Ernst-Moritz-Arndt-Universität Greifswald

Institut für Mathematik und Informatik

# Ein Kalkül paralleler Prozesse, Bisimulationen und Korrektheit

Wissenschaftliche Arbeit zur Erlangung
des akademischen Grades

„Diplommathematiker“

am Institut für Mathematik und Informatik
der Ernst-Moritz-Arndt-Universität Greifswald

vorgelegt von

Sebastian Wilhelmi

im Auftrag von

Prof. Dr. Christoph Bandt

Greifswald, den 18. April 1997

# Inhaltsverzeichnis

## Danksagung

Im danke K.V.S. Prasad von der Universität Göteborg für die Entwicklung des wunderschönen Kalküls (T)CBS und seine sehr gute Vorlesung über Protokolle, die diese Arbeit erst möglich machten. Außerdem danke ich meinem Diplomarbeitsbetreuer Prof. Bandt für die konstruktive Mithilfe an dieser Arbeit und dem Zweitgutachter Prof. Voelkel, ohne dessen Hinweise und Anregungen diese Arbeit sehr viel weniger gut lesbar gewesen wäre.

# 1 Einleitung

## 1.1 Kurzüberblick

Eine gegenwärtig explosionsartig an Bedeutung und Verbreitung gewinnende Technologie ist die der weltweiten und lokalen Vernetzung von Rechnern. Bekanntestes Beispiel dafür ist sicherlich das Internet. Für herkömmliche sequentielle Algorithmen bilden klassische Modelle der theoretischen Informatik wie z.B. endliche Automaten und die Turingmaschine eine gut ausgebaute Basis für formale Schlußmethoden.

An der Kommunikation in Rechnernetzen sind aber mehrere Rechner beteiligt, die mit Hilfe eines Protokolls miteinander kommunizieren. Bekannte Beispiele für solche Protokolle sind das HTTP (**H**yper**t**ext **T**ransfer **P**rotocol), das FTP (**F**ile **T**ransfer **P**rotocol) und das SMTP (**S**imple **M**ail **T**ransfer **P**rotocol), die alle auf die Internetprotokollfamilie TCP/UDP/IP (**T**ransmission **C**ontrol **P**rotocol/**U**ser **D**atagram **P**rotocol/**I**nternet **P**rotocol) aufsetzen.

Für diese spezielle Art von parallelen Algorithmen steckt die Erforschung der mathematischen Grundlagen trotz einiger Fortschritte in den letzten Jahren noch in den Anfängen. So ist z.B. nicht bewiesen, daß das Internet Protokoll in jeder Situation funktioniert, daß es korrekt ist.

Für die Modellierung paralleler Probleme hat sich die Verwendung paralleler Kalküle bewährt. Ein solcher paralleler Kalkül ist TCBS (**T**imed **C**alculus of **B**roadcasting **S**ystems) von K.V.S. Prasad. Auf Grundlage dieses Kalküls beschreibe ich in dieser Arbeit einen Weg zum Nachweis der Korrektheit von Protokollen und zeige als Beispiel dazu, wie das Alternatig-Bit-Protokoll, ein einfaches Protokoll, als korrekt nachgewiesen wird.

## 1.2 Der Nutzen paralleler Kalküle

In den letzten Jahren hat die Erforschung des Verhaltens verteilter und paralleler Systeme weltweit an Bedeutung gewonnen. Mit der immer stärkeren Vernetzung der Computer und dem Vordringen von Mikroprozessoren in weitere Felder des alltäglichen Lebens erhält das Verständnis der Kommunikation zwischen verschiedenen Rechnern, aber auch zwischen verschiedenen Programmen auf demselben Rechner eine immer höhere Priorität. Es geht dabei um Fragen der Zuverlässigkeit von Systemen. Um diese zu beantworten, gibt es die Möglichkeit der Überprüfung jedes einzelnen Systems mit wenig mehr als Erfahrung, gesundem Menschenverstand und einiger weniger Regeln für die Verhinderung von unerwünschten, immer wieder auftretenden Problemen wie z.B. dem Deadlock, dem gegenseitigen Blockieren von verschiedenen Prozessen.

Da die heute auftretenden parallelen und verteilten Systeme aber dazu tendieren, immer komplexer zu werden, gewinnt der formale beweistechnische Ansatz für die Verifikation von Programmen an Bedeutung. Dazu ist natürlich eine Abstraktion der Realität erforderlich. Diese besteht in der Definition eines Kalküls, welcher nur noch ganz wenige, für die Parallelität aber entscheidende Konstrukte aufweist und damit alles andere den bereits bekannten Techniken für die Verifikation von sequentiellen Programmen überläßt. Diese formalen parallelen Kalküle beschränken sich fast ausschließlich auf die Modellierung der Kommunikation zwischen Prozessen. Berechnungen werden in Black-Boxes vorgenommen. Diese sind einfach Funktionen, die als korrekt vorausgesetzt werden.

Andere parallele Modelle sind z.B. Zellularautomaten (wie „Conway's Game Of Life“) und die P-RAM (**P**arallel **R**andom **A**ccess **M**achine) . Erstere jedoch sind im allgemeinen in ihrer Struktur zu starr und deswegen für parallele Probleme weniger geeignet. Bei der P-RAM hingegen erfolgt die

Kommunikation implizit über den Speicher. Damit aber lassen sich viele Aussagen der parallelen Kalküle, die ja die Kommunikation zwischen den Prozessen untersuchen, nicht erhalten.

Der Einsatz von parallelen Kalkülen ist auch nicht, wie man leicht denken könnte, einfach nur eine Verschnörkelung der sequentiellen Programmiertechnik, da viele Probleme der Realität wirklich ein paralleles Modell erfordern, um vernünftig beschrieben zu werden. Man denke zum Beispiel an die Telekommunikation, wo verschiedene Ortsvermittlungen zusammenarbeiten müssen, um eine Verbindung herzustellen. Diese arbeiten dabei natürlich echt gleichzeitig und sind außerdem auf die gegenseitige Kommunikation angewiesen.

## 1.3 Parallele Kalküle

Im Jahre 1978 veröffentliche C.A.R. Hoare den Artikel „Communicating Sequential Processes“ [Hoa78], der einen der ersten parallelen Kalküle vorstellte, und zwar CSP (**C**ommunicating **S**equential **P**rocesses), welcher seitdem häufig verwendet wird. Es basiert auf verschiedenen Kanälen, auf denen Handshaking-Übertragung[1] stattfindet, d.h. ein Absender wird seine Botschaft erst los, wenn der Empfänger sie ihm abnimmt, wenn sie sich „die Hand geben“. Dasselbe Prinzip verfolgt auch die von Robin Milner in seinem 1980 erschienenen Artikel „A Calculus of Communicating Systems“ [Mil80] vorgestellte Sprache CCS (**C**alculus of **C**ommunicating **S**ystems).

Auf beiden Sprachen existiert eine interessante Algebra, und die Tatsache, daß beide Autoren mehr oder minder unabhängig auf ähnliche Sprachen gekommen sind, legt nahe, daß die benutzten Kommunikationskonstrukte grundlegender Natur sind. Bis dahin wurden vorwiegend höher liegende Modelle wie Semaphore, Monitor usw. genutzt, die in der neuen Theorie problemlos modelliert werden konnten. Ein Problem allerdings gibt es bei den Handshaking-Sprachen: Eine Priorisierung ist nur sehr schwer modellierbar. Das liegt vereinfacht gesprochen daran, daß ein Prozeß mit hoher Priorität, der eine Botschaft an einen anderen Prozeß mit niedriger Priorität hat, erst dann fortsetzten kann, wenn ihm seine Botschaft abgenommen wurde, d.h. der „langsame“ Prozeß bremst den „schnellen“ aus.

## 1.4 Der parallele Kalkül TCBS und seine Bedeutung

Die von K.V.S. Prasad von 1991 an entwickelte Sprache CBS (**C**alculus of **B**roadcasting **S**ystems) und deren Erweiterung TCBS (**T**imed CBS) hat dieses Problem nicht, da sie auf Broadcasting[2] basiert, d.h. die Botschaft eines Prozesses ist im Prinzip für alle anderen hörbar, und der sendende Prozeß kann sofort seine Arbeit fortsetzen. Außerdem ist Broadcast eigentlich das natürlichere Mittel der Kommunikation, auch und vor allem in der praktischen Anwendung, also z.B. in einem Local Area Network, aber auch bei der mobilen Telefonie. Punkt-zu-Punkt-Verbindungen, wie sie von CCS und CSP benutzt werden, sind meist nur auf der Grundlage von Broadcast-Medien implementiert. Insgesamt hat TCBS einige schwerwiegende Vorteile, die eine Beschäftigung mit diesem Kalkül sinnvoll erscheinen lassen (siehe dazu auch [Pra93], [Pra95] und [Pra96]).

So wird dieser Kalkül an der Universität Göteborg, wo K.V.S. Prasad Dozent ist, nicht nur in der Forschung von der renommierten Concurrency Group behandelt, sondern auch in der Ausbildung verwendet. Während meines Studiums in Göteborg im Studienjahr 1994/95 belegte ich den Kurs

[1] Handshake auf deutsch: Handschlag; in dieser Arbeit werden häufig die englischen Fachbegriffe anstelle der deutschen Übersetzungen verwendet.

[2] deutsch: Rundfunk; in dem Sinne, daß alle hören, was einer sendet.

„Rechnernetze und Rechnerkommunikation“, der auf Grundlage des Buches [Sha94] detailliert Netzprotokolle der Rechnerkommunikation behandelte. Zum erfolgreichen Bestehen des Kurses war eine Reihe von Protokollen in einer simulierten Umgebung in TCBS zu implementieren. Die Sprache TCBS war auf Grundlage der funktionalen Programmiersprache Haskell implementiert. Es gibt aber auch andere Implementationen, z.B. eine physisch verteilte, aber auch eine von mir programmierte Implementation auf Grundlage von C++ (siehe [Wil95b]).

