r})

Die Verteilung der unverteilter Transition t_{4l}

Analog zu der Verteilung von t_{4r} können wir auch die Transition t_{4l} verteilen.

Wir verteilen die unverteilter Transition t_{4l} (Abb. 7.1.8) zu einem verteilten Ersetzungsnetz $N_{t_{4l}}$ (Abb. 7.1.9).

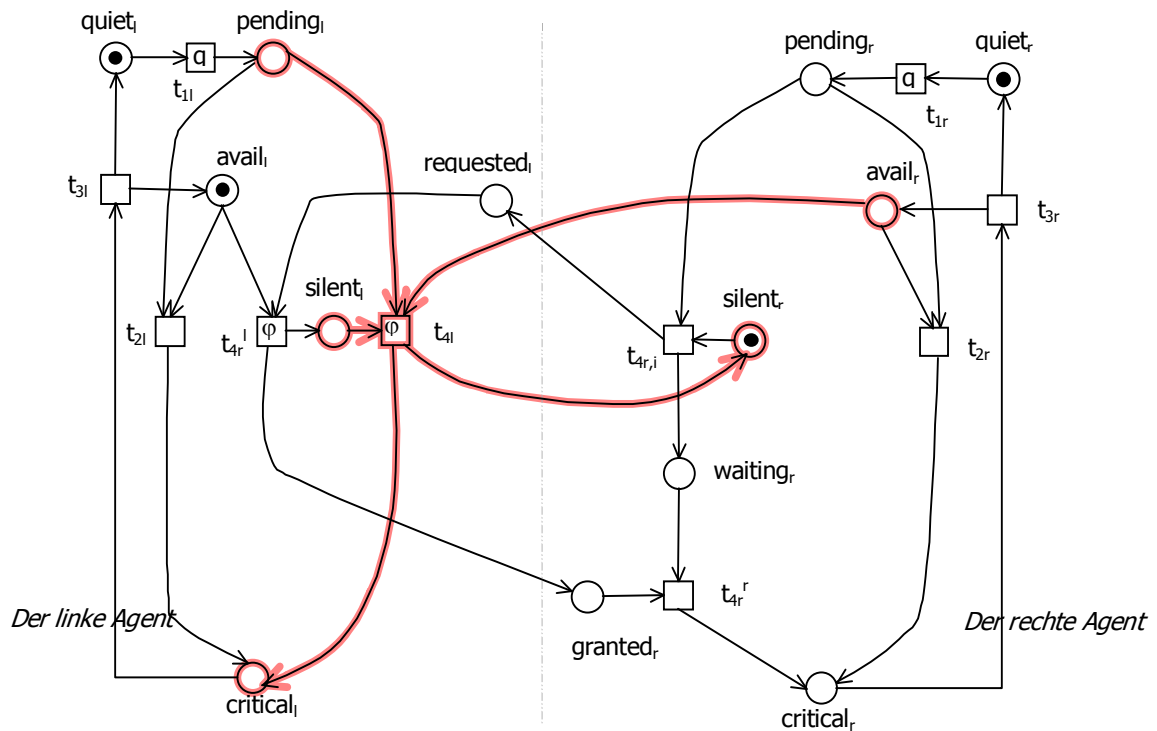
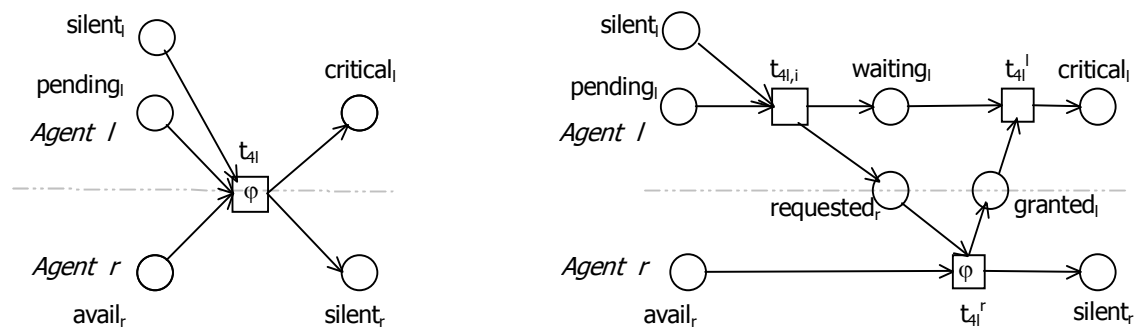


Abb. 7.1.8 System Σ_1 (vor der Verteilung von t_{4l})



(a) Transition t_{4l}

(b) Ersetzungsnetz $N_{t_{4l}}$

Abb. 7.1.9 Die unverteilter Transition t_{4l} und ihr verteiltes Ersetzungsnetz $N_{t_{4l}}$

Kausalität

Im lokalen Teilsystem $(N_{t_{4l}}, t_{4l}^-)$ gilt offensichtlich die folgende Kausalität:

$$\bullet t_{4l} \blacktriangleleft t_{4l} \bullet$$

Blockbedingung

Wie beim Verteilen von t_{4r} benutzen wir hier den dritten Grundtyp (vgl. Satz 6.4.2 in Abschnitt 6.4) zum Nachweisen, dass die Blockbedingung erfüllt ist.

Erhaltenbleiben von Eigenschaften

Auch wie beim Verteilen von t_{4r} benutzen wir hier die Sätze über Erhaltenbleiben der Eigenschaften durch Verfeinerung mit Kausalitäten aus Kapitel 3 (vgl. Lemma 3.5.12, Lemma 3.5.9 in Abschnitt 3.5) zum Nachweisen, dass alle gewünschten Eigenschaften erhalten bleiben.

In Abb. 7.1.10 ist das erhaltene System Σ^* nach der Verteilung von t_{4l} angegeben. Σ^* ist schon unser Ziel-Algorithmus – ein verteilter *Mutex*-Algorithmus.

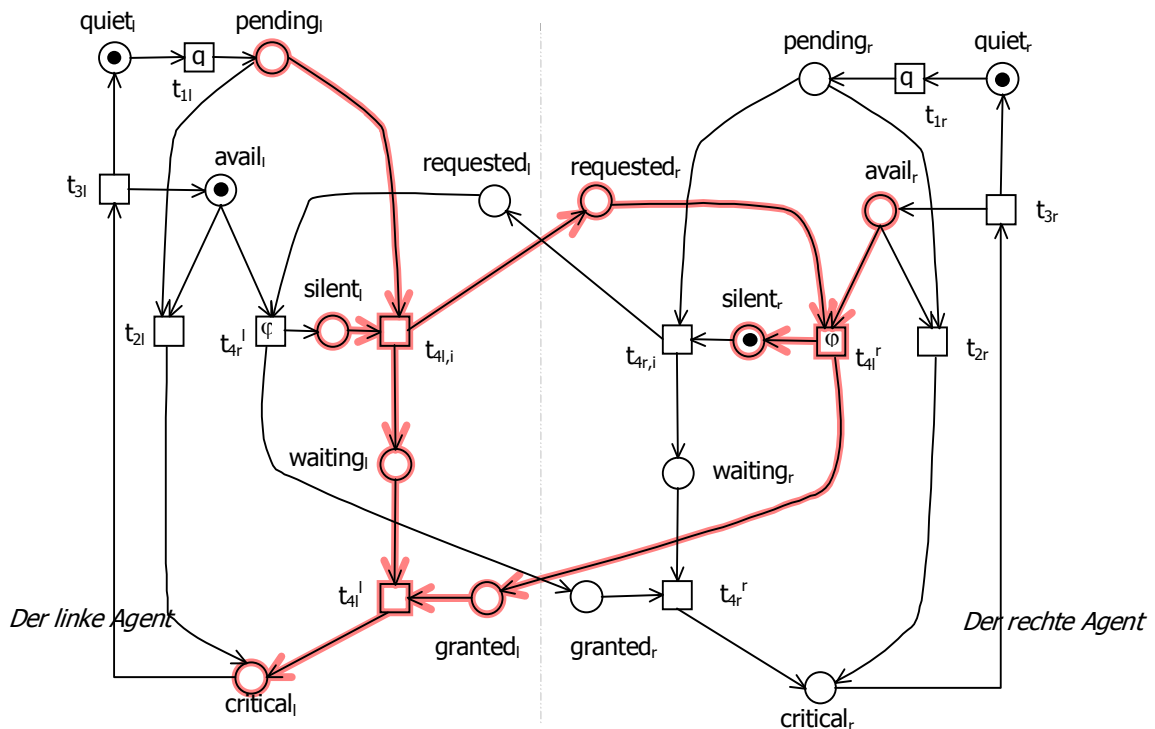


Abb. 7.1.10 System Σ^* (nach der Verteilung von t_{4l})

3. Die Verifikation des verteilten *Mutex*-Algorithmus

Die Korrektheit des Anfangsalgorithmus

Lemma 7.1.1 Σ_0 erfüllt (S), (L1) und (L2).

Beweis: siehe Abschnitt 4.3.1.

Die Korrektheit der verteilenden Verfeinerung von t_{4r}

Offensichtlich gilt die folgende Aussage:

Lemma 7.1.2 $(N_{t_{4r}}, t_{4r}^-) \models \bullet t_{4r} \blacktriangleleft t_{4r} \bullet$.

Lemma 7.1.3 $\Sigma_1 = \Sigma_0(N_{t_{4r}} \setminus t_{4r})$ erfüllt die Blockbedingung.

Beweis:

Wir benutzen den dritten Grundtyp (in Abschnitt 6.4).

Die Umgebung von t_{4r} erfüllt die folgenden Bedingungen:

- In Σ_0 gilt (vgl. (4-3-14)): $pending_r \wedge silent_r \triangleright avail_l$.
- t_{2r} ist nicht aktiviert, wenn $pending_r \wedge silent_r$ gilt (wegen Invariante $critical_r + avail_r + silent_r = 1$).

Das Ersetzungsnetz $N_{t_{4r}}$ für t_{4r} erfüllt die folgenden Bedingungen:

- In $(N_{t_{4r}}, t_{4r}^-)$ gelten:

Anfangstransition $t_{4r,i}$ von $r \blacktriangleleft$ Anfangstransition t_{4r}^l von l

und $\bullet t_{4r} \blacktriangleleft t_{4r} \bullet$.

- Anfangstransition t_{4r}^l von l ist *fair*.

Aus Satz 6.4.2 folgt: $(\Sigma_0 \rightarrow \Sigma_1)$ erfüllt die Blockbedingung. □

Lemma 7.1.4

Beim Verfeinerungsschritt $(\Sigma_0 \rightarrow \Sigma_1)$ bleiben (S), (L1) und (L2) erhalten.

Beweis:

Aus Lemma 3.5.12 folgt: Bei $(\Sigma_0 \rightarrow \Sigma_1)$ bleibt (S) erhalten (wegen Lemma 7.1.2, Lemma 7.1.3).

Aus Lemma 3.5.9 folgt: Bei $(\Sigma_0 \rightarrow \Sigma_1)$ bleiben (L1) und (L2) erhalten. \square

Lemma 7.1.5

Bei $(\Sigma_0 \rightarrow \Sigma_1)$ bleibt die folgende Eigenschaft erhalten:

$$pending_l \wedge silent_l \triangleright avail_r .$$

Beweis: Folgt direkt aus Lemma 3.5.11, Lemma 7.1.3 und Lemma 7.1.2.

Lemma 7.1.5 verwenden wir zum Nachweis der Blockbedingung bei der Verteilung von t_{4l} .

Die Korrektheit der verteilenden Verfeinerung von t_{4l}

Offensichtlich gilt die folgende Aussage:

$$\text{Lemma 7.1.6} \quad (N_{t_{4l}}, t_{4l}^-) \models \bullet t_{4l} \blacktriangleleft t_{4l} \bullet.$$

Lemma 7.1.7 $\Sigma^* = \Sigma_1(N_{t_{4l}} \setminus t_{4l})$ erfüllt die Blockbedingung.

Beweis:

Wie bei Lemma 7.1.3 benutzen wir den dritten Grundtyp (in Abschnitt 6.4).

Die Umgebung von t_{4l} erfüllt die folgenden Bedingungen:

- In Σ_1 gilt: $pending_l \wedge silent_l \triangleright avail_r$ (wegen (4-3-14), Lemma 7.1.5).
- t_{2r} ist nicht aktiviert, wenn $pending_l \wedge silent_l$ gilt (wegen (4.3.12)).

Das Ersetzungsnetz $N_{t_{4l}}$ für t_{4l} erfüllt die folgenden Bedingungen:

- In $(N_{t_{4l}}, t_{4l}^-)$ gelten:

Anfangstransition $t_{4l,i}$ von $l \blacktriangleleft$ Anfangstransition t_{4l}^r von r
und $\bullet t_{4l} \blacktriangleleft t_{4l} \bullet$.

- Anfangstransition t_{4l}^r von r ist *fair*.

Aus Satz 6.4.2 folgt: $(\Sigma_1 \rightarrow \Sigma^*)$ erfüllt die Blockbedingung. \square

Lemma 7.1.8

Beim Verfeinerungsschritt $(\Sigma_1 \rightarrow \Sigma^*)$ bleiben (S), (L1) und (L2) erhalten.

Beweis:

Aus Lemma 3.5.12 folgt: Bei $(\Sigma_1 \rightarrow \Sigma^*)$ bleibt (S) erhalten (wegen Lemma 7.1.6, Lemma 7.1.7).

Aus Lemma 3.5.9 folgt: Bei $(\Sigma_1 \rightarrow \Sigma^*)$ bleiben (L1) und (L2) erhalten. □

Schließlich erhalten wir, dass der Ziel-Algorithmus die gewünschten Eigenschaften erfüllt.

Die Korrektheit des Zielalgorithmus Σ^*

Satz 7.1.9 Σ^* erfüllt (S), (L1) und (L2).

Beweis: Folgt direkt aus Lemma 7.1.1, Lemma 7.1.4 und Lemma 7.1.8.

4. Verifikation des verteilten *Mutex*-Algorithmus mit Hilfe der Vertauschbarkeit der Verfeinerungsschritte

Wenn wir den Satz zur gleichzeitigen Verfeinerung (vgl. Abschnitt 6.5) verwenden, dann ist der Beweis noch einfacher.

Lemma 7.1.10 $\Sigma_2 = \Sigma_0(N_{t_{4l}} \setminus t_{4l})$ erfüllt die Blockbedingung.

Beweis: ähnlich wie bei Lemma 7.1.3.

Satz 7.1.11

$\Sigma_0 \rightarrow \Sigma_I \rightarrow \Sigma_2'$ mit $\Sigma_1 = \Sigma_0(N_{t_{4r}} \setminus t_{4r})$, $\Sigma_2' = \Sigma_1(N_{t_{4l}} \setminus t_{4l})$ ist Halbordnung vollständig erhaltend und Σ^* mit $\Sigma^* = \Sigma_2'$ erfüllt (S), (L1) und (L2).

Beweis:

Wegen Lemma 7.1.3, Lemma 7.1.2, Lemma 7.1.10 und Lemma 7.1.6 und Satz 6.5.2 folgt: $\Sigma_0 \rightarrow \Sigma_I \rightarrow \Sigma_2'$ ist Halbordnung vollständig erhaltend.