## 1.5 Motivation und Aufbau der Arbeit

Durch theoretische Überlegungen, aber auch die praktische Arbeit mit diesem Kalkül angeregt, begann mein Interesse für dieses Gebiet der theoretischen Informatik. In meiner Praktikumsarbeit [Wil95a] für den oben genannten Kurs entwarf und programmierte ich ein fiktives Netz von Wetterstationen, die über unsichere Medien verbunden waren und miteinander kommunizieren mußten. Ich war aber nicht in der Lage zu beweisen, daß die von mir verwendeten Protokolle bzw. deren Implementationen korrekt waren.

Ein großer Teil der heutzutage verwendeten Netzprotokolle ist ebenfalls nicht als korrekt nachgewiesen. Man hofft, daß sie funktionieren, und in praxi arbeiten sie zuverlässig. Aber bei vernetzten Rechnern, die ja ein verteiltes System darstellen, kann man nicht überblicken, welche Konstellationen auftauchen können, und wie sich das System in diesen verhält.

Aus diesem Grunde wählte ich das Thema dieser Arbeit. Sie ist ein kleiner Schritt hin zum automatischen Nachweis der Korrektheit von Protokollen, z.B. der Internet-Protokoll-Familie (TCP/UDP/IP), bietet aber auch Ansatzpunkte für andere parallele Algorithmenklassen.

Die vorliegende Arbeit benutzt also TCBS, um Beweistechniken für den Nachweis der Korrektheit von Protokollen vorzustellen. Dabei wird in Kapitel 2 eine anschauliche Darstellung des Kalküls TCBS anhand von Beispielen gegeben. Dieser Kalkül wird dann in Kapitel 3 formal eingeführt. Dazu verwende ich im Gegenteil zur Originalarbeit [Pra96] eine formale Definition mit Hilfe von Grammatiken. Diese etwas kompliziertere Vorgehensweise sichert faktisch erst die Beweisbarkeit von Sätzen über dieser Sprache. In Kapitel 4 gehe ich auf die vorhandenen Bisimulationsbegriffe ein. Diese definieren Äquivalenzrelationen auf TCBS, welche eine erste Form der Entsprechung zweier Prozesse bieten. In diesen zwei Kapiteln werden verschiedene in den Originalarbeiten lediglich intuitiv begründete Sätze bewiesen und einige neue formuliert (insbesondere Satz 3.4 zur Aktionsbereitschaft).

In Kapitel 5 wird die Bedeutung von Bisimulationen für den Beweis der Korrektheit speziell für eine Klasse von Protokollen dargestellt. Ich beschreibe, wann ein Protokoll als korrekt zu gelten hat und stelle die dabei mit den herkömmlichen Bisimulationen auftretenden Probleme dar. Ausgehend davon führe ich in Kapitel 6 einen diese Probleme lösenden erweiterten Kalkül ein. Damit kann ich dann in Kapitel 7 die in Kapitel 5 gegebene Beschreibung, wann ein Protokoll korrekt heißt, in eine vollständige Definition umwandeln. In Kapitel 7 wird außerdem als Beispiel dafür das Alternating-Bit-Protokoll kurz erläutert, dessen Implementation angegeben. Ein Weg zum Beweis der Korrektheit dieser Implementation wird vorgestellt und wichtige Teile bewiesen.

Insgesamt stellt diese Arbeit neben der formalen Einführung von TCBS auf Grundlage der theoretischen Informatik einen erweiterten Kalkül $\mathrm{TCBS}'$ vor, der zusammen mit der herkömmlichen schwachen Bisimulation die Abstraktion von der Zeit zuläßt und damit für die Behandlung von Protokollen als günstig erscheint.

# 2 Anschauliche Einführung von TCBS

## 2.1 Das Modell

In einem parallelen Programmierkalkül agieren, im Gegensatz zu einem sequentiellen, mehrere Prozesse gleichzeitig. TCBS kann man sich anhand des folgenden praktischen Modells veranschaulichen: In einem Raum sitzen mehrere Personen zusammen, die sich miteinander unterhalten. Den Personen entsprechen in TCBS die Prozesse, und der Sprache, in der sich die Personen unterhalten, entspricht in TCBS der Datentyp, über den die Prozesse kommunizieren. Für die Personen im Raum gelten bei der Unterhaltung folgende Einschränkungen:

(i) Es darf zu jedem Zeitpunkt nur eine der Personen (Prozesse) etwas sagen. Das bedeutet, daß auf dem Broadcast-Medium, auf dem die Unterhaltung stattfindet, schon von der Möglichkeit einer Kollision von Nachrichten abstrahiert wurde. Das Gegenteil ist z.B. auf einem praktischen Broadcast-Medium wie dem Ethernet der Fall. Dort kommt dann das CD/CSMA (Collision Detect/Carrier Sense Multiple Access) - Protokoll zum Einsatz, um die kollisionsfreie Übertragung von Daten zu sichern. (siehe [Sha94], S.92 ff.). Es ist aber für einen Kalkül, der zur Untersuchung der Parallelität gedacht ist, durchaus vernünftig, davon abzusehen.

(ii) Man kann nicht erkennen, wer gesprochen hat. Diese Einschränkung ist bei näherer Betrachtung keine wesentliche, denn jede Person kann ja ihrer Äußerung voranstellen, wer sie ist.

(iii) Alle Personen müssen dieselbe Sprache sprechen. Diese Bedingung erscheint natürlich und ist auch keine echte Einschränkung, denn man kann auch Übersetzer einführen, wie später gezeigt wird.

(iv) Die Auswahl des jeweiligen Sprechenden erfolgt zufällig gleichverteilt, wenn keine Prioritäten vergeben werden. Diese Annahme ist nur für die praktische Ausführung eines parallelen Programms wichtig, da in der Theorie immer nur danach gefragt wird, ob ein Prozeß etwas Bestimmtes sagen könnte, und wie sich in diesem Falle die an der Unterhaltung beteiligten Prozesse verhalten würden. Die Auswahl des Sprechenden spielt in diesem Kalkül also nur insofern eine Rolle, als daß jeder, der sprechen kann, auch zum Zuge kommen könnte.

Ein einfaches Beispiel für einen Algorithmus in TCBS ist das folgende: In einem Raum sitzen $n$ Personen zusammen und wollen herausfinden, wer von ihnen am ältesten ist. Dazu gibt es folgenden einfachen lokalen Algorithmus. Jeder versucht, sein Alter zu sagen. Sobald er es geschafft hat, ist er ruhig. Wenn er in der Zwischenzeit von jemand anderem etwas hört, dann ist er ruhig, wenn der Andere älter oder gleichalt ist, und versucht ansonsten weiter, sein Alter allen mitzuteilen. Dasjenige Alter, welches zuletzt im Raum gesagt wird, ist das höchste. Das ist intuitiv klar, denn würde es ein höheres Alter geben, würde es der Betreffende noch sagen. Andererseits muß die Folge der mitgeteilten Alter auch streng monoton wachsend sein, denn jede Person des Alters $k$ schaltet alle noch mitteilungsbedürftigen Personen mit einem Alter $\leq k$ aus.

Wie bereits erwähnt, existieren in TCBS Übersetzer. Diese dienen nicht nur dazu, Prozesse, die verschiedene Sprachen sprechen, zusammenarbeiten zu lassen, sondern auch dazu, bestimmte unerwünschte Ereignisse auf dem Medium auszufiltern. Das ist z.B. für die Programmierung von Unterroutinen praktisch, die lokal über das Medium Informationen austauschen können, die die

Umgebung nicht hört. Dieses Verhalten wird durch die Einführung eines mit $\tau$ bezeichneten Hintergrundrauschens erzielt, das ein Übersetzer anstatt eines Elementes der gesprochenen Sprache von sich geben kann. Dieses wird von allen Personen, die es hören, ignoriert, ohne daß sie selber etwas sagen könnten. Zur Erläuterung eines Übersetzers stelle man sich ein Zimmer vor, in dem deutsch sprechende Personen sitzen, sowie ein Nebenzimmer, in dem englisch sprechende Personen sitzen. An der Tür zwischen beiden Zimmern sitzt ein Übersetzer. Er übersetzt jetzt alles, was auf der einen Seite gesagt wird, in die jeweils andere Sprache für das andere Zimmer, oder aber in das Hintergrundrauschen $\tau$, das dann von allen Personen, die es hören, ignoriert wird. Das bedeutet, daß in beiden Zimmern zusammen nur einer reden darf, denn sonst könnte es zu einer Kollision zwischen dem, was der Übersetzer gesagt hat, und dem, was sonst im Raum gesagt wurde, kommen. Es darf auch zu keiner Kollision zwischen $\tau$ und anderen Worten kommen. Der Übersetzer ist kein Prozeß, wie die Personen in den Räumen, denn er hört und spricht in jedem Schritt und zwar gleichzeitig. Wichtig ist noch, daß die Personen in den Räumen nicht erkennen können, ob der Übersetzer oder eine Person in ihrem Raum gesprochen hat.

Außerdem wird in TCBS der Begriff der Zeit eingeführt. Das ist der eigentliche Unterschied zwischen CBS und TCBS. Zeit vergeht in TCBS genau dann, und zwar global, wenn keiner der Prozesse etwas sagen will, d.h. wenn alle Prozesse warten oder hören. Dies ist in vielen Anwendungen notwendig, weil z.B. bei obigem Beispiel von einem parallel arbeitenden Prozeß ohne Zeitbegriff nicht festgestellt werden könnte, wer der letzte war, der gesprochen hat und damit auch nicht, was das größte Alter ist. Dabei wird in dieser Arbeit die Zeit nur diskret betrachtet. K.V.S. Prasad hat TCBS auch mit stetiger Zeit betrachtet (siehe [Pra96]), dies jedoch würde in dieser Arbeit mit dem neudefinierten erweiterten Kalkül kollidieren (siehe Kapitel 6 auf Seite 45). Der Zeitbegriff von TCBS ist natürlich künstlich, da ihm zufolge keine Zeit vergeht, wenn ein Prozeß ununterbrochen arbeitet. Am nähesten kommt ihm noch die kumulative „idle“-Zeit eines Multitasking-Rechners.