Bei jedem Verfeinerungsschritt von $\Sigma_0 \rightarrow \Sigma_I \rightarrow \Sigma_2'$ bleiben (S), (L1) und (L2) erhalten. (wegen Lemma 3.5.12 und Lemma 3.5.9)

Wegen Lemma 7.1.1 gilt: In Σ^* gelten (S), (L1) und (L2). □

7.1.3 Anwendungsbeispiel – *asymmetrischer* *Mutex-Algorithmus*

1. Grundmodell und Anfangsalgorithmus für verteilende Verfeinerung

Grundmodell

Wie beim *Token-passing Mutex* verwenden wir hier das gleiche Grundmodell Σ_a (siehe Abb. 7.1.1). Die gewünschten Eigenschaften sind (S), (L1) und (L2).

Anfangsalgorithmus

Die algorithmische Idee ist hier folgende: Die Ressource gehört dem linken Agenten l . Der rechte Agent r kann die Ressource ausleihen, muss sie aber nach der Benutzung sofort zurück geben.

Nach dieser algorithmischen Idee ist die Benutzung der Ressource von l eine lokale Aktion. Dagegen ist die Benutzung der Ressource von r keine lokale Aktion und lässt sich (in einem verteilten System) durch Kommunikation erreichen. Also, die beiden Agenten verhalten sich unterschiedlich (asymmetrisch).

Obwohl die Ressource dem linken Agenten gehört, muss er *fair* zu dem rechten Agenten (bzgl. Benutzung der Ressource) sein.

In Abb. 7.1.11 ist der Anfangsalgorithmus Σ_0 angegeben. Die Stelle $lent_l$ vom Agenten l modelliert, ob er die Ressource ausgeliehen hat¹⁴.

¹⁴ Wenn es vom Agenten l keine zusätzliche Stelle $lent_l$ gibt, dann wird der Zustand vom Agenten l , dass die Ressource ausgeliehen ist, durch die Stelle $critical_r$ vom Agenten r beschrieben.

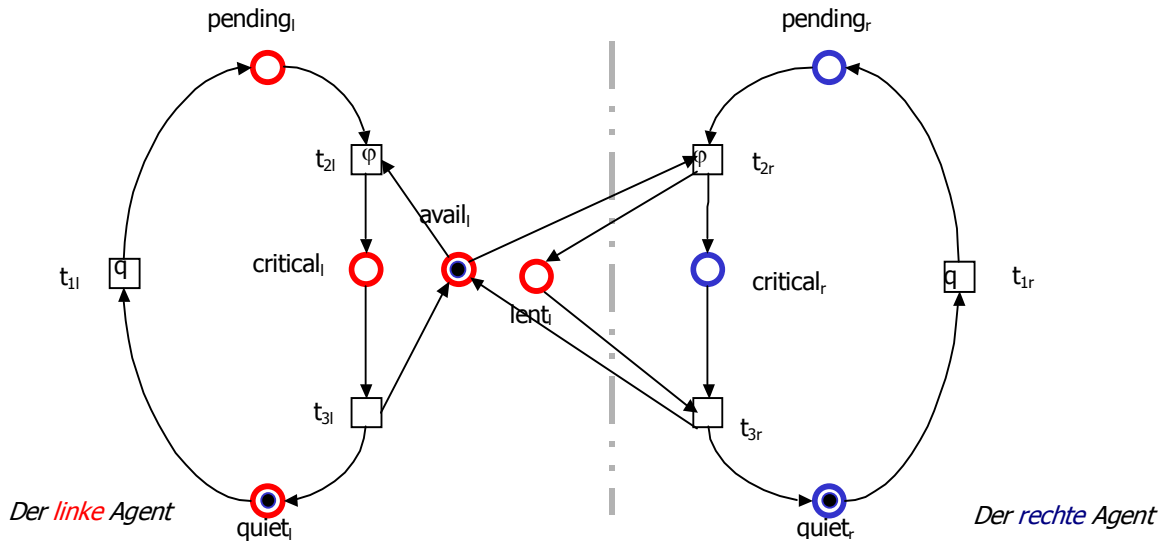


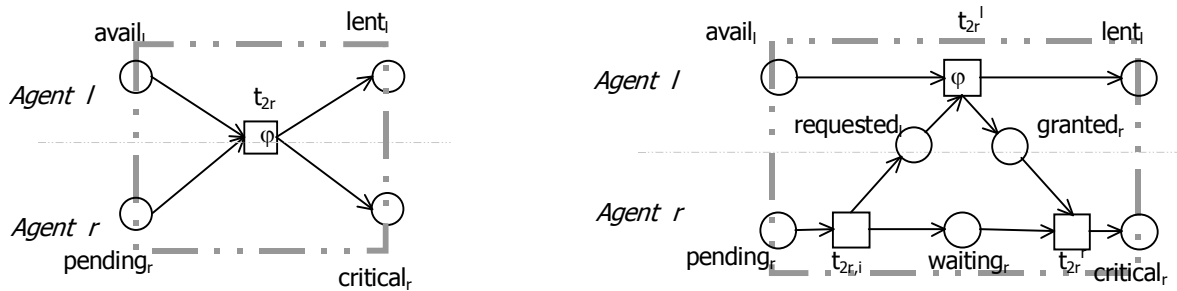
Abb. 7.1.11 Anfangsalgorithmus Σ_0

In Σ_0 sind die Transitionen t_{2r} und t_{3r} unverteilt. Wir werden in zwei Schritten jeweils diese unverteilt Transitionen verteilen. Schließlich erhalten wir den verteilten *Mutex*-Algorithmus Σ^* .

2. Die verteilende Verfeinerung

Die Verteilung der unverteilt Transition t_{2r}

Im ersten Schritt verteilen wir die unverteilt Transition t_{2r} zu einem verteilten Ersetzungsnetz $N_{t_{2r}}$ (Abb. 7.1.12).



(a) Transition t_{2r}

(b) Ersetzungsnetz $N_{t_{2r}}$

Abb. 7.1.12 Die unverteilt Transition t_{2r} und ihr verteiltes Ersetzungsnetz $N_{t_{2r}}$

Kausalität

Im lokalen Teilsystem $(N_{t_{2r}}, t_{2r}^-)$ gilt offensichtlich die folgende Kausalität:

$$\bullet t_{2r} \blacktriangleleft t_{2r} \bullet$$

Blockbedingung

Die Umgebung von t_{2r} erfüllt die folgenden Bedingungen:

- In Σ_0 gilt: $pending_r \triangleright avail_l$.
- t_{2r} hat keine Konflikt-Transition

Das Ersetzungsnetz $N_{t_{2r}}$ für t_{2r} erfüllt die folgenden Bedingungen:

- In $(N_{t_{2r}}, t_{2r}^-)$ gelten:

Anfangstransition $t_{2r,i}$ von $r \blacktriangleleft$ Anfangstransition t_{2r}^l von l

$$\text{und } \bullet t_{2r} \blacktriangleleft t_{2r} \bullet.$$

- Anfangstransition t_{2r}^l von l ist *fair*.

Wir benutzen den dritten Grundtyp (vgl. Satz 6.4.2 in Abschnitt 6.4) zum Nachweisen, dass die Blockbedingung erfüllt ist.

Erhaltenbleiben von Eigenschaften

Wir benutzen die Sätze über das Erhaltenbleiben der Eigenschaften durch Verfeinerung mit Kausalitäten aus Kapitel 3 (vgl. Lemma 3.5.12, Lemma 3.5.9 in Abschnitt 3.5) zum Nachweisen, dass alle gewünschten Eigenschaften erhalten bleiben.

In Abb. 7.1.14 ist das erhaltene System Σ_1 nach der Verteilung von t_{2r} angegeben.

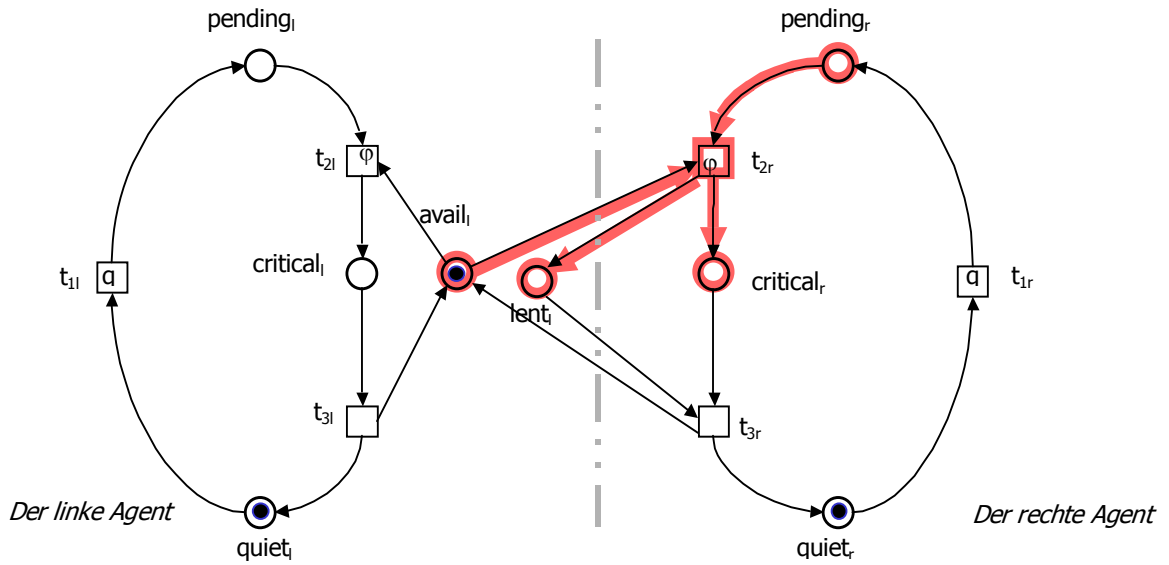


Abb. 7.1.13 System Σ_0 (vor der Verteilung von t_{2r} , die Kanten von und nach t_{2r} sind rot hervorgehoben.)

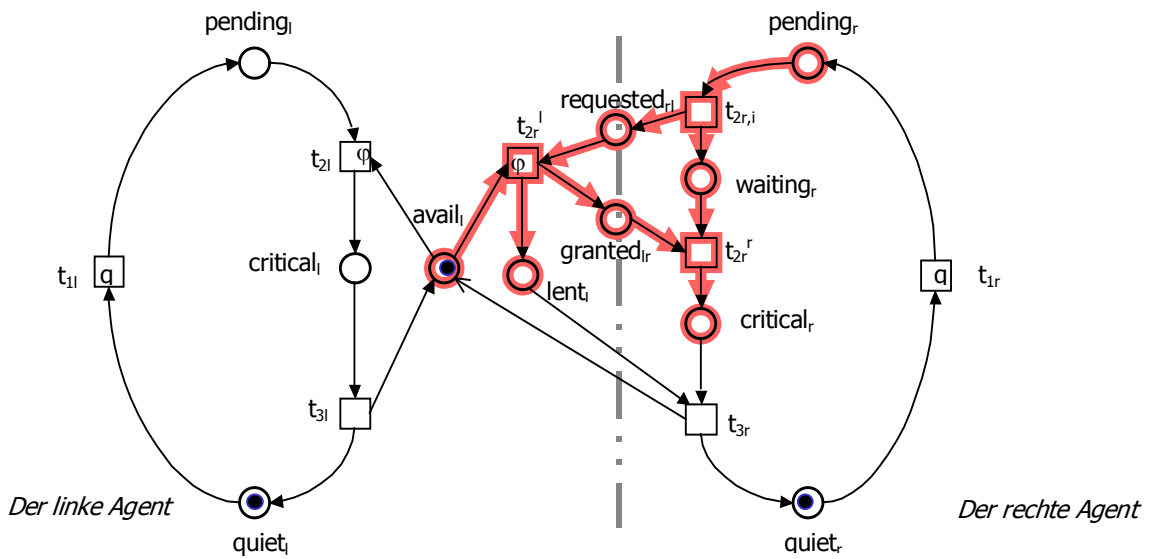


Abb. 7.1.14 System Σ_1 (nach der Verteilung von t_{2r})

Die Verteilung der unverteilt Transition t_{3r}

Wir verteilen die unverteilt Transition t_{3r} zu einem verteilten Ersetzungsnetz $N_{t_{3r}}$ (Abb. 7.1.15).

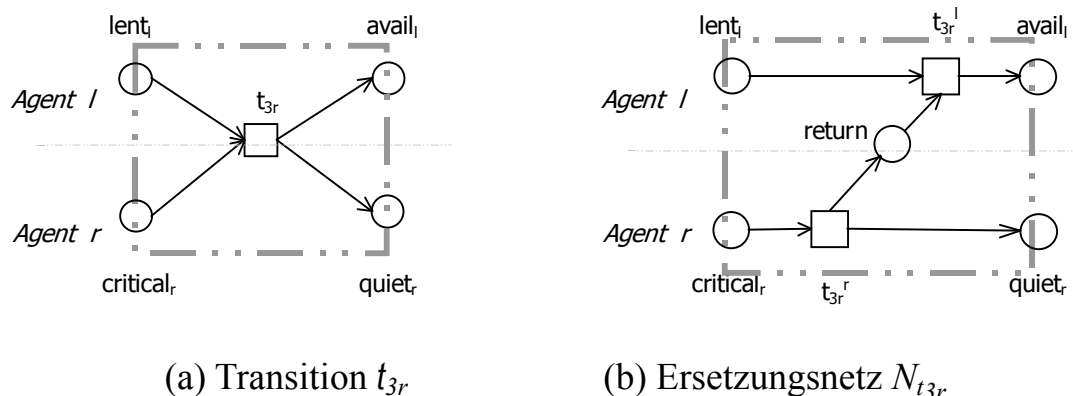


Abb. 7.1.15 Die unverteilt Transition t_{3r} und ihr verteiltes Ersetzungsnetz $N_{t_{3r}}$

Synchronisationsbedingung

Im lokalen Teilsystem $(N_{t_{3r}}, t_{3r}^-)$ gilt offensichtlich die folgende Synchronisationsbedingung:

$$critical_r \bullet \blacktriangleleft \bullet avail_l$$

Blockbedingung

Die Umgebung von t_{3r} erfüllt die folgenden Bedingungen:

- In Σ_1 gilt: $\square (critical_r \rightarrow lent_l)$.
- t_{3r} hat an der Stelle $lent_l$ keine Konflikt-Transition

Das Ersetzungsnetz $N_{t_{3r}}$ für t_{3r} erfüllt die folgende Bedingung:

- In $(N_{t_{3r}}, t_{3r}^-)$ gilt:

Anfangstransition t_{3r}^r von $r \blacktriangleleft$ Anfangstransition t_{3r}^l von l .

Wir benutzen den zweiten Grundtyp (vgl. Satz 6.3.1 in Abschnitt 6.3) zum Nachweisen, dass die Blockbedingung erfüllt ist.

Erhaltenbleiben von Eigenschaften

Wir benutzen die Sätze über Erhaltenbleiben der Eigenschaften durch Verfeinerung mit Synchronisationsbedingung aus Kapitel 5 (vgl. Lemma 5.4.4, Lemma 5.4.7 in Abschnitt 5.4) zum Nachweisen, dass alle

gewünschten Eigenschaften erhalten bleiben.