## 2.2 Der Talk-Operator

Während bei der Betrachtung herkömmlicher sequentieller formaler Programmierkalküle Berechenbarkeits- und Komplexitätsuntersuchungen im Vordergrund stehen, geht es bei parallelen Kalkülen vorwiegend um die Modellierung der Kommunikation von parallel laufenden Prozessen. TCBS wird durch eine Menge $P$ von Prozessen gebildet, die durch eine Syntax definiert ist. Wie sich Prozesse verhalten, ist durch die Semantik (eine operationale Semantik) festgelegt. Sie beschreibt, wie ein Prozeß aus $P$ durch Ausführung einer bestimmten Kommunikationsaktion in einen neuen Prozeß (wieder aus $P$) übergeht. Jetzt werden einzelne Elemente der Syntax anhand einiger einfacher Beispiele eingeführt und deren Semantik erläutert.

Wie am Modell von TCBS gut sichtbar, kann ein Prozeß etwas sagen wollen, muß aber gleichzeitig damit rechnen, daß ein anderer zuerst spricht und er dann reagieren muß. Dieses Verhalten zeigt der sogenannte **Talk**-Operator. Er heißt so, denn ein solches Verhalten entspricht dem in einer Unterhaltung. Um darzustellen, wie der Prozeß sich verhält, wenn er etwas hört, wird eine Funktion angegeben, die für jede mögliche Eingabe ergibt, welches der Nachfolgeprozeß sein wird. Der alte Prozeß wird dann durch diesen ersetzt, und es wird in der Unterhaltung auf dem Medium fortgefahren. Ansonsten ist für den Talk-Operator natürlich noch wichtig, was er sagt, wenn er es darf, und wie er sich dann entwickelt.

Sei nun also $P$ die Menge aller Prozesse und $\alpha$ die auf dem Medium gesprochene Sprache, in diesem Falle eine beliebige endliche Menge von Symbolen. Dann benötigt der Talk-Operator drei Argumente: eine Funktion $f$ von $\alpha$ in $P$, die angibt, wie der Prozeß sich verhält, wenn er etwas hört, ein Element $v \in \alpha$, das dieser Prozeß selbst sagen will und den Prozeß $p \in P$, in den er sich

in diesem Fall verwandelt. Die Schreibweise dafür ist:

$$f\&<v, p>$$

Der Prozeß $f\&<v, p>$ versucht $v$ zu sagen und zu $p$ zu werden. Ansonsten, wenn ein anderer Prozeß sprechen darf und $v'$ sagt, wird $f\&<v, p>$ zu $f(v')$, welches wieder ein Prozeß ist.

## 2.3 Der Compose-Operator

Bisher ist es so, daß ein Talk-Prozeß immer an der Reihe ist, etwas zu sagen, denn er ist noch völlig allein. Um mehrere Prozesse gleichzeitig laufen zu lassen, das eigentliche Ziel dieses Kalküls, müssen sie kombiniert werden können. Dazu dient der wichtigste Operator in TCBS, der Parallel-Composition-Operator oder kurz **Compose**-Operator genannt wird. Er wird mit | bezeichnet und verbindet zwei Prozesse $p$ und $q$ aus der Menge aller Prozesse $P$ zu einem neuen Prozeß $p|q$ aus $P$.

Wie regelt nun aber der Parallelitätsoperator die Kommunikation zwischen zwei (oder mehreren) Prozessen? Es werden alle Prozesse, die durch einen Parallelitätsoperator verbunden sind (wie in $p|q|r$), untersucht, ob sie sprechen möchten. Aus allen Prozessen, die sprechen möchten, wird zufällig einer ausgewählt. Dieser darf dann sprechen. Alle anderen hören zu. Per Compose-Operator verbundene Prozesse verhalten sich also wie Personen in einer moderierten Gesprächsrunde, zu jedem Zeitpunkt spricht nur einer. Dabei ist der Compose-Operator, wie später gezeigt wird, assoziativ, d.h. der Verzicht auf die Klammerung ist legitim.

Jetzt sind wir in der Lage, das oben angegebene Beispiel der Personen, die das höchste Alter herausfinden wollen, durchzurechnen. Dazu definiere ich den Prozeß, der eine Person vom Alter $n$ repräsentiert. Er werde mit $\mathrm{Cell}_n$ bezeichnet.

$$\mathrm{Cell}_n := f_{\mathrm{Cell}_n}\&<n, \mathbf{0}>$$

$\mathrm{Cell}_n$ versucht also, im Einklang mit der obigen Erklärung, sein Alter, also $n$ zu sagen und wird dann zu Nil (**0**). **Nil** ist der abgestorbene Prozeß. Die Person ist also ruhig, nachdem sie ihr Alter gesagt hat. Nun muß noch die Funktion $f_{\mathrm{Cell}_n}$ definiert werden, die angibt, wie sich $\mathrm{Cell}_n$ verhält, wenn er nicht selbst etwas sagen darf, sondern etwas gesagt bekommt:

$$f_{\mathrm{Cell}_n}(m) := \begin{cases} \mathbf{0} & m \geq n \\ \mathrm{Cell}_n & \text{sonst} \end{cases}$$

Er bleibt er selbst ($\mathrm{Cell}_n$), wenn das gehörte Alter unter seinem ist, und wird ruhig, wenn das gehörte Alter größer oder gleich seinem eigenen ist.

Dieses entspricht genau dem oben angegebenen Algorithmus. Nun kann man zur Probe einmal ein paar Personen (Prozesse) unterschiedlichen Alters mit Hilfe des Compose-Operators zusammensetzen, um zu sehen, was passiert:

Nehmen wir z.B. drei Prozesse mit dem Alter $4$, $5$ bzw. $6$. Die parallele Kombination daraus sieht folgendermaßen aus:

$$\mathrm{Cell}_4 \mid \mathrm{Cell}_5 \mid \mathrm{Cell}_6$$

Da alle drei Prozesse sprechen wollen, wählen wir einen beliebigen aus, also z.B. $\mathrm{Cell}_5$ und erlauben ihm, zu sprechen: $\mathrm{Cell}_5$ will $5$ sagen und zu **0** werden. Wie verhalten sich aber $\mathrm{Cell}_4$ bzw. $\mathrm{Cell}_6$, wenn sie $5$ hören? $\mathrm{Cell}_4$ wird zu **0**, weil das gehörte Alter $5$ über seinem eigenen Alter $4$ liegt und er damit aus dem Rennen ist. $\mathrm{Cell}_6$ hingegen bleibt $\mathrm{Cell}_6$, denn $5$ ist kleiner $6$. Damit bleibt nach einem Schritt folgendes übrig:

$$\mathbf{0}|\mathbf{0}|\,\mathrm{Cell}_6$$

Jetzt will nur noch $\mathrm{Cell}\,6$ etwas sagen, denn **0** ist ja der abgestorbene Prozeß. $\mathrm{Cell}_6$ sagt $6$. Die beiden anderen Prozesse reagieren nicht darauf. Übrig bleibt also nach 2 Schritten:

$$\mathbf{0}|\mathbf{0}|\mathbf{0}$$

Kein Prozeß hat mehr etwas zu sagen. Es wird nichts mehr passieren. Wir haben also erhalten, daß $\mathrm{Cell}_4 \,|\, \mathrm{Cell}_5 \,|\, \mathrm{Cell}_6$ erst $5$ und dann $6$ sagen kann und dann abstirbt. Aber natürlich hätte $\mathrm{Cell}_4 \,|\, \mathrm{Cell}_5 \,|\, \mathrm{Cell}_6$ auch erst $4$ sagen können. In diesem Falle hätte das übrigbleibende $\mathbf{0}|\,\mathrm{Cell}_5 \,|\, \mathrm{Cell}_6$ entweder $6$ oder $5$ sagen können usw. Es gibt schon in diesem einfachen Fall mehrere Möglichkeiten für **Läufe** eines Programms. Das ist Ausdruck des in parallelen Kalkülen vorkommenden Nichtdeterminismus. Und auch die Ursache dieses Nichtdeterminismus ist erkennbar. Es ist die sogenannte Racing-Condition, bei deren Auftreten das Ergebnis eines Programms davon abhängt, welches der parallel laufenden Teilprogramme zuerst eine bestimmte Aktion ausführen konnte. In diesem Beispiel ist das Endergebnis zwar vom aktuellen Lauf unabhängig, aber es könnte z.B auch sein, daß alle Personen, nachdem sie irgendein anderes Alter gehört haben, schweigen. Dann würde der Lauf davon abhängen, wer zuerst spricht. In der praktischen Programmierung ist man aber natürlich meistens daran interessiert, eindeutige Ergebnisse zu erzielen.

Nun kann es natürlich auch Personen geben, die in einem bestimmten Zustand nichts sagen wollen und nur zuhören, und solche, die nur etwas sagen wollen, und auf Gehörtes nicht reagieren. Für erstere führen wir den **Hear**-Operator ein, dessen Schreibweise

$$?f$$

lautet. Dabei ist $f$ vom gleichen Typ wie die Funktion $f$ im Talk-Operator. Dieser Operator ist eine wirkliche Erweiterung des Kalküls, wohingegen der **Say**-Operator, der nur etwas sagen will, und alles, was er hört, ignoriert, als abgeleitete Version des Talk-Operators definiert werden kann. Der Say-Operator, der $v$ sagen und zu $p$ werden will, hat folgende Schreibweise:

$$!{<}v, p{>}$$

## 2.4 Transitionen

Die Prozesse im obigen Beispiel entwickeln sich in jedem Schritt zu anderen Prozessen. Da liegt es nahe, Übergangsrelationen für solche Übergänge einzuführen. Diese heißen **Transitionen** und werden mit Hilfe eines Pfeils $\longrightarrow$ dargestellt. Dabei werden die Pfeile auch mit der Aktion versehen, die der Prozeß ausführte, bevor er sich veränderte, also z.B. etwas sagen, etwas hören oder Zeit vergehen lassen. Diese werden wie folgt bezeichnet:

- Wenn der Prozeß $p$, nachdem er $v$ aus $\alpha$ gesagt hat, zu $p'$ wird, dann schreibt man $p \xrightarrow{v!} p'$. Das Ausrufezeichen steht also für Sprechen.

- Wenn der Prozeß $p$, nachdem er $v$ aus $\alpha$ gehört hat, zu $p'$ wird, dann schreibt man $p \xrightarrow{v?} p'$. Das Fragezeichen steht also für Hören.
- Wenn der Prozeß $p$, nachdem er $\delta \in \mathbb{N}$ Zeiteinheiten hat vergehen lassen, zu $p'$ wird, dann schreibt man $p \xrightarrow{\delta:} p'$. Der Doppelpunkt steht also für Warten.