In Abb. 7.1.17 ist das erhaltene System Σ^* nach der Verteilung von t_{3r} angegeben. Σ^* ist schon unser Ziel-Algorithmus – ein verteilter *Mutex*-Algorithmus.

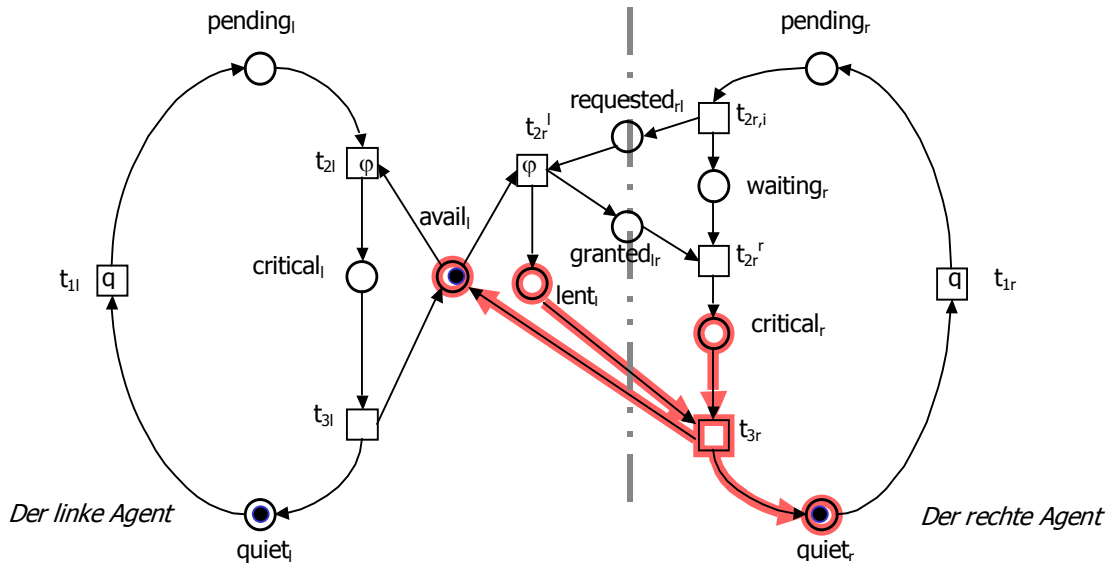


Abb. 7.1.16 System Σ_1 (vor der Verteilung von t_{3r})

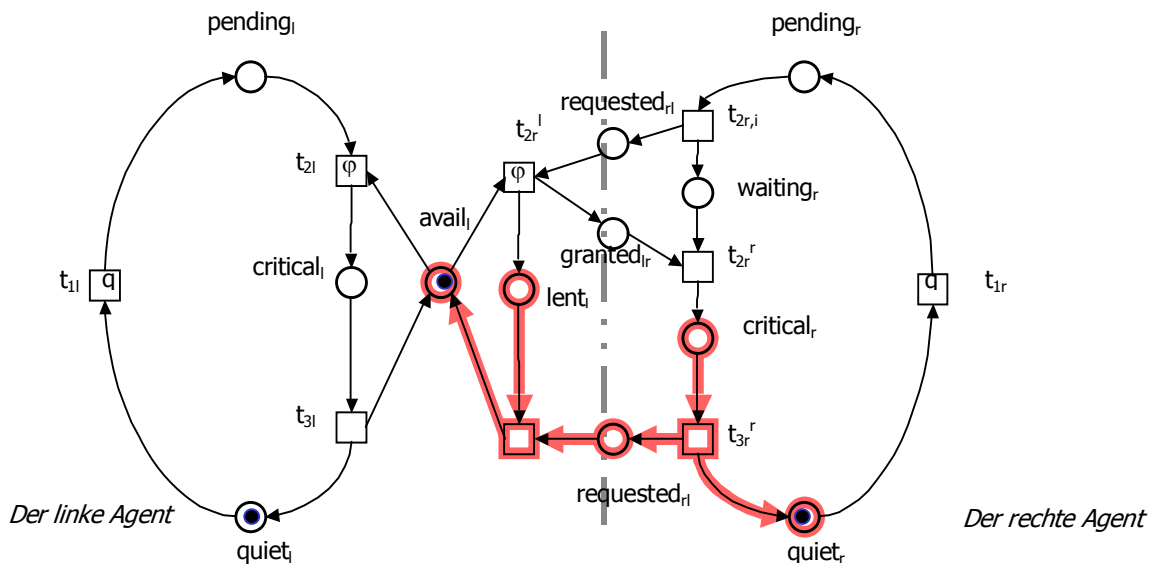


Abb. 7.1.17 System Σ^* (nach der Verteilung von t_{3r})

3. Die Verifikation des verteilten *Mutex*-Algorithmus

Zunächst geben wir einige Eigenschaften von Σ_0 an.

In Σ_0 gelten:

$$\square (\text{quiet}_r + \text{pending}_r + \text{critical}_r = 1) \quad (7-1-1)$$

$$\square (\text{quiet}_l + \text{pending}_l + \text{critical}_l = 1) \quad (7-1-2)$$

$$\square (\text{critical}_l + \text{avail}_l + \text{critical}_r = 1) \quad (7-1-3)$$

$$\square (\text{critical}_l + \text{avail}_l + \text{lent}_l = 1) \quad (7-1-4)$$

$$\square (\text{critical}_r \rightarrow \text{lent}_l) \quad (7-1-5)$$

$$\text{pending}_r \triangleright \text{avail}_l \quad (7-1-6)$$

Die Korrektheit des Anfangsalgorithmus

Lemma 7.1.12 Σ_0 erfüllt (S), (L1) und (L2).

Beweis: gleich wie der Beweis von Σ_a in Abschnitt 4.3.1.

Die Korrektheit der verteilenden Verfeinerung von t_{2r}

Offensichtlich gilt die folgende Aussage:

Lemma 7.1.13 $(N_{t_{2r}}, t_{2r}^-) \models \bullet t_{2r} \blacktriangleleft t_{2r} \bullet$.

Lemma 7.1.14 $\Sigma_1 = \Sigma_0(N_{t_{2r}} \setminus t_{2r})$ erfüllt die Blockbedingung.

Beweis:

Wir benutzen den dritten Grundtyp (in Abschnitt 6.4).

Die Umgebung von t_{2r} erfüllt die folgenden Bedingungen:

- In Σ_0 gilt (wegen (7-1-6)): $\text{pending}_r \triangleright \text{avail}_l$.
- An der Stelle pending_r gibt es keine alternative Transition.

Das Ersetzungsnetz $N_{t_{2r}}$ für t_{2r} erfüllt die folgenden Bedingungen:

- In $(N_{t_{2r}}, t_{2r}^-)$ gelten:

Anfangstransition $t_{2r,i}$ von $r \blacktriangleleft$ Anfangstransition t_{2r}^l von l

und $\bullet t_{2r} \blacktriangleleft t_{2r} \bullet$.

- Anfangstransition t_{2r}^l von l ist *fair*.

Aus Satz 6.4.2 folgt: $(\Sigma_0 \rightarrow \Sigma_1)$ erfüllt die Blockbedingung. \square

Lemma 7.1.15

Beim Verfeinerungsschritt $(\Sigma_0 \rightarrow \Sigma_1)$ bleiben (S), (L1) und (L2) erhalten.

Beweis:

Aus Lemma 3.5.12 folgt: Bei $(\Sigma_0 \rightarrow \Sigma_1)$ bleibt (S) erhalten (wegen Lemma 7.1.13, Lemma 7.1.14).

Aus Lemma 3.5.9 folgt: Bei $(\Sigma_0 \rightarrow \Sigma_1)$ bleiben (L1) und (L2) erhalten. \square

Lemma 7.1.16

Bei $(\Sigma_0 \rightarrow \Sigma_1)$ bleiben die folgenden Eigenschaften erhalten:

- \square $(critical_r \rightarrow lent_l)$;
- \square $\neg (pending_r \wedge critical_r)$;
- \square $\neg (pending_l \wedge critical_l)$;
- \square $(critical_l + avail_r + critical_l \leq 1)$.

Beweis:

(i) folgt direkt aus Lemma 5.4.5 (wegen $(N_{t_{2r}}, t_{2r}^-) \models \bullet lent_l \blacktriangleleft \bullet critical_r$).

(ii)-(iv) folgen direkt aus Lemma 3.5.12, Lemma 7.1.14 und Lemma 7.1.13. \square

Lemma 7.1.16 (i) werden wir zum Nachweis der Blockbedingung bei der Verteilung von t_{3r} benutzen.

Lemma 7.1.16 (ii)-(iv) werden wir zum Nachweis des Erhaltenbleiben von Eigenschaften bei der Verteilung von t_{3r} benutzen.

Die Korrektheit der verteilenden Verfeinerung von t_{3r}

Offensichtlich gilt die folgende Aussage:

Lemma 7.1.17 $(N_{t_{3r}}, t_{3r}^-) \models \text{critical}_r \bullet \blacktriangleleft \bullet \text{avail}_l$.

Lemma 7.1.18 $\Sigma^* = \Sigma_1(N_{t_{3r}} \setminus t_{3r})$ erfüllt die Blockbedingung.

Beweis:

Wir benutzen den zweiten Grundtyp (in Abschnitt 6.3).

Die Umgebung von t_{3r} erfüllt die folgenden Bedingungen:

- In Σ_1 gilt: $\square (\text{critical}_r \rightarrow \text{lent}_l)$ (wegen (7-1-5), Lemma 7.1.16(i)).
- t_{3r} hat an der Stelle lent_l keine Konflikt-Transition

Das Ersetzungsnetz $N_{t_{3r}}$ für t_{3r} erfüllt die folgende Bedingung:

- In $(N_{t_{3r}}, t_{3r}^-)$ gilt:

Anfangstransition t_{3r}^r von $r \blacktriangleleft$ Anfangstransition t_{3r}^l von l .

Aus Satz 6.3.1 folgt: $(\Sigma_1 \rightarrow \Sigma^*)$ erfüllt die Blockbedingung. □

Lemma 7.1.19

Beim Verfeinerungsschritt $(\Sigma_1 \rightarrow \Sigma^*)$ bleiben (S), (L1) und (L2) erhalten.

Beweis:

Aus Lemma 5.4.4 und Lemma 7.1.17 folgt: Bei $(\Sigma_1 \rightarrow \Sigma^*)$ bleibt (S) erhalten (wegen (7-1-3), Lemma 7.1.16 (iv)).

Aus Lemma 5.4.7 und Lemma 7.1.17 folgt: Bei $(\Sigma_1 \rightarrow \Sigma^*)$ bleiben (L1) und (L2) erhalten (wegen (7-1-1), (7-1-2) und Lemma 7.1.16 (ii),(iii)). □

Schließlich erhalten wir, dass der Ziel-Algorithmus die gewünschten Eigenschaften erfüllt.

Die Korrektheit des Zielalgorithmus Σ^*

Satz 7.1.20 Σ^* erfüllt (S), (L1) und (L2).

Beweis:

Folgt direkt aus Lemma 7.1.12, Lemma 7.1.15 und Lemma 7.1.19. □

7.2 Entwurf verteilter Algorithmen durch verteilende Verfeinerung

7.2.1 Entwurf durch verteilende Verfeinerung

Der Entwurf durch verteilende Verfeinerung umfasst drei Schritte:

1. Die Grundidee des Algorithmus wird durch ein einfaches System, das Grundmodell, dargestellt. Die geforderten Eigenschaften des Systems werden formuliert.
2. Das Grundmodell wird um eine algorithmische Idee zu einem Algorithmus erweitert und zwar so, dass ein unverteilter Agentensystem (Anfangsalgorithmus) entsteht. Die Gültigkeit der geforderten Eigenschaften des Systems wird bewiesen.
3. Die unverteilter Transitionen werden so verfeinert, dass die gewünschten Eigenschaften erhalten bleiben.

7.2.2 Anwendungsbeispiel *crossstalk*-Algorithmus

Zur Illustration des Entwurfs wird in diesem Abschnitt der *Crosstalk*-Algorithmus durch verteilende Verfeinerung erzeugt. Hierbei werden drei unverteilter Transitionen verteilt. Die Korrektheit des Entwurfs wird bewiesen.

1. Grundidee des *crossstalk*-Algorithmus

Wir betrachten zunächst das folgende System (vgl. Abb. 7.2.1): Zwei Partner – der linke Agent l und der rechte Agent r – werden wiederholend gemeinsame Aktionen zur Kommunikation ausführen. Sie haben drei unterschiedliche Aufgaben: *Aufgabe*₁, *Aufgabe*₂ und *Aufgabe*₃.

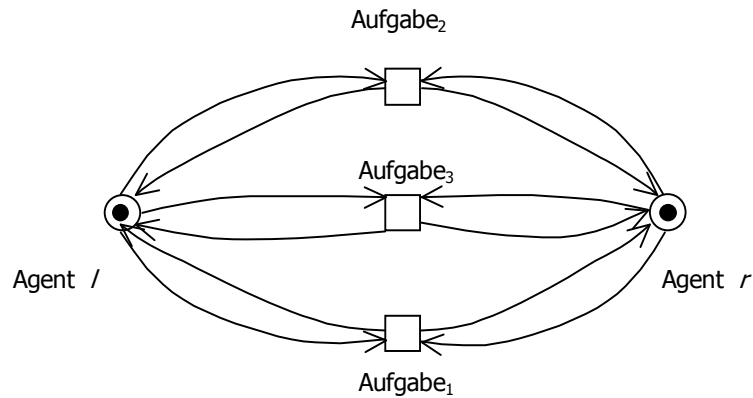


Abb. 7.2.1 *Crosstalk*

Jede der drei Aufgaben wird vollständig bearbeitet, bevor die nächste Aufgabe begonnen wird. Bei dem *crosstalk*-Algorithmus sind die drei Aufgaben:

Aufgabe₁ besteht aus folgenden Aktionen:

- der Agent *r* sendet eine Nachricht (Aktion *senden_r*);
- der Agent *l* antwortet auf diese Nachricht (Aktion *antworten_l*);
- der Agent *r* empfängt die Antwort (Aktion *empfangen_r*).

Aufgabe₂ besteht aus folgenden Aktionen:

- der Agent *l* sendet eine Nachricht (Aktion *senden_l*);
- der Agent *r* antwortet auf diese Nachricht (Aktion *antworten_r*);
- der Agent *l* empfängt die Antwort (Aktion *empfangen_l*).

Aufgabe₃ besteht aus folgenden Aktionen:

- beide Agenten senden jeweils eine Nachricht (Aktionen *senden_l*, *senden_r*);
- beide Agenten antworten jeweils auf die Nachricht (Aktionen *antworten_l*, *antworten_r*);
- beide Agenten empfangen jeweils die Antworten (Aktionen *empfangen_l*, *empfangen_r*).

Dieses Grundmodell des *crosstalk*-Algorithmus beschreibt die einzelnen Aktionen der Agenten verbal. Um die geforderten Eigenschaften des Algorithmus formalisieren zu können, müssen wir zunächst definieren, wie wir die Aufgaben der einzelnen Agenten formal darstellen.

2. Spezifikation von Aufgaben eines unverteilt Systems mit Agenten und Kanälen

Durch Agentensysteme haben wir Algorithmen modelliert, in denen mehrere Agenten gemeinsam vorgegebene Aufgaben lösen. In dem folgenden Netz *producer/consumer*

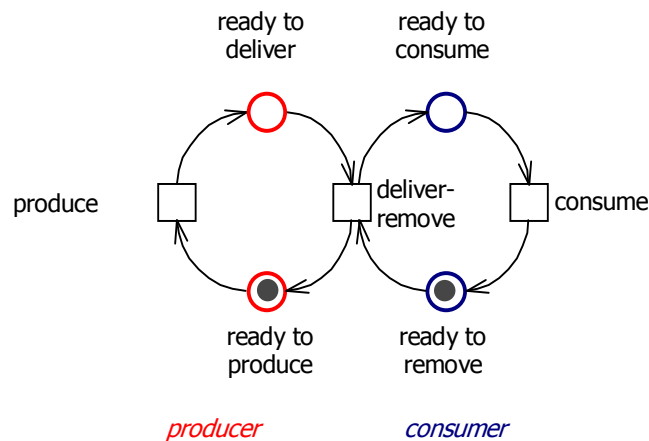


Abb. 7.2.2 System *producer/consumer*

bestehen die Aufgaben des Producers aus dem wiederholten Erzeugen eines Produktes (Transition *produce*) und der Übergabe des Produktes an den Consumer (Transition *deliver-remove*). Die Aufgaben des Consumers bestehen aus dem Abholen dieses Produktes (Transition *deliver-remove*) und dem Konsumieren des Produktes (Transition *consume*). Der Zusammenhang zwischen den Aufgaben des modellierten Algorithmus und dem Petrinetz wurde auf informale Weise durch die Namen der Transitionen hergestellt. Unverteilt Transitionen werden im modellierten Algorithmus mehrere Aufgaben zugeordnet. Um alle Teilaufgaben einer Transition beschreiben zu können, haben wir entweder lange Namen verwendet oder im Text genauer die Aufgaben einer unverteilt Transition beschrieben.

Aufgabenspezifikation

Eine Aufgabenspezifikation soll z.B. $senden_l; empfangen_r$ sein. Ein Agent l hat die Aufgabe 'senden'. Danach soll ein Agent r die Aufgabe 'empfangen_r' haben. Aufgaben sind also Bezeichner. Agenten sind die Agenten des Agentensystems. Die Reihenfolge der Aufgaben wird durch Symbole festgelegt. Die Aufgabenspezifikation $senden_l, empfangen_r$ besagt, die Agenten l und r senden beide in beliebiger Reihenfolge.

<i>Aufgabenbezeichnung:</i>	Bezeichner
<i>Agent:</i>	Bezeichner
<i>elementare Aufgabe:</i>	Aufgabenbezeichnung _{Agent}
<i>Term:</i>	elementare Aufgabe {Aufgabenspezifikation }
<i>Sequenz:</i>	Term Sequenz ; Term
<i>Aufgabenspezifikation:</i>	Sequenz Aufgabenspezifikation , Sequenz

Eine etwas umfassendere Aufgabenspezifikation ist:

$\{ senden_l, senden_r \} ; \{ antworten_l, antworten_r \} ; \{ empfangen_l, empfangen_r \}$

elementare Aufgaben (Aufgabenspezifikation) =_{def} Menge der in der Aufgabenspezifikation auftretenden elementaren Aufgaben.

Semantik einer Aufgabenspezifikation bzw. einer Transition

Wir verwenden Aufgabenspezifikationen als Namen von Transitionen. Damit ordnen wir den Transitionen eine Semantik zu. Das Ziel der Aufgabenspezifikation ist es, die elementaren Aufgaben der Agenten festzulegen und die Reihenfolge anzugeben, in der diese Aufgaben erfüllt werden sollen. Falls eine Verfeinerung diese Reihenfolgebeziehungen beachtet, werden diese Reihenfolgebeziehungen der \blacktriangleleft Relation entsprechen. Deshalb werden wir die \blacktriangleleft Relation zur Bezeichnung der Reihenfolgebeziehungen verwenden.

Für $a, b \in$ elementare Aufgaben (Aufgabenspezifikation) gilt $a \blacktriangleleft b$ gdw. es existiert ein Teilausdruck der Aufgabenspezifikation der Form Sequenz ; Term mit $a \in$ elementare Aufgaben (Sequenz), $b \in$ elementare Aufgaben (Term).

Gilt $t' \blacktriangleleft t$ und $a \in$ elementare Aufgaben (t), so wird definiert $t' \blacktriangleleft a$.

Gilt $t \blacktriangleleft t'$ und $a \in$ elementare Aufgaben (t), so wird definiert $a \blacktriangleleft t'$.

Beispiel für ein einfaches unverteiltetes Netz:

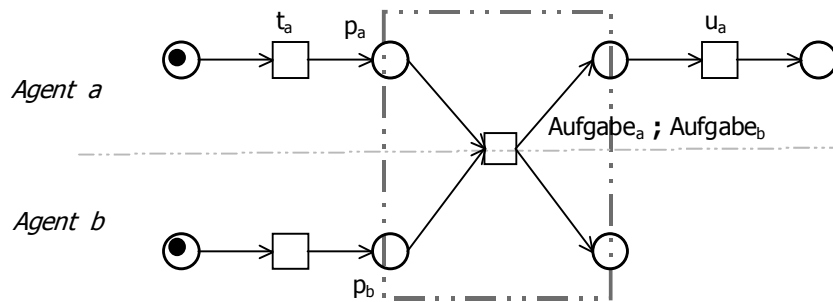


Abb. 7.2.3 Ein System Σ mit einer unverteilteten Transition $Aufgabe_a ; Aufgabe_b$

Es gilt z.B.:

$$t_a \triangleleft Aufgabe_a, \quad Aufgabe_a \triangleleft Aufgabe_b,$$

$$t_a \triangleleft Aufgabe_b, \quad Aufgabe_b \triangleleft u_a.$$

Verfeinerung konform zur Aufgabenspezifikation

Damit die Eigenschaften eines unverteilteten Systems erhalten bleiben, muss ein Verfeinerungsnetz N_t einer unverteilteten Transition t die durch den Namen t gegebene Aufgabenspezifikation beachten.

Im Beispiel der Transition $Aufgabe_a ; Aufgabe_b$ geschieht das durch folgendes Verfeinerungsnetz:

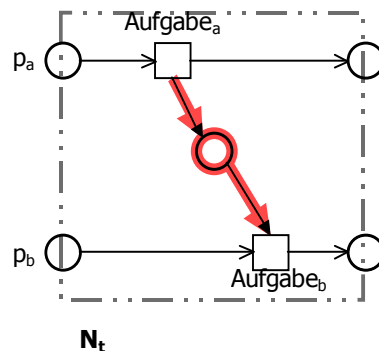


Abb. 7.2.4 Ersetzungsnetz für die Transition $Aufgabe_a ; Aufgabe_b$ in Abb. 7.2.3

Die Relation $Aufgabe_a \triangleleft Aufgabe_b$ wird durch dieses Verfeinerungsnetz erhalten. Außerdem haben wir eine eindeutige Abbildung der elementaren Aufgaben der Spezifikation (also $Aufgabe_a, Aufgabe_b$) in die Menge der Transitionen des Verfeinerungsnetzes, die in unserem Beispiel die Identität ist. Leider kann nicht immer die Identität als Abbildung benutzt werden. Das kommt daher, dass zwei verschiedene Aufgabenspezifikationen die gleiche elementare Aufgabe enthalten

dürfen. Die Verfeinerungsnetze zweier Transitionen dürfen aber nicht den gleichen Namen für eine Transition verwenden. In unseren Beispielen werden wir dieses Problem umgehen, indem wir den Namen elementarer Aufgaben mit keinem, einem oder mehreren Strichen versehen, wenn wir sie als Namen von Transitionen in Verfeinerungsnetzen verwenden.

Definition 7.2.1 (*Verfeinerungsnetz konform zu einer durch den Namen der Transition gegebenen Aufgabenspezifikation*)

Sei t der Name einer Transition, wobei t syntaktisch eine Aufgabenspezifikation ist. Ein Verfeinerungsnetz N_t ist konform zur Aufgabenspezifikation t , wenn es eine eindeutige Abbildung c elementare Aufgaben(t) in Transitionen(N_t) gibt mit: sind $a, b \in$ elementare Aufgaben(t) und $a \blacktriangleleft b$, so gilt im Verfeinerungsnetz $c(a) \blacktriangleleft c(b)$.

Durch die Festlegung, dass eine Aufgabenspezifikation durch den Namen der Transitionen gegeben wird, vermeiden wir eine Erweiterung der Netzdefinition.

Durch das Verwenden der Namen elementarer Aufgaben als Namen der Transitionen in Verfeinerungsnetzen (gegebenfalls ergänzt um ein oder mehrere Striche) vermeiden wir die explizite Angabe der Abbildung zwischen den elementaren Aufgaben und den Transitionen.

Bisher können wir nur Aussagen darüber machen, welche temporal-logischen Eigenschaften eines Systems Σ im verfeinerten System $\Sigma' = \Sigma(N_t \setminus t)$ erhalten bleiben (siehe Kapitel 3, Kapitel 5). Jetzt ist es auch möglich festzustellen, welche Aussagen über elementare Aufgaben einer Transition t erhalten bleiben.

Satz 7.2.2

Es gelte für $\Sigma' = \Sigma(N_t \setminus t)$ die Blockbedingung. Sei N_t eine zur Aufgabenspezifikation t konforme Verfeinerung von t und in (N_t, t^-) gelte $\bullet t \blacktriangleleft t^$.*

Sei e eine elementare Teilaufgabe von t , sei $c(e)$ der Transitionsname von e in N_t . Weiter sei $t' \in T$ mit $t' \neq t$.

- i. Wenn $\Sigma \models t' \blacktriangleleft e$ gilt und es ein $p \in \bullet t$ mit $\Sigma \models t' \blacktriangleleft p$ und $(N_t, t^-) \models p \blacktriangleleft c(e)$ gibt, dann gilt $\Sigma' \models t' \blacktriangleleft e$.*

- ii. Sei e' eine weitere Aufgabe von t , d.h. $e' \in \text{elementare Aufgaben}(t)$.
Dann bleibt $e \blacktriangleleft e'$ beim Verfeinerungsschritt $(\Sigma \rightarrow \Sigma')$ erhalten.
- iii. Sei e' eine Aufgabe, $T_{e'} := \{ t_e \in T \mid e' \in \text{elementare Aufgaben}(t_e) \}$
mit $T_{e'} \neq \emptyset$ und $t \notin T_{e'}$. Dann bleiben $t' \blacktriangleleft e'$ und $e' \blacktriangleleft t'$ beim
Verfeinerungsschritt $(\Sigma \rightarrow \Sigma')$ erhalten.
- iv. Seien e_1 und e_2 Aufgaben, $T_{e_1} := \{ t_{e_1} \in T \mid e_1 \in \text{elementare Aufgaben}(t_{e_1}) \}$, $T_{e_2} := \{ t_{e_2} \in T \mid e_2 \in \text{elementare Aufgaben}(t_{e_2}) \}$ mit:
 T_{e_1}, T_{e_2} sind nicht leer und enthalten t nicht. Dann bleibt $e_1 \blacktriangleleft e_2$
beim Verfeinerungsschritt $(\Sigma \rightarrow \Sigma')$ erhalten.

Beweis:

$\bullet t \blacktriangleleft t'$ sichert, dass $<$ in allen Abläufen erhalten bleibt. Bei der konformen Verfeinerung von t gelten zusätzlich in jedem Ablauf für jede $t \rightarrow N_i$ Ersetzung die zwischen den Teilaufgaben geforderten Relationen.

Sei $T_e := \{ t_e \in T \mid e \in \text{elementare Aufgaben}(t_e) \}$.

Zu (i):

Wegen $\Sigma \models t' \blacktriangleleft p$ gilt: $\Sigma' \models t' \blacktriangleleft p$. (wegen Satz 3.5.4)

Weiter gilt: $(N_b, t^-) \models p \blacktriangleleft c(e)$.

Daraus folgt: (wegen Bem. 3.5.6)

$$\Sigma' \models t' \blacktriangleleft c(e). \quad (1)$$

Für jedes $t_e \in T_e \setminus \{t\}$ gilt: (wegen Satz 3.5.4)

$$\text{Beim Verfeinerungsschritt } (\Sigma \rightarrow \Sigma') \text{ bleibt } t' \blacktriangleleft t_e \text{ erhalten.} \quad (2)$$

Aus (1), (2) folgt: $\Sigma' \models t' \blacktriangleleft e$.

Zu (ii):

Angenommen $\Sigma \models e \blacktriangleleft e'$. (3)

Weil N_i konform zu t ist, gilt $(N_b, t^-) \models c(e) \blacktriangleleft c(e')$.

Daraus folgt: $\Sigma' \models c(e) \blacktriangleleft c(e')$. (wegen Lemma 3.5.5) (4)

Für jedes $t_e \in T_e \setminus \{t\}$ und jedes $t_{e'} \in T \setminus \{t\}$ mit $e' \in \text{elementare Aufgaben}(t_{e'})$ gilt:

$$\Sigma \models t_e \blacktriangleleft t_{e'}. \quad (\text{wegen (3)})$$

Dann gilt: $\Sigma' \models t_e \blacktriangleleft t_{e'}$. (wegen Lemma 3.5.4) (5)

Aus (4), (5) folgt: $\Sigma' \models e \triangleleft e'$.

Zu (iii):

Für jedes $t_{e'} \in T_{e'}$ gilt: (wegen Satz 3.5.4)

Beim Verfeinerungsschritt $(\Sigma \rightarrow \Sigma')$ bleibt $t' \triangleleft t_{e'}$ erhalten.

Daraus folgt, bei $(\Sigma \rightarrow \Sigma')$ bleibt $t' \triangleleft e'$ erhalten.

Analog bleibt bei $(\Sigma \rightarrow \Sigma')$ $e' \triangleleft t'$ erhalten.

Zu (iv):

Angenommen $\Sigma \models e_1 \triangleleft e_2$.

Dann gibt es für jedes $t_{e_2} \in T_{e_2}$ ein $t_{e_1} \in T_{e_1}$ mit $t_{e_1} \triangleleft t_{e_2}$.