## 2.5 Der Timeout-Operator

Zurück zum Raum mit den Personen, die das Maximalalter bestimmen wollen. Wie soll eine sich noch in diesem Raum befindliche Person herausfinden, welches das höchste Alter ist? Sie kann ja nicht ohne weiteres beurteilen, ob der letzte Prozeß bereits gesprochen hat oder ob noch etwas kommt. Sie kann aber erkennen, wann niemand mehr etwas sagen kann. Das ist, wie oben angegeben, der Fall, wenn Zeit vergeht. Wenn diese Person merkt, daß Zeit vergeht, dann weiß sie, das das zuletzt gesagte Alter das höchste ist.

Es muß ein neuer Konstruktor zur Sprache dazugenommen werden, der **Timeout**-Operator. Der Timeout-Operator hat als Argumente die Zeit, die er maximal warten soll ($\delta \in \mathbb{N}$), sowie den Prozeß ($p \in P$), den er umschließt. Die zugehörige Notation ist:

$$\delta{:}p$$

Der Timeout-Operator hält dem Prozeß $p$ bildlich gesprochen den Mund zu, bis dieser etwas gehört hat oder bis $\delta$ Zeiteinheiten vergangen sind. Sobald der Prozeß $p$ etwas gehört hat, darf er wieder sprechen. Er kann sich dabei natürlich eventuell durch das Gehörte bereits verändert haben. Dieser Operator trägt den Namen Timeout zu Recht, denn er wartet maximal $\delta$ Zeiteinheiten, während derer er aber immer ansprechbar bleibt. Aber erst nach den $\delta$ Zeiteinheiten wird der von ihm gekapselte Prozeß aktiv, könnte also etwas sagen.

Damit kann man nun einen Prozeß, der das höchste Alter im Raum feststellen soll, beschreiben. Er heiße $\mathrm{Max}_n$. Dabei ist $n$ das zuletzt gehörte Alter.

$$\mathrm{Max}_n := 1{:}f_{\mathrm{Max}_n}\&<-n, \mathbf{0}>$$

Wenn eine Zeiteinheit vergeht, und also kein anderer Prozeß mehr etwas sagt, dann ruft $\mathrm{Max}_n$ das zuletzt gehörte Alter. Es wird negiert, damit man es von außen von den Äußerungen der anderen Prozesse unterscheiden kann. Die Negierung bildet also aus einer Altersangabe eine negative Zahl, damit das eine nicht mit dem anderen verwechselt werden kann. Wenn $\mathrm{Max}_n$ jedoch noch etwas hört, dann muß er natürlich das als letztes Alter annehmen. Das geschieht durch die Funktion $f_{\mathrm{Max}_n}$:

$$f_{\mathrm{Max}_n}(v) := \mathrm{Max}_v$$

Hieran sieht man auch sehr gut, daß es in TCBS keinen Speicher im herkömmlichen Sinne gibt. Jeder Prozeß trägt die von ihm benötigten Daten in seinem Namen mit sich. Einer Änderung von Speicherinhalten entspricht die Änderung des Zustands eines Prozesses. Würde man in TCBS (oder in anderen ähnlichen parallelen Kalkülen) globalen Speicher zulassen, dann könnten die Prozesse über ihn, an den Kommunikationskonstrukten von TCBS vorbei, kommunizieren. Das würde aber diesen Kalkül ad absurdum führen. Lokaler Speicher für einen Prozeß hingegen ist einfach zu erhalten. Man versieht, wie schon in den bisherigen Beispielen geschehen, die Prozeßkonstanten (wie $\mathrm{Cell}_n, \mathrm{Max}_n$) mit einem Index. Dieser ist der Speicher des Prozesses. Er kann

ja von einem zum nächsten Schritt geändert werden. Das wird vor allem durch die Funktion $f$ des Talk-Operators auf Grundlage des gerade Gesagten und des letzten Zustandes bewirkt. Daran sieht man auch die Beziehungen der Bezeichnungen Name, Prozeß und Zustand untereinander. Ein Name eines Prozesses ist eine Prozeßkonstante, der ein Prozeß zugeordnet ist. Ein Zustand eines Prozesses, z.B. nach einer Transition, ist wieder ein Prozeß.

Lassen wir aber nun zur Kontrolle den Prozeß $\mathrm{Cell}_4 \mid \mathrm{Cell}_5 \mid \mathrm{Max}_0$ laufen. Da $\mathrm{Max}_0$ nichts sagen will, wird $\mathrm{Cell}_4$ oder $\mathrm{Cell}_5$ ausgewählt. Nehmen wir an $\mathrm{Cell}_4$ ist an der Reihe. Er sagt also $4$ und entwickelt sich zu $\mathbf{0}$, das heißt in der Notation der Transitionen, daß $\mathrm{Cell}_4 \xrightarrow{4!} \mathbf{0}$. $\mathrm{Cell}_5$ bleibt, wenn es $4$ hört, unverändert, also gilt: $\mathrm{Cell}_5 \xrightarrow{4?} \mathrm{Cell}_5$. $\mathrm{Max}_0$ hingegen wird, wenn es $4$ hört, zu $\mathrm{Max}_4$, hat sich das letzte gesagte Alter gemerkt, also seinen eigenen Zustand mit dem lokalen Speicher $n$ geändert. Es gilt: $\mathrm{Max}_0 \xrightarrow{4?} \mathrm{Max}_4$. Als Ergebnis haben wir also den Gesamtprozeß $\mathbf{0}\mid \mathrm{Cell}_5 \mid \mathrm{Max}_4$ und es gilt:

$$\mathrm{Cell}_4 \mid \mathrm{Cell}_5 \mid \mathrm{Max}_0 \xrightarrow{4!} \mathbf{0}\mid \mathrm{Cell}_5 \mid \mathrm{Max}_4 \,.$$

Jetzt will nur noch $\mathrm{Cell}_5$ etwas sagen, nämlich $5$, sein Alter. Es wird dann zu $\mathbf{0}$. $\mathbf{0}$ ist abgestorben, kann also ignoriert werden, $\mathrm{Max}_4$ hingegen, wird zu $\mathrm{Max}_5$, wenn es $5$ hört. Es gilt also:

$$\mathbf{0}\mid \mathrm{Cell}_5 \mid \mathrm{Max}_4 \xrightarrow{5!} \mathbf{0}|\mathbf{0}\mid \mathrm{Max}_5 \,.$$

Da nun keiner der Prozesse mehr etwas sagen will, kann Zeit vergehen. Wenn $\mathrm{Max}_5$ eine Zeiteinheit vergehen läßt, wird es zu $f_{\mathrm{Max}_5}\&<-5,\mathbf{0}>$. Es gilt also:

$$\mathbf{0}|\mathbf{0}\mid \mathrm{Max}_5 \xrightarrow{1:} \mathbf{0}|\mathbf{0}|f_{\mathrm{Max}_5}\&<-5,\mathbf{0}>.$$

$f_{\mathrm{Max}_5}\&<-5,\mathbf{0}>$ aber kann $-5$ sagen und zu $\mathbf{0}$ werden. Es gilt also weiterhin:

$$\mathbf{0}|\mathbf{0}|f_{\mathrm{Max}_5}\&<-5,\mathbf{0}> \xrightarrow{-5!} \mathbf{0}|\mathbf{0}|\mathbf{0}.$$

Damit ist der Prozeß abgestorben und außerdem können dazu parallelgesetzte Prozesse jetzt genau bestimmen, welches das höchste Alter ist. Es ist die Negation der einzigen negativen Zahl, die gesagt wird (natürlich seien dazu alle Altersangaben größer Null).

Ein weiterer die Zeit betreffender Operator ist der **Delay**-Operator. Er hält, um im obigen Bild zu bleiben, der entsprechenden Person nicht nur den Mund zu sondern auch die Ohren und zwar genauso lange, wie gewünscht. Ein um $\delta$ Einheiten verzögerter Prozeß $p$ wird mit

$$\delta.p$$

bezeichnet.

## 2.6 Übersetzer

Wenn nun andere Prozesse das höchste Alter wissen, aber keine anderen Äußerungen hören wollen, muß man um die Prozesse $\mathrm{Cell}_n$ und $\mathrm{Max}_n$ einen Übersetzer legen, der nur das höchste Alter nach außen dringen läßt. Ein solcher Übersetzer, auch **Translate**-Operator genannt, wird häufig mit dem griechischen Buchstaben $\Phi$ bezeichnet. Er wird direkt vor den zu übersetzenden Prozeß geschrieben. Für die Definition von $\Phi$ werden eigentlich zwei Definitionen benötigt, eine für die Übersetzung von innen nach außen sowie eine für die Übersetzung von außen nach innen. Diese

Funktionen werden dann mit $\Phi_\uparrow$ bzw. $\Phi_\downarrow$ bezeichnet. In unserem Beispiel könnte der Übersetzer so definiert werden:

$$\Phi_{\mathrm{Cell}\uparrow}(v) := \begin{cases} -v & \text{falls } v < 0 \\ \tau & \text{sonst} \end{cases}$$
$$\Phi_{\mathrm{Cell}\downarrow}(v) := \tau$$

Die Definition von $\Phi_{\mathrm{Cell}\downarrow}$ sorgt dafür, daß innerhalb des Übersetzers nichts von außen gehört werden kann, da alles, was außen gesagt wird, innen als das Hintergrundrauschen $\tau$ ankommt, das von allen Prozessen ignoriert wird. Was innen gesagt wird, dringt negiert nach außen, wenn es negativ ist. Eine innen negativ gesagte Zahl aber stammt vom Prozeß $\mathrm{Max}_n$ und ist damit das höchste Alter. Nun können wir testen, welches Verhalten der Prozeß $\Phi_{\mathrm{Cell}}(\mathrm{Cell}_1 \mid \mathrm{Cell}_2 \mid \mathrm{Max}_0)$ zeigen könnte. Dazu untersuchen wir aber zur besseren Übersicht erst das Verhalten des Prozesses $\mathrm{Cell}_1 \mid \mathrm{Cell}_2 \mid \mathrm{Max}_0$. Dieser hat folgende Entwicklungsmöglichkeiten:

$$\mathrm{Cell}_1 \mid \mathrm{Cell}_2 \mid \mathrm{Max}_0 \xrightarrow{2!} \xrightarrow{1:} \xrightarrow{-2!}$$
$$\mathrm{Cell}_1 \mid \mathrm{Cell}_2 \mid \mathrm{Max}_0 \xrightarrow{1!} \xrightarrow{2!} \xrightarrow{1:} \xrightarrow{-2!}$$