Bei $(\Sigma \rightarrow \Sigma')$ bleibt $t_{e_1} \triangleleft t_{e_2}$ erhalten. (wegen Satz 3.5.4)

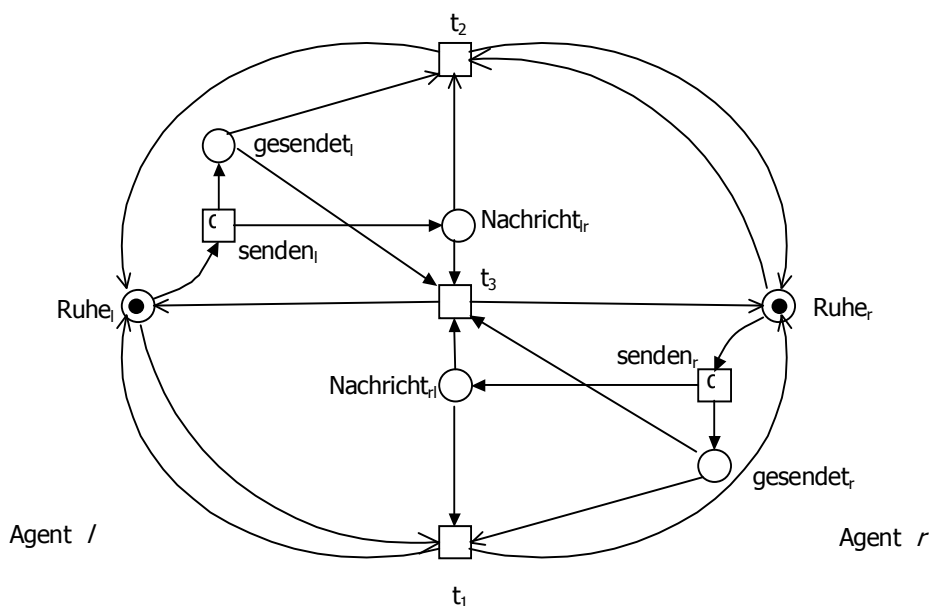
Daraus folgt: $\Sigma' \models e_1 \triangleleft e_2$.

□

3. Anfangsalgorithmus

In Abb. 7.2.5 ist ein etwas konkreteres Modell Σ_0 unter folgender Entwurfsentscheidung angegeben: Es wird für beide Agenten jeweils nur eine Sendeoperation modelliert. Ein Agent kann nur im *Ruhe*-Zustand eine Nachricht senden, und danach steht er im Zustand *gesendet*. Dann modelliert die Transition t_1 in Σ_0 $antworten_l; empfangen_r$, und die Transition t_2 modelliert $antworten_r; empfangen_l$, die Transition t_3 modelliert

$\{antworten_l, antworten_r\}; \{empfangen_l, empfangen_r\}$.



Abkürzungen für die Namen der Transitionen:

$t_1^?$ $antworten_l; empfangen_r$

$t_2^?$ $antworten_r; empfangen_l$

$t_3^?$ $\{antworten_l, antworten_r\}; \{empfangen_l, empfangen_r\}$

Fehler!

Abb. 7.2.5 System Σ_0

Seien $l = \{ Ruhe_l, gesendet_l \}$, $r = \{ Ruhe_r, gesendet_r \}$. Weiter seien die Agentenmenge $A = \{ l, r \}$, die Kanalmenge $C = \{ Nachricht_{rl}, Nachricht_{lr} \}$.

In Σ_0 gilt für den linken Agenten l die folgende Invariante:

$$Ruhe_l + gesendet_l = 1 \quad (7-2-1)$$

(d.h. der Agent l steht entweder im Zustand $Ruhe_l$ oder im Zustand $gesendet_l$).

Analog gilt für den rechten Agenten r :

$$Ruhe_r + gesendet_r = 1. \quad (7-2-2)$$

Außerdem sind die Kanalstellen $Nachricht_{r,l}$ und $Nachricht_{l,r}$ 1-beschränkt durch folgende Invarianten:

$$Ruhe_r + Nachricht_{r,l} = 1 \quad (7-2-3)$$

(d.h. entweder steht r im Zustand $Ruhe_r$ oder es gibt eine Nachricht für l ($Nachricht_{r,l}$) und

$$Ruhe_l + Nachricht_{l,r} = 1. \quad (7-2-4)$$

Daraus folgt: Das System Σ_0 ist ein Agentensystem.

Wir brauchen noch folgende Formeln:

$$Nachricht_{l,r} \leftrightarrow gesendet_l \quad (7-2-5)$$

(wegen Invarianten (7-2-1) und (7-2-4))

$$Nachricht_{r,l} \leftrightarrow gesendet_r \quad (7-2-6)$$

(wegen Invarianten (7-2-2) und (7-2-3)).

Das System Σ_0 enthält drei unverteilte Transitionen t_1 , t_2 und t_3 , ist also ein unverteilt System. Wir werden durch verteilende Verfeinerung dieses System zu einem verteilten System verfeinern. Das erhaltene System ist gerade der verteilte *crossstalk*-Algorithmus (System Σ^*).

Bei *crossstalk* ist die folgende Eigenschaft gewünscht: Ein Agent, z.B. der Agent l , soll die Kommunikation nicht beenden, bevor r die gesendete Nachricht empfangen hat. Dann soll gelten:

1. Vor einer Antwortoperation eines Agenten liegt eine Sendeoperation des anderen Agenten, d.h.

$$senden_l \blacktriangleleft antworten_r; \quad (K1)$$

$$senden_r \blacktriangleleft antworten_l. \quad (K2)$$

2. Vor einer Empfangsoperation eines Agenten liegt eine Antwortoperation des anderen Agenten, d.h.

$$antworten_r \blacktriangleleft empfangen_l; \quad (K3)$$

$$antworten_l \blacktriangleleft empfangen_r. \quad (K4)$$

Damit gilt: Zwischen zwei Sendeoperationen eines Agenten gibt es immer eine Antwortaktion des anderen Agenten und zwischen zwei Antwortoperationen eine Sendeoperation.

Das folgende Lemma besagt, dass der Anfangsalgorithmus Σ_0 die gewünschten Eigenschaften (K1)-(K4) erfüllt.

Lemma 7.2.3 Σ_0 erfüllt (K1), (K2), (K3) und (K4).

Beweis:

Zu (K1):

In Σ_0 gilt: $\text{senden}_l \blacktriangleleft t_2$ und $\text{senden}_l \blacktriangleleft t_3$.

Nur *elementare Aufgaben* (t_2) und *elementare Aufgaben* (t_3) enthalten Aufgabe antworten_r .

Daraus folgt: In Σ_0 gilt $\text{senden}_l \blacktriangleleft \text{antworten}_r$.

Zu (K2): analog zu (K1).

Zu (K3):

Nur *elementare Aufgaben* (t_2) und *elementare Aufgaben* (t_3) enthalten Aufgabe empfangen_l .

Für t_2 und t_3 gilt $\text{antworten}_r \blacktriangleleft \text{empfangen}_l$.

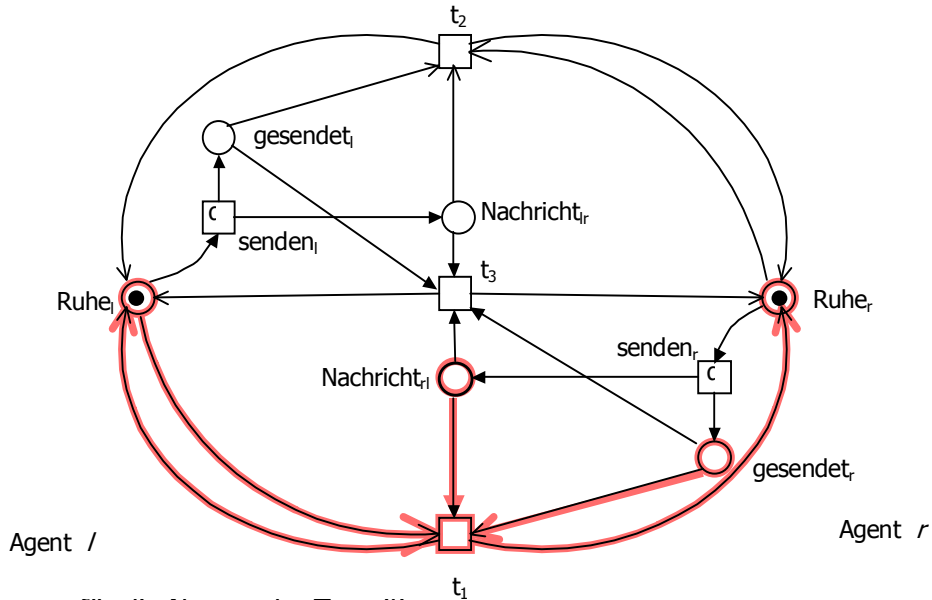
Daraus folgt: In Σ_0 gilt $\text{antworten}_r \blacktriangleleft \text{empfangen}_l$.

Zu (K4): analog zu (K3). □

Im Folgenden werden wir nacheinander die drei Transitionen t_1 , t_2 und t_3 verteilen. Da bei unseren verteilenden Verfeinerungen hier die Kausalordnungen vollständig erhalten bleiben, wissen wir nach dem Satz *Reihenfolge von Verfeinerungsschritten* im Kapitel 6, dass die Reihenfolge, in der die Transitionen verfeinert werden, den Entwurf der Verfeinerungsnetze nicht beeinflusst, d.h. wir erhalten immer dieselben Verfeinerungsnetze. Wir können deshalb den Entwurf wie folgt durchführen: Zunächst werden wir jede einzelne Transition t_1 , t_2 und t_3 in Σ_0 verfeinern und die Korrektheit der Verfeinerungen nachweisen. Dann werden wir alle drei Transitionen gleichzeitig in Σ_0 jeweils durch die entsprechenden Verfeinerungsnetze ersetzen und ein korrektes verteiltes System Σ^* erhalten.

4. Die Verteilung der unverteilt Transition t_1

Wir verfeinern jetzt die Transition t_1 , also $antworten_l$; $empfangen_r$ (das rot angestrichene Teilnetz in Abb. 7.2.6 hebt den Vor- und Nachbereich von t_1 hervor).



Abkürzungen für die Namen der Transitionen:

t_1 ? $antworten_l$; $empfangen_r$

t_2 ? $antworten_r$; $empfangen_l$

t_3 ? $\{antworten_l, antworten_r\}$; $\{empfangen_l, empfangen_r\}$

Abb. 7.2.6 System Σ_0 mit der zu verfeinernden Transition t_1

Wenn r schon eine Nachricht $Nachricht_{r,l}$ gesendet hat, und im Zustand $gesendet_r$ steht, und l im Zustand $Ruhe_l$ steht, dann werden die beiden Agenten diese Transition t_1 gemeinsam ausführen. Danach werden sie in den Zustand $Ruhe_l$ bzw. $Ruhe_r$ zurück gehen. In Abb. 7.2.7 wird t_1 isoliert dargestellt.

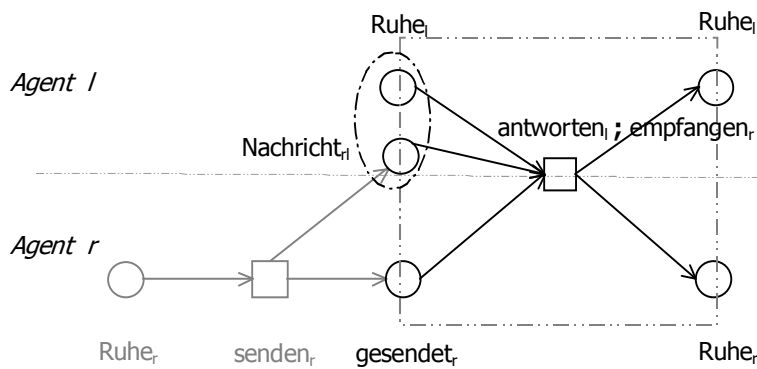


Abb. 7.2.7 Die zu verfeinernde Transition t_1

Wir werden den Entwurf der Verteilung von t_l wie folgt durchführen:
 Zunächst werden wir die Umgebung der zu verfeinernden Transition analysieren und dann die Sätze aus dem Kapitel 6 benutzen, um die Kausalitäten zu finden, die das Ersetzungsnetz erfüllen soll, damit die Blockbedingung erfüllt werden kann. Dann konstruieren wir ein Ersetzungsnetz, das diese Kausalitäten erfüllt.

In Σ_0 gilt:

$$Ruhe_l \wedge Nachricht_{r,l} \rightarrow gesendet_r$$

(wegen Formel (7-2-6), also $Nachricht_{r,l} \rightarrow gesendet_r$).

Der Agent r hat an der Stelle $gesendet_r$ eine alternative Transition t_3 .

Aus $Ruhe_l$ folgt $\neg gesendet_l$ (wegen Invariante (7-2-1), d.h. $Ruhe_l + gesendet_l = 1$)

Und $gesendet_l \in \bullet t_3$.

Daraus folgt: t_3 ist nicht aktiviert.

Nun können wir den Satz 6.3.2 (Abschwächung von Satz 6.3.1) benutzen. Laut diesem Satz garantiert ein Ersetzungsnetz die Blockbedingung, wenn es die folgenden Kausalitäten erfüllt:

- Anfangs-Transition von Agent $l \blacktriangleleft$ Anfangs-Transition von Agent r , und die Anfangs-Transition von einem Agenten kann erst schalten, wenn alle Stellen für diesen Agenten im Vorbereich von t_l schon belegt sind.
- $\bullet t_l \blacktriangleleft t_l \bullet$, d.h. ein Agent kann erst im Nachbereich von t_l Token legen, wenn sein Partner schon seine Anfangs-Transition ausgeführt hat.

Das in Abb. 7.2.8 angegebene Verfeinerungsnetz N_{t_l} erfüllt offensichtlich die oben genannten Bedingungen.

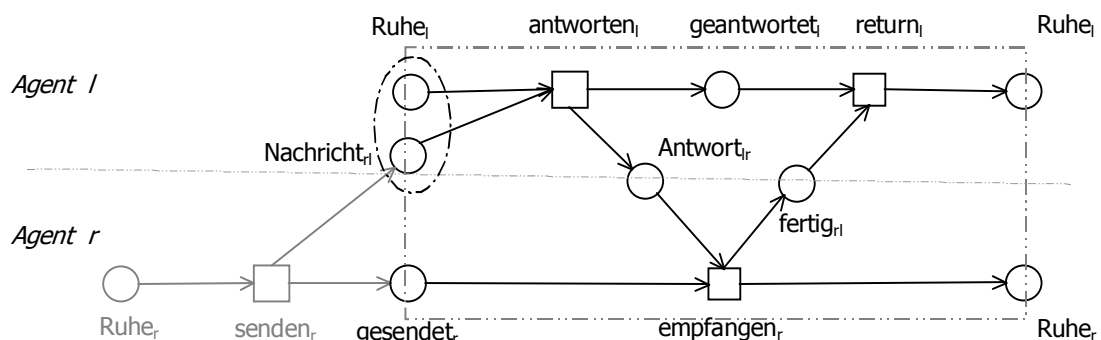


Abb. 7.2.8 Verfeinerungsnetz N_{t_l} von t_l

Wenn der Agent l im Zustand $Ruhe_l$ ist und eine Nachricht $Nachricht_{r,l}$

$Nachricht_{t_l} \triangleleft antworten_r$.

Σ_1 ist auch ein Agentensystem, weil das Verfeinerungsnetz N_{t_l} ein Agentensystem ist und die Invarianten, die im Σ_0 und (N_{t_l}, t_l^-) die Agenten l und r überdecken, kompatibel sind und damit gemeinsam Invarianten zur Überdeckung von l und r in Σ_1 ergeben.