Wenn nun wieder der Übersetzer hinzugenommen wird, dann sehen die Transitionen wie folgt aus:

$$\Phi_{\mathrm{Cell}}(\mathrm{Cell}_1 \mid \mathrm{Cell}_2 \mid \mathrm{Max}_0) \xrightarrow{\tau!} \xrightarrow{1:} \xrightarrow{2!}$$
$$\Phi_{\mathrm{Cell}}(\mathrm{Cell}_1 \mid \mathrm{Cell}_2 \mid \mathrm{Max}_0) \xrightarrow{\tau!} \xrightarrow{\tau!} \xrightarrow{1:} \xrightarrow{2!}$$

Diese Darstellung zeigt, daß nun der gewünschte Effekt erzielt wurde. Außerhalb des Übersetzers wird nur noch das maximale Alter gehört. Auf diese Art und Weise kann man also in TCBS eine Art von Unterroutinen programmieren, die bis auf die spezifizierte Ein- und Ausgabe von der Außenwelt gekapselt sind.

In den obigen vier Transitionsangeben sind die Zwischen- und Endzustände der Prozesse weggelassen. Diese Notation wird in dieser Arbeit häufig verwendet. Im obigen Beispiel kann man die fehlenden Zustände jedoch leicht ableiten.

Übersetzer können aber nicht nur, wie hier gezeigt, innerhalb einer Sprache Symbole wandeln, sondern auch zwischen verschiedenen Sprachen übersetzen. Eine weitere wichtige Bedingung für einen Übersetzer ist, daß er in beiden Richtungen $\tau$ immer zu $\tau$ übersetzt. Ansonsten könnte ja ein Übersetzer die von einem anderen Übersetzer gekapselten Daten wieder sichtbar machen. Die Daten als solche wären natürlich verloren, aber ein außen liegender Prozeß könnte merken, daß ein gekapselter Prozeß etwas gesagt hat. Auch das ist schon unerwünscht.

## 2.7 Die speisenden Philosophen

Ein in der Theorie der parallelen Prozesse beliebtes Beispiel ist das der speisenden Philosophen (englisch: Dining Philosophers). Dabei wollen $n$ Philosophen Spaghetti essen. Sie sitzen um einen runden Tisch. Auf diesem stehen $n$ Teller mit Nudeln und zwischen je 2 Tellern liegt eine Gabel,

also insgesamt $n$ Gabeln. Jeder Philosoph benötigt zum Essen 2 Gabeln, die auf seiner linken und die auf seiner rechten Seite.

Um diese Philosophen in TCBS zu repräsentieren, müssen wir erst modellieren, wie ein Philosoph eine Gabel aufnimmt und wie er sie wieder hinlegt. Dazu numerieren wir die Gabeln so, daß rechts neben dem Philosophen Nr. $k$ die Gabel Nr. $k$ liegt. Die Philosophen ihrerseits sitzen auch von 1 bis $n$ durchnumeriert am Tisch. Wenn also der Philosoph Nr. $k$ essen will, muß er die Gabel $k$ und die Gabel $k-1$ aufnehmen, es sei denn $k$ ist gleich 1. Der Philosoph Nr. 1 muß die Gabeln Nr. 1 und Nr. $n$ aufnehmen, um zu essen.

Wenn ein Philosoph die Gabel Nr. $k$ aufnimmt, dann ruft der dem Philosophen entsprechende Prozeß ein $k$ und wenn ein Philosoph die Gabel Nr. $k$ wieder hinlegt, dann ruft der Prozeß $-k$. Nun können wir eine erste Definition für einen Prozeß angeben, der die Gabeln $n_1$ und $n_2$ zu nehmen versucht und bei Erfolg zu einem neuen Prozeß $p$ wird. Dieser Prozeß $\mathrm{Nimm}$ hat noch zwei weitere dreiwertige Parameter $b_1$ und $b_2$, die angeben, ob die entsprechende Gabel auf dem Tisch liegt (Tisch), ob ein anderer Philosoph sie gerade hat (Phil), oder ob man selber sie hat (Selbst).

$$\mathrm{Nimm}_{n_1,n_2,b_1,b_2,p} := \begin{cases} f_{\mathrm{Nimm}_{n_1,n_2,b_1,b_2,p}}\&<n_1, \mathrm{Nimm}_{n_1,n_2,\mathrm{Selbst},b_2,p}> & \text{falls } b_1 = \mathrm{Tisch} \\ f_{\mathrm{Nimm}_{n_1,n_2,b_1,b_2,p}}\&<n_2, p> & \text{falls } b_1 = \mathrm{Selbst} \\ & \text{und } b_2 = \mathrm{Tisch} \\ ?f_{\mathrm{Nimm}_{n_1,n_2,b_1,b_2,p}} & \text{sonst} \end{cases}$$

$$f_{\mathrm{Nimm}_{n_1,n_2,b_1,b_2,p}}(v) := \begin{cases} \mathrm{Nimm}_{n_1,n_2,\mathrm{Phil},b_2,p} & v = n_1 \\ \mathrm{Nimm}_{n_1,n_2,b_1,\mathrm{Phil},p} & v = n_2 \\ \mathrm{Nimm}_{n_1,n_2,\mathrm{Tisch},b_2,p} & v = -n_1 \\ \mathrm{Nimm}_{n_1,n_2,b_1,\mathrm{Tisch},p} & v = -n_2 \end{cases}$$

Wenn also die Gabel Nr. $n_1$ auf dem Tisch liegt ($b_1 = \mathrm{Tisch}$), dann versucht der Philosoph, sie zu nehmen, und markiert sie dann als genommen. Wenn er sie schon hat ($b_1 = \mathrm{Selbst}$) und die Gabel Nr. $n_2$ auf dem Tisch liegt ($b_2 = \mathrm{Tisch}$), dann versucht er, sie zu nehmen und zu $p$ zu werden. Die Funktion $f_{\mathrm{Nimm}_{n_1,n_2,b_1,b_2,p}}$ schaut immer, was mit den beiden relevanten Gabeln passiert, ob sie aufgenommen werden ($v = n_{1,2}$) oder ob sie hingelegt werden ($v = -n_{1,2}$) und ändert die lokalen Daten $b_1$ und $b_2$ entsprechend.

Ein Philosoph könnte nun folgendermaßen repräsentiert werden.

$$\begin{aligned} \mathrm{Phil}_k &:= \mathrm{Nimm}_{k,k-1,\mathrm{Tisch},\mathrm{Tisch},\mathrm{Esse}_k} \text{ für } 2 \le k \le n \\ \mathrm{Phil}_1 &:= \mathrm{Nimm}_{1,n,\mathrm{Tisch},\mathrm{Tisch},\mathrm{Esse}_1} \\ \mathrm{Esse}_k &:= 1.!<-k, !<-(k-1), \mathbf{0}>> \text{ für } 2 \le k \le n \\ \mathrm{Esse}_1 &:= 1.!<-1, !<-n, \mathbf{0}>> \end{aligned}$$

Die Sonderrolle für den Philosophen Nr. 1 rührt davon her, daß er sich die Gabeln Nr. 1 und Nr. $n$ holen muß. Das Essen der Philosophen ist durch das Vergehenlassen von einer Zeiteinheit symbolisiert. Danach legen sie die Gabeln wieder hin, und werden zu $\mathbf{0}$. Lassen wir nun für $n = 3$ diese Philosophen miteinander speisen. Wir erhalten z.B. folgende Transitionsfolge:

$$
\begin{aligned}
& \mathrm{Phil}_1 \mid \mathrm{Phil}_2 \mid \mathrm{Phil}_3 \\
\xrightarrow{1!}\ & \mathrm{Nimm}_{1,3,\mathrm{Selbst},\mathrm{Tisch},\mathrm{Esse}_1} \mid \mathrm{Nimm}_{2,1,\mathrm{Tisch},\mathrm{Phil},\mathrm{Esse}_2} \mid \mathrm{Nimm}_{3,2,\mathrm{Tisch},\mathrm{Tisch},\mathrm{Esse}_3} \\
\xrightarrow{2!}\ & \mathrm{Nimm}_{1,3,\mathrm{Selbst},\mathrm{Tisch},\mathrm{Esse}_1} \mid \mathrm{Nimm}_{2,1,\mathrm{Selbst},\mathrm{Phil},\mathrm{Esse}_2} \mid \mathrm{Nimm}_{3,2,\mathrm{Tisch},\mathrm{Phil},\mathrm{Esse}_3} \\
\xrightarrow{3!}\ & \mathrm{Nimm}_{1,3,\mathrm{Selbst},\mathrm{Phil},\mathrm{Esse}_1} \mid \mathrm{Nimm}_{2,1,\mathrm{Selbst},\mathrm{Phil},\mathrm{Esse}_2} \mid \mathrm{Nimm}_{3,2,\mathrm{Selbst},\mathrm{Phil},\mathrm{Esse}_3}
\end{aligned}
$$

Jetzt kann man deutlich sehen, daß jeder Philosoph seine rechte Gabel in der Hand hält, seine linke aber nicht in die Hand nehmen kann, da sie der links von ihm sitzenden Philosoph als seine rechte bereits in der Hand hat. Die Philosophen werden also verhungern. Dieses Phänomen hat den Namen Deadlock. Die Prozesse verklemmen sich bei ihrem Lauf so, daß eine weitere Ausführung unmöglich wird.