Invarianten für die Agenten:

$Ruhe_l = 1$ kompatibel zu $Ruhe_l + geantwortet_l = 1$

$gesendet_r + Ruhe_r = 1$ kompatibel zu $Ruhe_r + gesendet_r = 1$

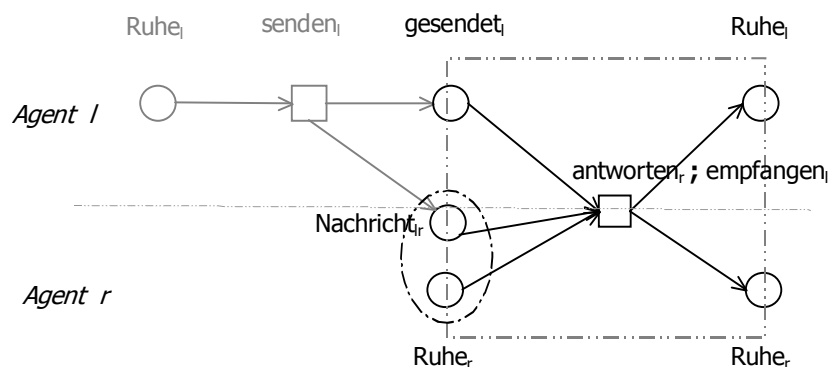
Gemeinsame Invarianten:

$Ruhe_l + geantwortet_l = 1$

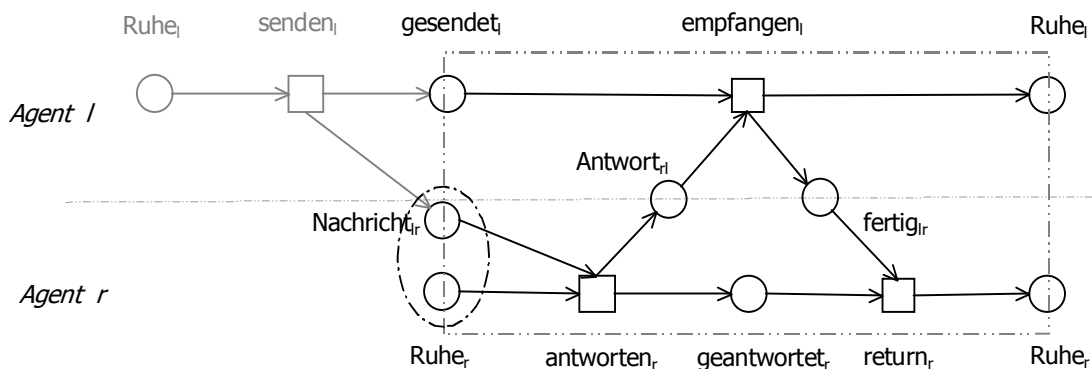
$gesendet_r + Ruhe_r = 1$.

5. Die Verteilung der unverteilter Transition t_2

Analog zu der Verteilung von t_1 verfeinern wir jetzt die Transition t_2 , also $antworten_r$; $empfangen_l$ (die rot angestrichene Transition in Abb. 7.2.10(a)). Wenn l schon an seinen Partner r eine Nachricht $Nachricht_{lr}$ gesendet hat, und im Zustand $gesendet_l$ steht, und r im Zustand $Ruhe_r$ steht, dann werden die beiden Agenten diese Transition t_2 gemeinsam ausführen. Danach werden sie in den Zustand $Ruhe_l$ bzw. $Ruhe_r$ zurück gehen.



(a) Die zu verfeinernde Transition t_2



(b) Verfeinerungsnetz N_{t_2} von t_2

Abb. 7.2.10 Verteilung von t_2

Wir werden wieder zunächst die Umgebung der zu verfeinernden Transition analysieren, und dann die Sätze aus dem Kapitel 6 verwenden, um ein geeignetes korrektes Ersetzungsnetz zu finden.

In Σ_0 gilt:

$Ruhe_r \wedge Nachricht_{lr} \rightarrow gesendet_l$ (wegen $Nachricht_{lr} \rightarrow gesendet_l$).

Die alternative Transition t_3 an der Stelle $gesendet_l$ ist nicht aktiviert,

weil $gesendet_r \in \bullet t_3$ aber $gesendet_r$ gilt nicht (da $Ruhe_r$ gilt).

Nach dem Satz 6.3.2 (Abschwächung von Satz 6.3.1) garantiert das in

Abb. 7.2.10(b) angegebene Verfeinerungsnetz N_{t_2} die Blockbedingung.

Wenn der Agent r im Zustand $Ruhe_r$ ist und eine Nachricht $Nachricht_{r,l}$ von seinem Partner l bekommt, dann sendet er ein Signal $Antwort_{r,l}$ (Transition $antworten_r$). Wenn l dieses Signal $Antwort_{r,l}$ bekommt, dann geht l in den Zustand $Ruhe_l$ zurück und sendet dabei an r ein Signal $fertig_{lr}$ (um ihn mitzuteilen, dass er schon mit dem Empfangen des Signals $Antwort_{r,l}$ fertig ist). Wenn r das Signal $fertig_{lr}$ bekommt, geht er in den Zustand $Ruhe_r$ zurück (Transition $return_r$).

In Abb. 7.2.11 (b) ist das erhaltene System Σ_2 nach der Verteilung von t_2 in Σ_0 dargestellt.

Damit erhalten wir das folgende Lemma.

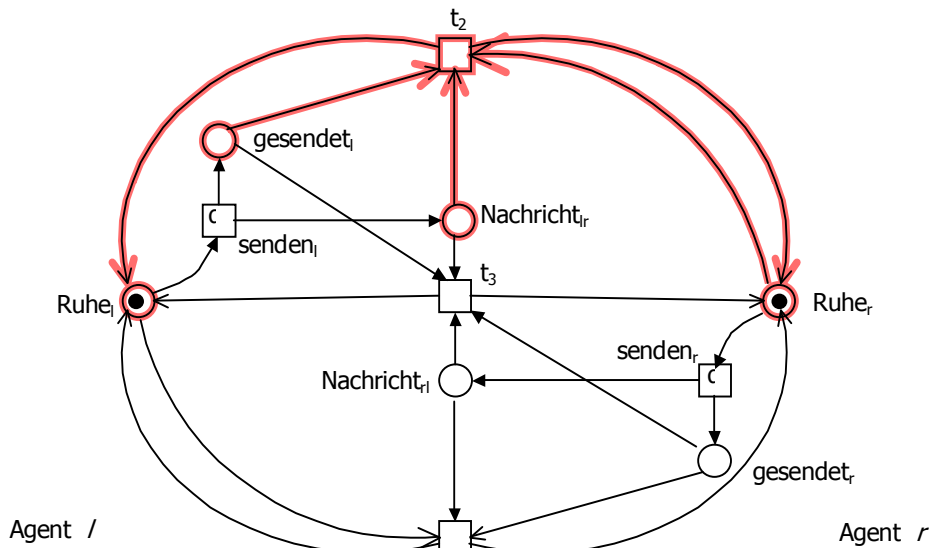
Lemma 7.2.6

Verfeinerungsschritt ($\Sigma_0 \rightarrow \Sigma_2$) mit $\Sigma_2 = \Sigma_0(N_{t_2} \setminus t_2)$ ist Halbordnung vollständig erhaltend.

Offensichtlich gilt die folgende Aussage.

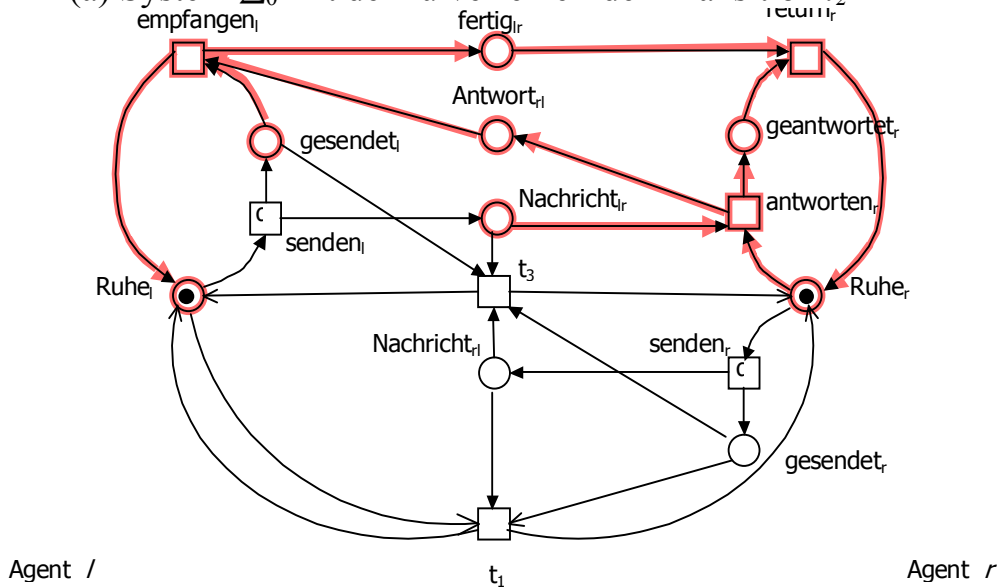
Lemma 7.2.7

N_{t_2} ist konform zur Aufgabenspezifikation t_2 und in (N_{t_2}, t_2^-) gilt $Nachricht_{r,l} \blacktriangleleft antworten_l$.



Abkürzungen für die Namen der Transitionen:
 t_1 ? $\{antworten_l, empfangen_r\}$
 t_2 ? $\{antworten_r, empfangen_l\}$
 t_3 ? $\{antworten_l, antworten_r\}; \{empfangen_l, empfangen_r\}$

(a) System Σ_0 mit der zu verfeinernden Transition t_2

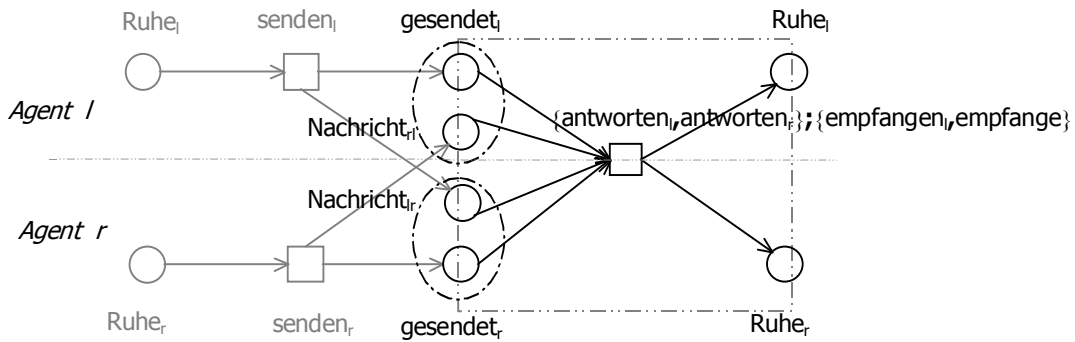


Abkürzungen für die Namen der Transitionen:
 t_1 ? $\{antworten_l, empfangen_r\}$
 t_2 ? $\{antworten_l, antworten_r\}; \{empfangen_l, empfangen_r\}$

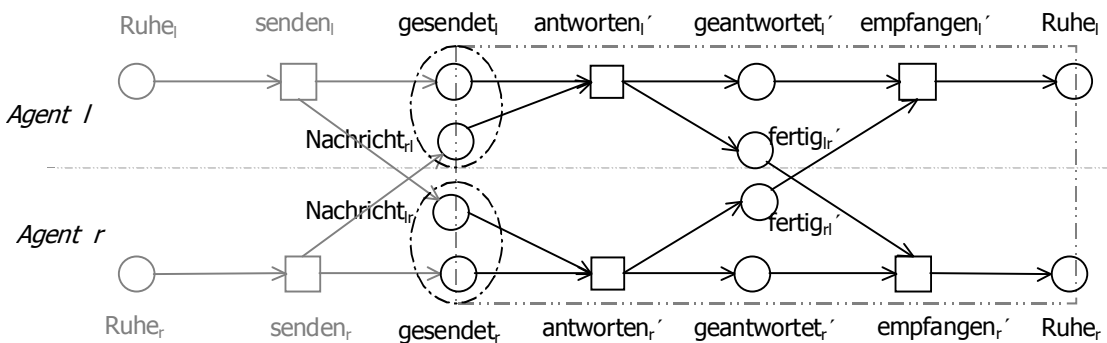
(b) Das erhaltene System Σ_2 nach der Verteilung von t_2 in Σ_0
 Abb. 7.2.11 Verteilung von t_2

6. Die Verteilung der unverteilter Transition t_3

Im Folgenden verfeinern wir die Transition t_3 , also $\{antworten_l, antworten_r\}; \{empfangen_l, empfangen_r\}$ (die rot angestrichene Transition in Abb. 7.2.13(a)). Wenn der Agent l schon an seinen Partner eine Nachricht $Nachricht_{lr}$ gesendet hat, und im Zustand $gesendet_l$ steht, und auch der Agent r schon an seinen Partner eine Nachricht $Nachricht_{rl}$ gesendet hat, und im Zustand $gesendet_r$ steht, dann werden die beiden Agenten diese Transition t_3 gemeinsam ausführen. Danach werden sie in den Zustand $Ruhe_l$ bzw. $Ruhe_r$ zurück gehen.



(a) Die zu verfeinernde Transition t_3



(b) Verfeinerungsnetz N_{t_3} von t_3

Abb. 7.2.12 Verteilung von t_3

Wir werden wieder zunächst die Umgebung der zu verfeinernden Transition analysieren, und dann die Sätze aus dem Kapitel 6 verwenden, um ein korrektes Ersetzungsnetz zu entwerfen.

In Σ_0 gilt:

$$gesendet_l \wedge Nachricht_{rl} \rightarrow gesendet_r \wedge Nachricht_r$$

(wegen $gesendet_l \rightarrow Nachricht_{lr}$ und $Nachricht_{rl} \rightarrow gesendet_r$).

Umgekehrt gilt auch, d.h.:

$$gesendet_r \wedge Nachricht_{lr} \rightarrow gesendet_l \wedge Nachricht_{rl}.$$

Die alternative Transition t_2 von Agent l an der Stelle $gesendet_l$ ist nicht aktiviert, weil $Ruhe_r \in \bullet t_2$ aber $Ruhe_r$ gilt nicht (da $gesendet_r$ gilt).

Die andere alternative Transition t_1 von Agent l an der Stelle $Nachricht_{rl}$ ist auch nicht aktiviert, weil $Ruhe_l \in \bullet t_1$ aber $Ruhe_l$ gilt nicht (da $gesendet_l$ gilt).

Analog sind die beiden alternativen Transitionen t_1 und t_2 von Agent r jeweils an der Stelle $gesendet_r$ und der Stelle $Nachricht_{lr}$ nicht aktiviert.

Nach dem Satz 6.2.6 (Abschwächung von Satz 6.2.4) garantiert das in Abb. 7.2.12(b) angegebene Verfeinerungsnetz N_{t_3} die Blockbedingung.

Wenn der Agent l im Zustand $gesendet_l$ eine Nachricht $Nachricht_{rl}$ von seinem Partner r bekommt, dann wird er die Transition $antworten_l'$ ausführen. Dabei wird er ein Signal $fertig_{lr}'$ senden und in den Zustand $geantwortet_l'$ übergehen.

Und wenn der Agent r im Zustand $gesendet_r$ eine Nachricht $Nachricht_{lr}$ von seinem Partner l bekommt, dann wird r ebenfalls die Transition $antworten_r'$ ausführen. Dabei wird er ein Signal $fertig_{rl}'$ senden und in den Zustand $geantwortet_r'$ übergehen.

Wenn der Agent l im Zustand $geantwortet_l'$ das Signal $fertig_{rl}'$ von seinem Partner r bekommt, geht er in den Zustand $Ruhe_l$ zurück. Der Agent r geht ebenfalls auch in den Zustand $Ruhe_r$ zurück, wenn er von seinem Partner das Signal $fertig_{lr}'$ bekommt.

In Abb. 7.2.13 (b) ist das erhaltene System Σ_3 nach der Verteilung von t_3 in Σ_0 dargestellt.

Damit erhalten wir Lemma 7.2.8.

Lemma 7.2.8

Verfeinerungsschritt ($\Sigma_0 \rightarrow \Sigma_3$) mit $\Sigma_3 = \Sigma_0(N_{t_3} \setminus t_3)$ ist Halbordnung vollständig erhaltend.

Offensichtlich gilt die folgende Aussage.

Lemma 7.2.9

N_{t_3} ist konform zur Aufgabenspezifikation t_3 und in (N_{t_3}, t_3^-) gelten $Nachricht_{lr} \blacktriangleleft antworten_r'$ und $Nachricht_{rl} \blacktriangleleft antworten_l'$.

Mit dem Satz aus Abschnitt 6.2 können wir zeigen, dass beim Verfeinerungsschritt $(\Sigma_0 \rightarrow \Sigma_3)$ die Voraussetzungen für die Umgebung für den Nachweis der Blockbedingung bei verteilender Verfeinerung von t_1 und von t_2 erhalten bleiben.

Lemma 7.2.10

i. Beim Verfeinerungsschritt $(\Sigma_0 \rightarrow \Sigma_3)$ bleiben folgende Eigenschaften erhalten:

Ruhe_l \wedge Nachricht_{r,l} \triangleright gesendet_r;

Ruhe_r \wedge Nachricht_{l,r} \triangleright gesendet_l.

ii. In Σ_3 gibt es

- keine neue aktivierte Transition zu t_1 und
- auch keine neue aktivierte Transition zu t_2 .

Beweis:

i. In Σ_0 gilt offensichtlich: (wegen Transition *senden_l*)

$$Ruhe_l \rightarrow \neg \text{Nachricht}_{l,r}. \tag{1}$$

Dann gilt in Σ_0 auch:

$$Ruhe_l \wedge \text{Nachricht}_{r,l} \rightarrow \neg \text{Nachricht}_{l,r}.$$

Bei $(\Sigma_0 \rightarrow \Sigma_3)$ bleibt die folgende Eigenschaft erhalten: (wegen Satz 6.2.8)

$$Ruhe_l \wedge \text{Nachricht}_{r,l} \triangleright \text{gesendet}_r.$$

Analog bleibt bei $(\Sigma_0 \rightarrow \Sigma_3)$ die folgende Eigenschaft erhalten:

$$Ruhe_r \wedge \text{Nachricht}_{l,r} \triangleright \text{gesendet}_l.$$

ii. Bei $(\Sigma_0 \rightarrow \Sigma_3)$ bleibt (1) erhalten. (wegen Bem. 3.5.3)

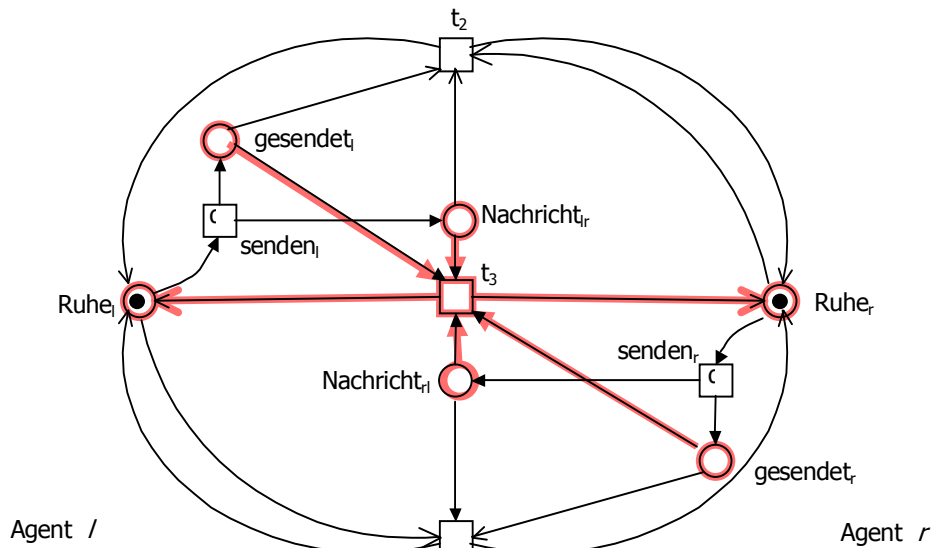
Also, in Σ_3 gilt (1) weiter.

In Σ_3 gilt:

Wenn t_1 aktiviert ist, dann ist die Transition *antworten_r'* (die Anfangstransition vom Agenten r im Ersetzungsnetz N_{t_3}) nicht aktiviert.

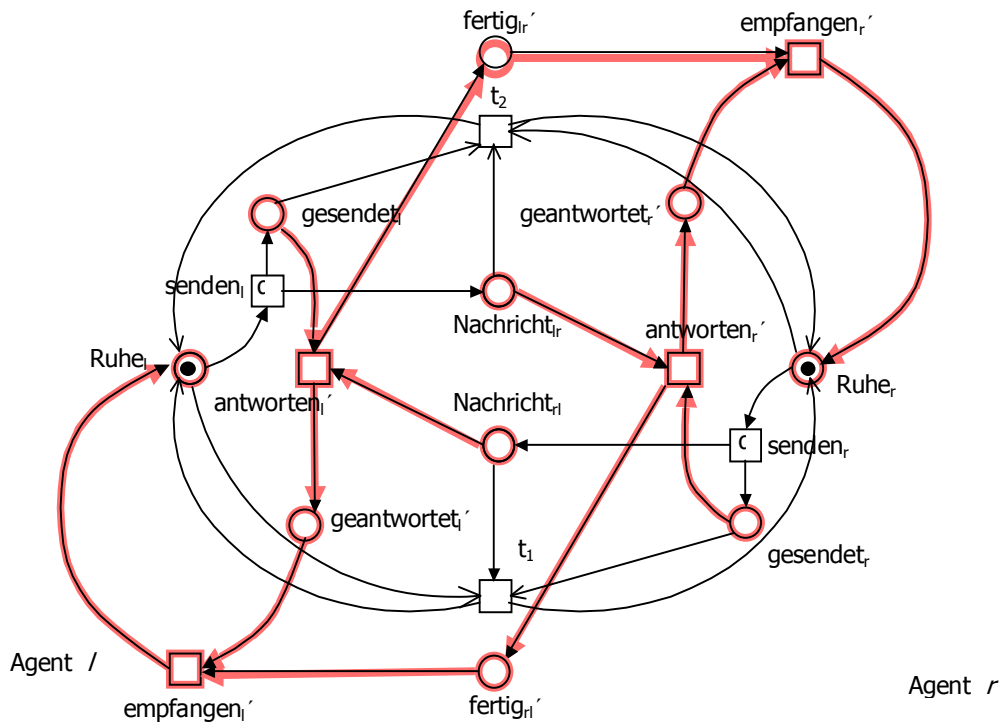
Also, in Σ_3 gibt es keine neue aktivierte Transition zu t_1 .

Analog gibt es in Σ_3 keine neue aktivierte Transition zu t_2 . □



Abkürzungen für die Namen der Transitionen:
 $t_1^?$ $\{antworten_l, empfangen_r\}$
 $t_2^?$ $\{antworten_r, empfangen_l\}$
 $t_3^?$ $\{antworten_l, antworten_r\}; \{empfangen_l, empfangen_r\}$

(a) System Σ_0 mit der zu verfeinernden Transition t_3



Abkürzungen für die Namen der Transitionen:
 $t_1^?$ $\{antworten_l, empfangen_r\}$
 $t_2^?$ $\{antworten_r, empfangen_l\}$

(b) Das erhaltene System Σ_3 nach der Verteilung von t_3 in Σ_0
 Abb. 7.2.13 Verteilung von t_3

7. Der verteilte Zielalgorithmus

Wir ersetzen jetzt alle drei Transitionen t_1 , t_2 und t_3 in Σ_0 jeweils durch ihre Verfeinerungsnetze N_{t_1} , N_{t_2} und N_{t_3} . Das erhaltene System ist Σ_*' (Abb. 7.2.14).

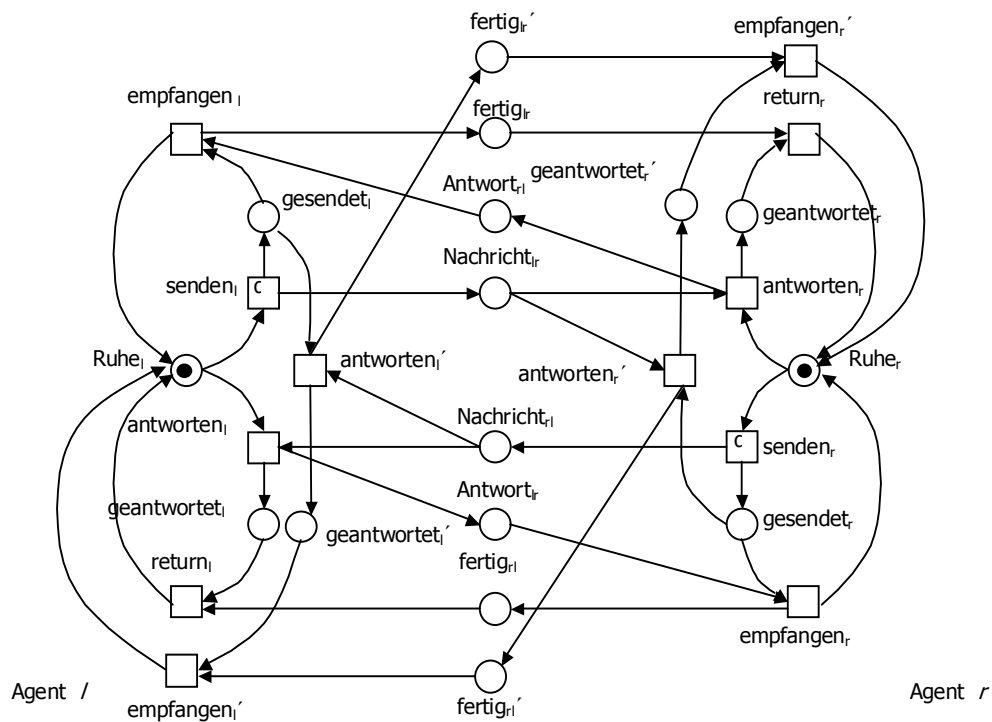


Abb. 7.2.14 System Σ_*'

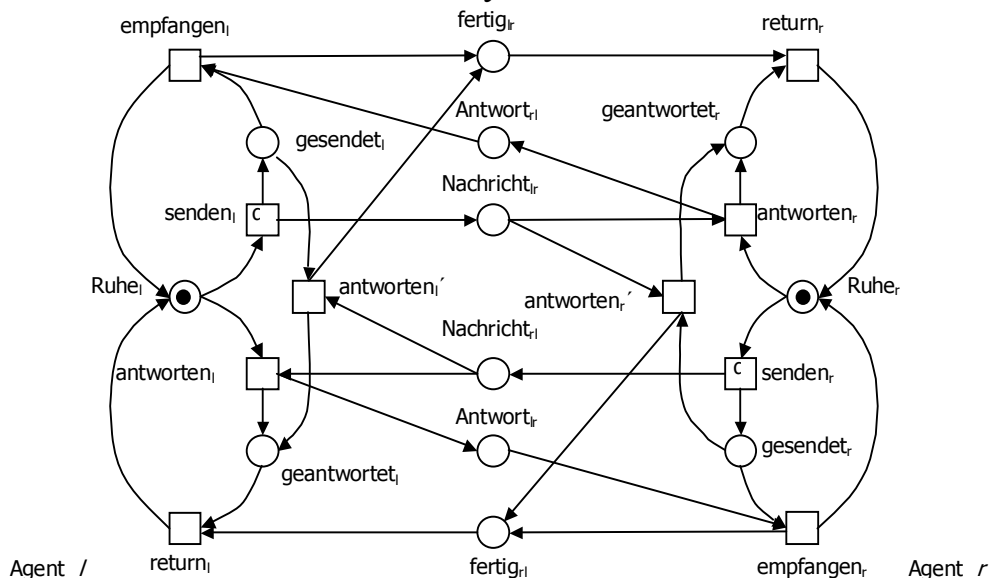


Abb. 7.2.15 Zielalgorithmus Σ_* – verteilter *crosstalk*-Algorithmus

Lemma 7.2.11

$\Sigma_0 \rightarrow \Sigma_3 \rightarrow \Sigma_{1'} \rightarrow \Sigma_{2'}$ mit $\Sigma_3 = \Sigma_0(N_{t_3} \setminus t_3)$, $\Sigma_{1'} = \Sigma_3(N_{t_1} \setminus t_1)$, $\Sigma_{2'} = \Sigma_{1'}(N_{t_2} \setminus t_2)$ ist Halbordnung vollständig erhaltend.

Beweis: Folgt direkt aus Satz 6.5.4 wegen Lemma 7.2.4, Lemma 7.2.6, Lemma 7.2.8 und Lemma 7.2.10. \square

Satz 7.2.12 $\Sigma_{*'}'$ erfüllt (K1), (K2), (K3) und (K4).

Beweis:

Wir zeigen hier, $\Sigma_{*'}'$ erfüllt (K1) und (K3). Der Beweis für (K2) und (K4) ist analog.

Wegen Lemma 7.2.11 gilt:

Jeder Verfeinerungsschritt von $\Sigma_0 \rightarrow \Sigma_3 \rightarrow \Sigma_{1'} \rightarrow \Sigma_{2'}$ mit $\Sigma_3 = \Sigma_0(N_{t_3} \setminus t_3)$, $\Sigma_{1'} = \Sigma_3(N_{t_1} \setminus t_1)$, $\Sigma_{2'} = \Sigma_{1'}(N_{t_2} \setminus t_2)$ und $\Sigma_{2'} = \Sigma_{*'}'$ ist Halbordnung vollständig erhaltend. (1)

Wegen Lemma 7.2.3 gelten (K1) und (K3) in Σ_0 .

Wir brauchen nur zu zeigen, dass bei jedem Verfeinerungsschritt von $\Sigma_0 \rightarrow \Sigma_3 \rightarrow \Sigma_{1'} \rightarrow \Sigma_{2'}$ (K1) und (K3) erhalten bleiben.