Das Interessante daran ist, daß man den Philosophen dieses Verhalten nicht ohne weiteres ansieht. Wenn man so etwas programmierte, würde man eben erst beim Lauf des Programmes davon überrascht werden. Und tatsächlich ist es so, daß der Deadlock z.B. in der Betriebssystemprogrammierung lange Zeit ein gefürchteter Fehler war, weil er an den Einzelprogrammen schwer zu entdecken ist. Heutzutage gibt zwar einige Vorschriften, die einen Deadlock verhindern, doch andere unerwünschte Phänomene, wie ein Lifelock, sind nur schwer schon während der Programmierung zu entdecken. Beim Lifelock käme es in unserem Beispiel dazu, daß ein Philosoph verhungerte, obwohl andere immer wieder, aber nicht ständig, essen.

Wie verhindern wir aber nun, daß die Philosophen verhungern? Das wird am liebsten mit Hilfe einer Asymmetrie in der Reihenfolge der Aufnahme der Gabeln gemacht. So nimmt z.B. der Philosoph Nr. 1 erst die linke und dann die rechte Gabel. Das würde in der Definition so aussehen:

$$\mathrm{Phil}_1 := \mathrm{Nimm}_{n,1,\mathrm{Tisch},\mathrm{Tisch},\mathrm{Esse}_1}$$

Jetzt können wir uns noch einmal eine mögliche Transitionsfolge ansehen:

$$
\begin{aligned}
&\mathrm{Phil}_1 \,|\, \mathrm{Phil}_2 \,|\, \mathrm{Phil}_3 \\
\xrightarrow{3!}\ &\mathrm{Nimm}_{3,1,\mathrm{Selbst},\mathrm{Tisch},\mathrm{Esse}_1} \,|\, \mathrm{Nimm}_{2,1,\mathrm{Tisch},\mathrm{Tisch},\mathrm{Esse}_2} \,|\, \mathrm{Nimm}_{3,2,\mathrm{Phil},\mathrm{Tisch},\mathrm{Esse}_3} \\
\xrightarrow{2!}\ &\mathrm{Nimm}_{3,1,\mathrm{Selbst},\mathrm{Tisch},\mathrm{Esse}_1} \,|\, \mathrm{Nimm}_{2,1,\mathrm{Selbst},\mathrm{Tisch},\mathrm{Esse}_2} \,|\, \mathrm{Nimm}_{3,2,\mathrm{Phil},\mathrm{Phil},\mathrm{Esse}_3} \\
\xrightarrow{1!}\ &\mathrm{Esse}_1 \,|\, \mathrm{Nimm}_{2,1,\mathrm{Selbst},\mathrm{Phil},\mathrm{Esse}_2} \,|\, \mathrm{Nimm}_{3,2,\mathrm{Phil},\mathrm{Phil},\mathrm{Esse}_3} \\
\xrightarrow{1:}\ &!<-1,!<-3,\mathbf{0}>>|\, \mathrm{Nimm}_{2,1,\mathrm{Selbst},\mathrm{Phil},\mathrm{Esse}_2} \,|\, \mathrm{Nimm}_{3,2,\mathrm{Phil},\mathrm{Phil},\mathrm{Esse}_3} \\
\xrightarrow{-1!}\ &!<-3,\mathbf{0}>|\, \mathrm{Nimm}_{2,1,\mathrm{Selbst},\mathrm{Tisch},\mathrm{Esse}_2} \,|\, \mathrm{Nimm}_{3,2,\mathrm{Phil},\mathrm{Phil},\mathrm{Esse}_3} \\
\xrightarrow{-3!}\ &\mathbf{0}|\, \mathrm{Nimm}_{2,1,\mathrm{Selbst},\mathrm{Tisch},\mathrm{Esse}_2} \,|\, \mathrm{Nimm}_{3,2,\mathrm{Tisch},\mathrm{Phil},\mathrm{Esse}_3} \\
\xrightarrow{1!}\ &\mathbf{0}|\, \mathrm{Esse}_2 \,|\, \mathrm{Nimm}_{3,2,\mathrm{Tisch},\mathrm{Phil},\mathrm{Esse}_3} \\
\xrightarrow{1:}\ &\mathbf{0}|!<-2,!<-1,\mathbf{0}>>|\, \mathrm{Nimm}_{3,2,\mathrm{Tisch},\mathrm{Phil},\mathrm{Esse}_3} \\
\xrightarrow{-2!}\ &\mathbf{0}|!<-1,\mathbf{0}>|\, \mathrm{Nimm}_{3,2,\mathrm{Tisch},\mathrm{Tisch},\mathrm{Esse}_3} \\
\xrightarrow{-1!}\ &\mathbf{0}|\mathbf{0}|\, \mathrm{Nimm}_{3,2,\mathrm{Tisch},\mathrm{Tisch},\mathrm{Esse}_3} \\
\xrightarrow{3!}\ &\mathbf{0}|\mathbf{0}|\, \mathrm{Nimm}_{3,2,\mathrm{Selbst},\mathrm{Tisch},\mathrm{Esse}_3} \\
\xrightarrow{2!}\ &\mathbf{0}|\mathbf{0}|\, \mathrm{Esse}_3 \\
\xrightarrow{1:}\ &\mathbf{0}|\mathbf{0}|!<-3,!<-2,\mathbf{0}>> \\
\xrightarrow{-3!}\ &\mathbf{0}|\mathbf{0}|!<-2,\mathbf{0}> \\
\xrightarrow{-2!}\ &\mathbf{0}|\mathbf{0}|\mathbf{0}
\end{aligned}
$$

Nun sind alle Philosophen satt geworden. Im allgemeinen setzen die Philosophen, nachdem sie gegessen haben, nach einer kurzen Ruhepause wieder mit dem Essen fort. Hier wurde zur besseren Übersichtlichkeit des Algorithmus darauf verzichtet. Außerdem ist die obige Folge kein Beweis dafür, daß es keinen Deadlock gibt, man müßte dafür alle möglichen untersuchen. Das bestätigt noch einmal, wie wichtig ein formaler Beweis für die Funktionstüchtigkeit eines parallelen Algorithmus ist. Ein erfolgreicher Probelauf, wie eben, sichert nicht in jedem Falle, daß alle möglichen Läufe erfolgreich wären. So könnte man für die erste Definition der Philosophen auch ohne Probleme einen Lauf finden, der alle essen läßt, indem erst Philosoph 1 beide Gabeln nehmen würde, dann Philosoph 2 usw.

## 2.8 Schlußbemerkungen

Wie in den bisher angeführten Beispielen deutlich geworden ist, bezieht TCBS wie viele andere parallele Kalküle seine Stärke in der Programmierung aus der rekursiven Definition von Prozeßkonstanten. Alle bisher aufgeführten Prozeßkonstanten wurden immer autorekursiv definiert. Entweder, indem in der Definition der Prozeßkonstante $X_n$ wieder das $X_n$ (vielleicht mit verändertem Speicher, also in diesem Fall $n$) auftauchte, oder über die Funktionen $f$ von der Sprache $\alpha$ in die Menge aller Prozesse $P$, wie sie im Hear- und Talk-Operator auftreten. Ohne diese Rekursion wären viele der obigen Beispiele nur schwer, andere überhaupt nicht implementierbar. Es gibt jedoch auch Probleme mit dieser Rekursivität. Das erste ist, daß man in einem Prozeß wie

$X :=!<,1, X>$ nicht sofort $X$ durch seine Definition ersetzten darf, weil das in einer unendlich langen Kette enden würde. Das zweite Problem ist noch gravierender. Man muß verhindern, daß Definitionen der Form $X := X$ vorkommen, denn über diese kann man keine Aussage bezüglich des Verhaltens machen. Beide Ziele sind bei der Definition der Syntax und Semantik zu beachten und verkomplizieren diese auch erheblich. Durch Vermeidung dieser Probleme jedoch entsteht ein sauberer Kalkül, über dem formale Beweise möglich sind, die ansonsten schwer oder unmöglich wären.

Berechenbarkeits und Komplexitätsuntersuchungen waren bei der Entwicklung dieses parallelen Kalküls nicht das primäre Ziel. Solche Betrachtungen kann man aber natürlich für die Funktionen $f$ des Talk-Operators sowie die Definitionen der Prozeßkonstanten durchführen. Sie bilden das aus der Berechenbarkeits- und Komplexitätstheorie entlehnte Rückgrat dieses Kalküls, d.h. um z.B. zu bestimmen, ob gewisse Probleme parallel in polynomialer Zeit lösbar sind, muß man auf Kenntnisse aus der klassischen Theorie zurückgreifen. Auf Probleme solcher Art wird jedoch im folgenden nicht näher eingegangen, da sich diese Arbeit auf Untersuchungen der Korrektheit von Programmen beschränkt.

# 3 Der parallele Kalkül TCBS

## 3.1 Grammatiken

Im diesem Abschnitt werden Grammatiken und Sprachen eingeführt, um TCBS formal definieren zu können. Dabei folgt diese Arbeit mit wenigen Ausnahmen [BB93].