Zu $(\Sigma_0 \rightarrow \Sigma_3)$ mit $\Sigma_3 = \Sigma_0(N_{t_3} \setminus t_3)$:

In Σ_0 gilt: $\text{senden}_l \blacktriangleleft \text{Nachricht}_{lr}$. (2)

In (N_{t_3}, t_3^-) gilt $\text{Nachricht}_{lr} \blacktriangleleft \text{antworten}_r'$. (wegen Lemma 7.2.9)

Daraus folgt: (wegen Satz 7.2.2(i))

Beim Verfeinerungsschritt $(\Sigma_0 \rightarrow \Sigma_3)$ bleibt $\text{senden}_l \blacktriangleleft \text{antworten}_r$ erhalten.

Es gilt: N_{t_3} ist konform zu t_3 . (wegen Lemma 7.2.9)

Daraus folgt: (wegen Satz 7.2.2(ii))

Beim Verfeinerungsschritt $(\Sigma_0 \rightarrow \Sigma_3)$ bleibt $\text{antworten}_r \blacktriangleleft \text{empfangen}_l$ erhalten.

Also, bei $(\Sigma_0 \rightarrow \Sigma_3)$ bleiben (K1) und (K3) erhalten.

Zu $(\Sigma_3 \rightarrow \Sigma_{1'})$ mit $\Sigma_{1'} = \Sigma_3(N_{t_1} \setminus t_1)$:

Es gilt: $\text{antworten}_r \notin \text{elementare Aufgaben}(t_1)$.

Wegen (1) gilt: (wegen Satz 7.2.2(iii))

Beim Verfeinerungsschritt $(\Sigma_3 \rightarrow \Sigma_{1'})$ bleibt $\text{senden}_l \blacktriangleleft \text{antworten}_r$ erhalten.

Wegen $\text{antworten}_r \notin \text{elementare Aufgaben}(t_1)$ und $\text{empfangen}_l \notin$

elementare Aufgaben(t_1) gilt: (wegen Satz 7.2.2(iv))

Beim Verfeinerungsschritt ($\Sigma_3 \rightarrow \Sigma_1'$) bleibt *antworten_r* ◀ *empfangen_l* erhalten.

Also, bei ($\Sigma_3 \rightarrow \Sigma_1'$) bleiben (K1) und (K3) erhalten.

Zu ($\Sigma_1' \rightarrow \Sigma_2'$) mit $\Sigma_2' = \Sigma_1'(N_{t_2} \setminus t_2)$:

Wegen (1) gilt: (wegen Satz 3.5.4)

Bei jedem Verfeinerungsschritt von $\Sigma_0 \rightarrow \Sigma_3 \rightarrow \Sigma_1'$ bleibt *senden_l* ◀ *Nachricht_{l,r}* erhalten.

Aus (2) folgt:

In Σ_1' gilt *senden_l* ◀ *Nachricht_{l,r}*.

In (N_{t_2}, t_2^-) gilt *Nachricht_{l,r}* ◀ *antworten_r*. (wegen Lemma 7.2.7)

Daraus folgt: (wegen Satz 7.2.2(i))

Beim Verfeinerungsschritt ($\Sigma_1' \rightarrow \Sigma_2'$) bleibt *senden_l* ◀ *antworten_r* erhalten.

Es gilt: N_{t_2} ist konform zu t_2 . (wegen Lemma 7.2.7)

Daraus folgt: (wegen Satz 7.2.2(ii))

Beim Verfeinerungsschritt ($\Sigma_1' \rightarrow \Sigma_2'$) bleibt *antworten_r* ◀ *empfangen_l* erhalten.

Also, bei ($\Sigma_1' \rightarrow \Sigma_2'$) bleiben (K1) und (K3) erhalten.

Daraus folgt: In Σ_{*}' gelten (K1) und (K3). □

Wir machen von diesem System Σ_{*}' aus eine äquivalente System-Transformation und erreichen dann schon den Zielalgorithmus Σ_{*} - *crossstalk*-Algorithmus (Abb. 7.2.15). Z. B. haben *empfangen_r'* und *return_r* die gleiche Wirkung. Bei ihrer Identifikation fallen die Stellen *fertig_{l,r}'* und *fertig_{l,r}* und auch die Stellen *geantwortet_r* und *geantwortet_r'* zusammen. Es gibt für derartige Vereinfachungen, bei denen zwei Aktionen mit der gleichen Wirkung identifiziert werden, gut bekannte Algorithmen.

Der entworfene Algorithmus erfüllt die geforderten Eigenschaften. Es ist verteilt, lebendig und es gelten die Sicherheitseigenschaften: Zwischen zwei Sendeoperationen eines Agenten gibt es immer eine Antwortaktion des anderen Agenten und zwischen zwei Antwortoperationen eine Sendeoperation.

Literaturverzeichnis

- AL91 M. Abadi, L. Lamport: The existence of refinement mapping. Theoretical Computer Science. [1991](#), [82](#), S.[253-284](#) .
- Bac90 R.-J. Back: Refinement Calculus, Part II: Paralle and Reactive Programs. Hrsg.: J. W. de Bakker, W. P. de Roever: Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness (LNCS). Band [430](#). Springer-Verlag, [1990](#) S.[67-93](#), ISBN: 978-3-540-52559-2.
- BDE93 E. Best, R. R. Devillers, J. Esparza: General Refinement and Recursion Operators for the Petri Box Calculus. Hrsg.: P. Enjalbert, A. Finkel, K. W. Wagner: STACS93, 10th Annual Symposium on Theoretical Aspects of Computer Science (LNCS). Band [665](#). Springer-Verlag, [1993](#) S.[130-140](#), ISBN: 978-3-540-56503-1.
- BGV90 W. Brauer, R. Gold, W. Vogler: A survey of behaviour and equivalence preserving refinements of Petri nets. Hrsg.: G. Rozenberg: Advances in Petri Nets (LNCS). Band [483](#). Springer-Verlag, [1990](#) S.[1-46](#), ISBN: 978-3-540-53863-9.
- Bro97 M. Broy: Compositional refinement of interactive systems. Journal of the ACM. [1997](#), [6 \(Volume 44\)](#), S.[850-891](#).
- BS96 R. J. R. Back, K. Sere: Superposition refinement of reactive systems. Formal Aspects of Computing. [1996](#), [8](#), S.[324-346](#) .
- CM88 K. M. Chandy, J. Misra: Paralle Program Design: A Foundation. Addison-Wesley, [1988](#) .
- Des97 J. Desel: How distributed algorithms play the token game. Hrsg.: C. Fesksa, M. Jantzen, R. Valk: Foundations of Computer Science -

- Potential, Theory, Cognition (LNCS). Band 1337. Springer-Verlag, 1997 S.297-306, ISBN: 978-3-540-63746-2.
- dRe98 W.-P. de Roever, K. Engelhardt: Data Refinement: Model-oriented Proof Methods and their Comparison. Cambridge Tracts in Theoretical Computer Science. 1998, 47.
- HHS87 C. A. R. Hoare, J. He, J. W. Sanders: Prespecification in data refinement. Information Processing Letters. 1987, 25, S.71-76.
- Ho99 C. A. R. Hoare: Top-Down and Bottom-Up and Meeting in the Middle. Hrsg.: E. R. Olderog, B. Steffen: Correct System Design, Recent Insight and Advances. Band 1710. Springer-Verlag, 1999 S.3-28, ISBN: 978-3-540-66624-0.
- Kin95 E. Kindler: Modularer Entwurf verteilter Systeme mit Petrinetzen. Dissertation. TU München, Bertz Verlag, 1995
- LL90 L. Lamport, N. Lynch: Distributed computing: Models and methods. Hrsg.: J. v. Leeuwen: Handbook of Theoretical Computer Science, volume B: Formal Models and Semantics . Elsevier Science Publishers B. V., 1990 S.1158-1199 .
- Lyn96 N. A. Lynch: Distributed Algorithms. Morgan Kaufmann, 1996
- Mat89 F. Mattern: Verteilte Basisalgorithmen. Informatik-Fachberichte (Springer-Verlag). 1989, 226.
- ME92 C. Morgan, T. Vickers: On the Refinement Calculus. Springer-Verlag, 1992
- MP92 Z. Manna, A. Pnueli: The Temporal Logic of Reactive and Concurrent Systems, Specification. Springer-Verlag, 1992 ISBN: 978-0-387-97664-8.

- MP92 **Z. Manna, A. Pnueli**: The Temporal Logic of Reactive and Concurrent Systems, Verification. Springer-Verlag, 1995 ISBN: 978-0-387-94459-3.
- Peu01 **S. Peuker**: Halbordnungsbasierte Verfeinerung zur Verifikation verteilter Algorithmen. Dissertation. HU Berlin, 2001
- PU03 **J. Padberg, M. Urbasek**: Rule-Based Refinement of Petri Nets: A Survey. Hrsg.: H. Ehrig, W. Reisig, G. Rozenberg, H. Weber : Petri Net Technology for Communication-Based Systems - Advances in Petri Nets (LNCS). Band 2472. Springer-Verlag, 2003 S.161-196, ISBN: 978-3-540-20538-8.
- Rei85 **W. Reisig**: Petri Nets. Springer-Verlag, 1985
- Rei87 **W. Reisig**: A Strong Part of Concurrency. Hrsg.: G. Rozenberg: Advances in Petri Nets (LNCS). Band 266. Springer-Verlag, 1987 S.238-272, ISBN: 978-3-540-18086-9.
- Rei95 **W. Reisig**: Petri Net Models of Distributed Algorithms. Hrsg.: J. v. Leeuwen: Computer Science Today. Recent Trends and Developments (LNCS). Band 1000. Springer-Verlag, 1995 S.441-454.
- Rei98 **W. Reisig**: Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets. Springer-Verlag, 1998 ISBN: 978-3-540-62752-4.
- RK97 **W. Reisig, E. Kindler**: Verification of Distributed Algorithms with Algebraic Petri Nets. Hrsg.: C. Fesksa, M. Jantzen, R. Valk: Foundations of Computer Science - Potential, Theory, Cognition. Band 1337. Springer-Verlag, 1997 S.297-306, ISBN: 978-3-540-63746-2.

- SL90 **A. U. Shankar, S. S. Lam**: Construction of Network Protocols by Stepwise Refinement. Hrsg.: J. W. de Bakker, W. P. de Roever: Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness (LNCS). Band 430. Springer-Verlag, 1990 S.669-695, ISBN: 978-3-540-52559-2.
- Tel91 **G. Tel**: Topics in Distributed Algorithms. Cambridge University Press, 1991
- Val79 **R. Valette**: Analysis of Petri Nets by stepwise refinement. Journal of Computer and System Sciences. 1979, 18, S.35-46.
- vGG01 **R. J. van Glabbeek, U. Goltz**: Refinement of actions and equivalence notions for concurrent systems . Acta Informatica. 2001, 4 (Volume 37), S.229-327.
- vGG90 **R. J. van Glabbeek, U. Goltz**: Refinement of actions in causality based models. Hrsg.: J. W. de Bakker, W. P. de Roever: Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness (LNCS). Band 430. Springer-Verlag, 1990 S.267-300, ISBN: 978-3-540-52559-2.
- Vog87 **W. Vogler**: Behaviour Preserving Refinement of Petri Nets. Hrsg.: G. Tinhofer, G. Schmidt: Graphtheoretic Concepts in Computer Science (LNCS). Band 246. Springer-Verlag, 1987 S.82-93, ISBN: 978-3-540-17218-5.
- Vog92 **W. Vogler**: Modular Construction and Partial Order Semantics of Petri Nets (LNCS). Band 625. Springer-Verlag, 1992 ISBN: 978-3-540-55767-8.
- Wal95 **R. Walter**: Petrinetzmodelle verteilter Algorithmen. Beweistechnik und Intuition. Dissertation. HU Berlin, Bertz Verlag, 1995

WWV+97 M. Weber, R. Walter, H. Völzer, T. Vesper, W. Reisig, S. Peuker,
E. Kindler, J. Freiheit, J. Desel: DAWN: Petrinetzmodelle zur
Verifikation verteilter Algorithmen. Informatik-Berichte (HU
Berlin). 1997, 88 .

Danksagung

Ich bedanke mich herzlich bei Herrn Prof. Wolfgang Reisig für die Betreuung und Begutachtung der Arbeit, sowie besonders dafür, dass ich an seinem Lehrstuhl unter hervorragenden Bedingungen arbeiten durfte.

Besonders herzlich bedanken möchte ich mich bei meinem Betreuer, Herrn Prof. Bodo Hohberg, für sehr viele anregende Diskussionen und seine umfassende Unterstützung bei der Fertigstellung dieser Arbeit.

Bedanke ich mich auch bei unserer Sekretärin Birgit Heene und den anderen Kollegen des Lehrstuhls für gute Arbeitsatmosphäre und interessante Diskussionen bei der Kaffeerunde. Mein Dank gilt auch Birgit Schiefner für die Unterstützung bei technischen Problemen.

Ich danke dem chinesischen Erziehungsministerium und der Hanns-Seidel-Stiftung für finanzielle Unterstützung.

Ganz besonders möchte ich meinen Freunden und meiner Familie danken. Ohne den Rückhalt und die Motivation, die sie mir immer wieder gegeben haben, hätte die Arbeit niemals entstehen können.

Lebenslauf

Name Bixia Wu
Geburtsdatum 18. Juli 1966
Geburtsort Provinz Fujian, China
Staatsangehörigkeit VR China

Schulbildung

Sept.1970-Juli 1975 Grundschule in Sanming, Provinz Fujian, China
Sept.1975-Juli 1980 Mittelschule in Sanming, Provinz Fujian, China

Studium

Sept.1980-Juli 1984 Studium am Institut für Automatisierung der
Technischen Universität für Nationale
Verteidigung, Changsha, China
Juli 1984 Abschluß: "Bachelor of Engineering"
Sept.1984-Feb. 1987 Aspirantin am Institut für Automatisierung der
Universität Xiamen, China
Juni1987 Abschluß: "Master of Science"

Wissenschaftlicher Werdegang

März 1987-Okt.1991 Assistentin am Institut für Automatisierung der
Technischen Universität für Nationale
Verteidigung, Changsha, China
Nov.1991-Sept.1992 Assistentin am Institut für Informatik der
Universität Xiamen, China
Okt.1992-Feb.1998 Dozentin am Institut für Informatik der
Universität Xiamen, China
Mai1998-Sept. 2005 Studienaufenthalt am Lehrstuhl Professor Reisig,
Institut für Informatik der HU (teilweise
Stipendiatin der Hanns-Seidel-Stiftung)

Berlin, den 28. März 2007

Bixia Wu

Erklärung

Hiermit erkläre ich, dass

- ich die vorliegende Dissertationsschrift „Entwurf und Verifikation von Petrinetzmodellen verteilter Algorithmen durch Verfeinerung unverteilter Algorithmen“ selbstständig und ohne unerlaubte Hilfe angefertigt habe;
- ich mich nicht anderwärts um einen Doktorgrad beworben habe oder einen solchen besitze;
- mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät II der Humboldt-Universität zu Berlin vom 17.01.2005 (zuletzt geändert am 13.02.2006) bekannt ist.

Berlin, den 28. März 2007

Bixia Wu