**Definition 3.1 (Alphabet)**
***Ein Alphabet ist eine endliche, nicht leere Menge von Symbolen.***

**Definition 3.2 (Wort)**
***Sei*** $w$ ***eine Folge von*** $k$ ***Symbolen aus dem Alphabet*** $A$***.*** $w \in A^k$ ***heißt ein Wort der Länge*** $k$ ***über dem Alphabet*** $A$***. Außerdem sei*** $A^0 := \{\epsilon\}$ ***(*** $\epsilon$ ***ist das leere Wort) und*** $A^* := \bigcup_{k \in \mathbb{N}} A^k$***.***

**Definition 3.3 (Sprache)**
***Sei*** $A$ ***ein Alphabet, dann heißt*** $L \subseteq A^*$ ***eine Sprache.***

**Definition 3.4 (Grammatiken)**
***Eine Grammatik*** $G$ ***ist gegeben durch ein 4-Tupel***

$$G = (N, \Sigma, P, S),$$

***wobei folgendes gilt:***

- $N$ ***und*** $\Sigma$ ***sind zueinander disjunkte Alphabete (***$N \cap \Sigma = \emptyset$***).*** $N$ ***heißt Nonterminalalphabet,*** $\Sigma$ ***heißt Terminalalphabet,*** $X := N \cup \Sigma$ ***heißt Totalalphabet.***
- $P \subseteq X^*NX^* \times X^*$ ***ist eine endliche Menge, die Menge der Ableitungsregeln oder Produktionen. Anstelle von*** $(v, w) \in P$ ***schreibt man auch*** $v \underset{P}{\longrightarrow} w$***.***
- $S$ ***ist ein ausgezeichnetes Nonterminal (***$S \in N$***) und heißt Startsymbol.***

**Definition 3.5 (von Grammatiken erzeugte Sprachen)**
***Sei*** $G = (N, \Sigma, P, S)$ ***eine Grammatik,*** $X := N \cup \Sigma$***. Auf*** $X^*$ ***werden nun die Ableitungsregeln*** $\xrightarrow[P]{n}$ ***und*** $\xrightarrow[P]{*}$ ***definiert:***

$$
\begin{aligned}
v \xrightarrow[P]{1} w &:\Longleftrightarrow \exists \alpha, \beta, v_1, v_2 \in X^* : v = v_1 \alpha v_2 \ \wedge \ w = v_1 \beta v_2 \ \wedge \ \alpha \underset{P}{\longrightarrow} \beta) \\
v \xrightarrow[P]{0} w &:\Longleftrightarrow v = w \\
v \xrightarrow[P]{n+1} w &:\Longleftrightarrow \exists w' \in X^* : v \xrightarrow[P]{n} w' \ \wedge \ w' \xrightarrow[P]{1} w \\
w \xrightarrow[P]{*} v &:\Longleftrightarrow \exists n \in \mathbb{N} : v \xrightarrow[P]{n} w
\end{aligned}
$$

***Die von*** $G$ ***erzeugte Sprache ist nun:***

$$L(G) := \left\{ w \ \middle| \ w \in \Sigma^* \ \wedge \ S \xrightarrow[P]{*} w \right\}$$

**Definition 3.6 (Tiefe eines Wortes)**
***Sei*** $w \in L(G)$ ***ein Wort der durch die Grammatik*** $G$ ***erzeugten Sprache. Dann ist***

$$\mathrm{Depth}(w) := \min_{S \xrightarrow[P]{n} w} (n)$$

***die Ableitungstiefe des Wortes*** $w$ ***bezüglich der Grammatik*** $G$ ***oder kürzer die Tiefe des Wortes*** $w$***.***

Die Tiefe eines Wortes ist ein praktischer Induktionsparameter. Im übrigen ist die Ableitung eines Wortes in den meisten für die praktische Programmierung gebrauchten Grammatiken eindeutig. Dann braucht natürlich kein Minimum gebildet zu werden.

Die hier angegebenen Ableitungspfeile $\underset{P}{\longrightarrow}$ betreffen ausschließlich die Syntax und haben mit den Transitionspfeilen wie z.B. $\xrightarrow{\delta:}$, die die Semantik betreffen, nichts zu tun.

## 3.2 Die Syntax von TCBS

In diesem Abschnitt definiere ich die Syntax von TCBS. Dabei gehe ich im Gegensatz zur Originalarbeit [Pra96] nach der in der theoretischen Informatik gut ausgebauten Theorie der Grammatiken vor, die im letzten Abschnitt kurz eingeführt wurde. K.V.S. Prasad definiert TCBS etwas kürzer, aber auch etwas weniger formal, was der Beweisbarkeit von Sätzen über diesem Kalkül schadet. Er führt in seiner Arbeit aber auch keine formalen Beweise durch, da die Sätze einleuchtend klingen und der Schwerpunkt auf der Vorstellung dieses neuen Kalküls liegt.

Wie gezeigt wurde, gibt es Prozesse über mehreren Sprachen. Diese Sprachen, alles endliche Mengen, nennen wir Datentypen und fassen sie zur endlichen Mengenfamilie $\Gamma$ zusammen. Dann kann die Menge aller Prozesse, die die Sprache $\alpha \in \Gamma$ sprechen, mit $\mathbf{P}_\alpha$ bezeichnet werden. Wir wollen nun einen ersten Versuch unternehmen, Ersetzungsregeln für die Erzeugung der Menge $\mathbf{P}_\alpha$ anzugeben. Die Menge der Nichtterminale $N$ besteht auf dem ersten Blick nur aus den Repräsentanten für Prozesse, für jedes $\alpha \in \Gamma$ einer. Sie bilden auch die Startsymbole $S$ für die Sprachen $\mathbf{P}_\alpha$. Wir nennen sie $\mathbf{p}_\alpha$. Sei $P$ nun die Menge der Produktionen. Die Ersetzungsregeln für den Parallelitätsoperator lauten nun naheliegend:

$$\mathbf{p}_\alpha \underset{P}{\longrightarrow} (\mathbf{p}_\alpha | \mathbf{p}_\alpha) \ (\alpha \in \Gamma).$$

Die Klammern sind notwendig, da wir noch nicht von der Assoziativität des Compose-Operators ausgehen können. Die dafür bisher notwendigen Terminalsymbole aus $\Sigma$ sind der Strich „ | “, sowie die beiden Klammern „ ( “ und „ ) “. Wenn aber nun der Talk-Operator hinzugenommen werden soll, treten auch noch andere Terminalsymbole auf. Insbesondere muß dasjenige Element aus $\alpha$, das der Talk-Operator sagen will, in $\Sigma$ enthalten sein. Also muß für alle $\alpha \in \Gamma$ gelten, daß $\alpha \subset \Sigma$. Die Elemente aus $\alpha$ dienen also als Konstanten in der TCBS. Außerdem muß die Funktion, die, wenn der Talk-Operator hört, angibt, wie sich der Prozeß entwickelt, noch angegeben werden. Da wir zum Zeitpunkt der Definition von $\mathbf{P}_\alpha$ noch keine Funktionen haben, die in $\mathbf{P}_\alpha$ abbilden können, müssen wir als erstes Platzhalter für Funktionen dieser Art einführen. Diese liegen für die Funktionen, die von $\alpha$ nach $\mathbf{P}_\alpha$ abbilden, in der Menge $\mathrm{Func}_\alpha$. Dabei kann natürlich nicht für jede solche Funktion ein Symbol existieren, da es nicht nur endlich viele solche Funktionen gibt. Die Ersetzungsregeln für den Talk-Operator lauten jetzt wie folgt:

$$\mathbf{p}_\alpha \underset{P}{\longrightarrow} f\& < v, \mathbf{p}_\alpha > \ (\alpha \in \Gamma; f \in \mathrm{Func}_\alpha; v \in \alpha).$$

Neu hinzugekommene Terminalsymbole sind also das Kaufmanns-Und „ & “, die spitzen Klammern „ < “ und „ > “, das Komma „ , “, sowie alle Elemente der Mengen $\alpha$ und alle Elemente der Mengen $\mathrm{Func}_\alpha$. Die Ersetzungsregeln für den Hear-Operator, den Delay-Operator und den Timeout-Operator sind nun klar:

$$\mathbf{p}_\alpha \underset{P}{\longrightarrow} ?f \ (\alpha \in \Gamma; f \in \mathrm{Func}_\alpha),$$
$$\mathbf{p}_\alpha \underset{P}{\longrightarrow} \delta.\mathbf{p}_\alpha \ (\alpha \in \Gamma; \delta \in Z),$$
$$\mathbf{p}_\alpha \underset{P}{\longrightarrow} \delta{:}\mathbf{p}_\alpha \ (\alpha \in \Gamma; \delta \in Z).$$

Neu zu $\Sigma$ hinzukommende Symbole sind hier das Fragezeichen „ ? “, der Punkt „ . “, der Doppelpunkt „ : “ sowie eigentlich alle natürlichen Zahlen. Da aber $\Sigma$ eine endliche Menge sein muß, kann $\delta$ nur aus einer endlichen Teilmenge der natürlichen Zahlen $\mathbb{N}$ stammen. Diese endliche Teilmenge bezeichnen wir mit $Z$. Der einzige noch nicht in dieser Syntax definierte Operator ist der Translateoperator, auch für ihn werden erst einmal wie bei den Funktionen des Talk- und Hear-Operators nur Platzhalter eingeführt. Ein Symbol aus der Menge $\mathrm{Trans}_{\alpha,\beta}$ repräsentiert dann einen Übersetzer, der innen die Sprache $\beta$ und außen die Sprache $\alpha$ spricht. Wenn man nun für alle $\alpha$ und $\beta$ die Elemente der Menge $\mathrm{Trans}_{\alpha,\beta}$ zu $\Sigma$ hinzunimmt, dann lauten die entsprechenden Ersetzungsregeln:

$$\mathbf{p}_\alpha \underset{P}{\longrightarrow} \Phi\mathbf{p}_\beta \ (\alpha, \beta \in \Gamma; \Phi \in \mathrm{Trans}_{\alpha,\beta}).$$

Nun sind wir aber noch nicht auf die Prozeßkonstanten eingegangen. Diese bilden zusammen mit den Funktionen $f : \alpha \to \mathbf{P}_\alpha$ die Grundlage der rekursiven Definition, die eine der Stärken von TCBS ausmacht. Für sie werden wieder nur Platzhalter eingeführt. Prozeßkonstanten, die Prozesse aus $\mathbf{P}_\alpha$ repräsentieren, sind in der Menge $\mathrm{Def}_\alpha$. Die entsprechenden Ersetzungsregeln lauten

$$\mathbf{p}_\alpha \underset{P}{\longrightarrow} A \ (\alpha \in \Gamma; A \in \mathrm{Def}_\alpha).$$

Elementen aus $\mathrm{Def}_\alpha$ werden dann später Prozesse aus $\mathbf{P}_\alpha$ zugeordnet. Dabei kann es jedoch zu unerwünschten Phänomenen kommen, wenn z.B. $X \in \mathrm{Def}_\alpha$ eine Prozeßkonstante ist und die zugehörige Definition auch $X$ lautet, denn aus $A \in \mathrm{Def}_\alpha$ folgt ja, daß $A \in \mathbf{P}_\alpha$. Dann ist aber über dieses $X$ keine Aussage bezüglich der Kommunikationsaktionen möglich. Man kann

auch sagen, daß die Gleichung $X = X$ keine eindeutige Lösung hat. Um sicherzustellen, daß so etwas nicht auftreten kann, müssen außer $\mathbf{P}_\alpha$ noch die Mengen $\mathbf{D}_\alpha$ eingeführt werden (siehe dazu Satz 3.3 auf Seite 32 und die anschließende Diskussion). Elemente aus ihnen haben die Eigenschaft, daß ihr Verhalten ohne Kenntnis des Verhaltens eventuell enthaltener Prozeßkonstanten bestimmbar ist. Sie heißen überwachte Prozesse (in der englischsprachigen Fachliteratur heißen sie „guarded sums“, siehe [Pra95] und [Mil89]). Diesen Mengen $\mathbf{D}_\alpha$ werden die Nichtterminalsymbole $\mathbf{d}_\alpha$ zugeordnet. Jeder überwachte Prozeß ist auch ein wieder ein normaler Prozeß ($\mathbf{p}_\alpha \underset{P}{\longrightarrow} \mathbf{d}_\alpha$). Die oben mit $\mathbf{p}_\alpha$ angegebenen Ersetzungsregeln für den Compose-, Translate-, Delay- und Timeout-Operator gelten nun auch für $\mathbf{d}_\alpha$ anstelle von $\mathbf{p}_\alpha$, da so kombinierte überwachte Prozesse wieder überwachte Prozesse liefern. Die Regeln für die Verwendung von Prozeßkonstanten $\mathbf{p}_\alpha \underset{P}{\longrightarrow} A$ $(\alpha \in \Gamma; A \in \mathrm{Def}_\alpha)$ bleiben so erhalten.

Vollständig geändert werden nur die Regeln für den Hear- und den Talk-Operator. Sie geben uns nämlich die Möglichkeit, zu entscheiden, wie sie sich verhalten werden, ohne enthaltene Prozeßkonstanten auszuwerten. Der Talk-Operator, der einen normalen Prozeß umschließt, sowie der Hear-Operaor sind also in jedem Falle überwachte Prozesse. Daher sind für sie ausschließlich folgende Regeln zuständig:

$$\mathbf{d}_\alpha \underset{P}{\longrightarrow} f\& < v, \mathbf{p}_\alpha > \ (\alpha \in \Gamma; f \in \mathrm{Func}_\alpha; v \in \alpha,$$
$$\mathbf{d}_\alpha \underset{P}{\longrightarrow} ?f \ (\alpha \in \Gamma; f \in \mathrm{Func}_\alpha).$$

Nun ist auch die folgende Definition der Syntax von TCBS zu verstehen. In ihr sind Buchstaben eines Alphabets zur besseren Übersichtlichkeit grau hinterlegt. Die auftretenden Disjunktheitsforderungen dienen dazu, die Zuordnung der Semantik eindeutig durchführen zu können, daß also z.B. Platzhalter nur dann gleiche Namen haben, wenn aus dem Kontext zu schließen ist, welcher gemeint ist.

**Definition 3.7 (Die Menge aller TCBS-Prozesse)**
***Sei** $\Gamma$ **eine endliche Menge von endlichen Mengen, Datentypen genannt. Für jede dieser Mengen** $\gamma \in \Gamma$ **gelte:** $\tau \notin \gamma$ **und** $\gamma_\tau := \gamma \cup \{\tau\}$**. Für alle** $\alpha \in \Gamma$ **seien jetzt** $\mathrm{Func}_\alpha$ **und** $\mathrm{Def}_\alpha$ **beliebige Alphabete, dabei heißt** $\mathrm{Func}_\alpha$ **Menge der** $\alpha$**-Prozeßfunktionen und** $\mathrm{Def}_\alpha$ **Menge der** $\alpha$**-Prozeßdefinitionen. Für alle** $\alpha, \beta \in \Gamma$ **sei** $\mathrm{Trans}_{\alpha,\beta}$ **ein Alphabet.** $\mathrm{Trans}_{\alpha,\beta}$ **heißt Menge der** $\alpha, \beta$**-Übersetzer. Für alle** $\alpha_1, \alpha_2 \in \Gamma$ **gelte:** $\mathrm{Func}_{\alpha_1} \cap \mathrm{Func}_{\alpha_2} = \emptyset$ **und** $\mathrm{Def}_{\alpha_1} \cap \mathrm{Def}_{\alpha_2} = \emptyset$**. Für alle** $\alpha_1, \alpha_2, \beta_1, \beta_2 \in \Gamma$ **gelte:** $\mathrm{Trans}_{\alpha_1,\beta_1} \cap \mathrm{Trans}_{\alpha_2,\beta_2} = \emptyset$**. Außerdem sei** $Z \subset \mathbb{N}$ **eine endliche Teilmenge der natürlichen Zahlen.***

***Jetzt ist***

$$\mathrm{Trans}_\alpha := \bigcup_{\beta \in \Gamma} \mathrm{Trans}_{\alpha,\beta}$$
$$N := \left\{ \mathbf{d}_\alpha, \mathbf{p}_\alpha \mid \alpha \in \Gamma \right\}$$
$$\Sigma := \bigcup_{\alpha \in \Gamma} (\alpha \cup \mathrm{Func}_\alpha \cup \mathrm{Def}_\alpha \cup \mathrm{Trans}_\alpha) \cup Z \cup$$
$$\{(, ), |, ?, \&, <, ,, >, ., :\}$$

***Nun wird die Menge der Ersetzungsregeln*** $P$ ***definiert. Dabei ist*** $\alpha, \beta \in \Gamma$***,*** $\delta \in Z$***,*** $v \in \alpha$***,*** $f \in \mathrm{Func}_\alpha$***,*** $\Phi \in \mathrm{Trans}_{\alpha,\beta}$ ***sowie*** $A \in \mathrm{Def}_\alpha$***. Außerdem sind die Namen der Operatoren zur besseren Verständlichkeit in den Spalten angegeben (siehe hierzu die Erläuterungen zur Syntax auf Seite 21 sowie zur Semantik auf Seite 22).***

| **Operator** | $\mathbf{d}_\alpha$ | $\mathbf{p}_\alpha$ |
|---|---|---|
| **Hear** | $\mathbf{d}_\alpha \underset{P}{\longrightarrow} ?\, f$ | |
| **Talk** | $\mathbf{d}_\alpha \underset{P}{\longrightarrow} f \,\&\, <\! v \,,\, \mathbf{p}_\alpha \!>$ | |
| **Compose** | $\mathbf{d}_\alpha \underset{P}{\longrightarrow} (\, \mathbf{d}_\alpha \mid \mathbf{d}_\alpha \,)$ | $\mathbf{p}_\alpha \underset{P}{\longrightarrow} (\, \mathbf{p}_\alpha \mid \mathbf{p}_\alpha \,)$ |
| **Translate** | $\mathbf{d}_\alpha \underset{P}{\longrightarrow} \Phi\, \mathbf{d}_\beta$ | $\mathbf{p}_\alpha \underset{P}{\longrightarrow} \Phi\, \mathbf{p}_\beta$ |
| **Delay** | $\mathbf{d}_\alpha \underset{P}{\longrightarrow} \delta\,.\, \mathbf{d}_\alpha$ | $\mathbf{p}_\alpha \underset{P}{\longrightarrow} \delta\,.\, \mathbf{p}_\alpha$ |
| **Timeout** | $\mathbf{d}_\alpha \underset{P}{\longrightarrow} \delta\,:\, \mathbf{d}_\alpha$ | $\mathbf{p}_\alpha \underset{P}{\longrightarrow} \delta\,:\, \mathbf{p}_\alpha$ |
| **Define** | | $\mathbf{p}_\alpha \underset{P}{\longrightarrow} A$ |
| | | $\mathbf{p}_\alpha \underset{P}{\longrightarrow} \mathbf{d}_\alpha$ |

***Die erzeugten Sprachen*** $\mathbf{P}_\alpha$ ***und*** $\mathbf{D}_\alpha$ ***werden dann wie folgt definiert:***

$$\mathbf{P}_\alpha := L\left(N, \Sigma, P, \mathbf{p}_\alpha\right)$$
$$\mathbf{D}_\alpha := L\left(N, \Sigma, P, \mathbf{d}_\alpha\right)$$

$\mathbf{P}_\alpha$ ***heißt nun Menge aller TCBS-Prozesse über*** $\alpha$***.*** $\mathbf{D}_\alpha$ ***heißt Menge aller überwachten TCBS-Prozesse über*** $\alpha$***.***

Um nun einen ausführbaren Prozeß zu erhalten, müssen noch die Mengen $\mathrm{Func}_\alpha$, $\mathrm{Def}_\alpha$ und $\mathrm{Trans}_{\alpha,\beta}$ mit einer Bedeutung versehen werden. Das geschieht in der folgenden Definition:

**Definition 3.8 (Ein ausführbarer TCBS-Prozeß)**
***Das 4-Tupel***

$$(\mathbf{P}_\alpha, \mathrm{Ins}_{\mathrm{Func}}, \mathrm{Ins}_{\mathrm{Def}}, \mathrm{Ins}_{\mathrm{Trans}})$$

***heißt ausführbarer TCBS-Prozeß, wenn folgendes gilt:***

- ***Für alle*** $\alpha \in \Gamma$***, alle*** $f \in \mathrm{Func}_\alpha$ ***und alle*** $v \in \alpha$ ***gilt:*** $\mathrm{Ins}_{\mathrm{Func}}(f, v) \in \mathbf{P}_\alpha$ ***Die abkürzende Schreibweise hierfür ist:*** $f(v) \in \mathbf{P}_\alpha$
- ***Für alle*** $\alpha \in \Gamma$***, alle*** $A \in \mathrm{Def}_\alpha$ ***gilt:*** $\mathrm{Ins}_{\mathrm{Def}}(A) \in \mathbf{D}_\alpha$ ***Die abkürzende Schreibweise hierfür ist:*** $A() \in \mathbf{D}_\alpha$
- ***Für alle*** $\alpha, \beta \in \Gamma$***, alle*** $\Phi \in \mathrm{Trans}_{\alpha,\beta}$***, alle*** $w_\alpha \in \alpha_\tau$ ***und alle*** $w_\beta \in \beta_\tau$ ***gilt:*** $\mathrm{Ins}_{\mathrm{Trans}}(\textbf{\textit{Down}}, \Phi, w_\alpha) \in \beta_\tau$ ***und*** $\mathrm{Ins}_{\mathrm{Trans}}(\textbf{\textit{Up}}, \Phi, w_\beta) \in \alpha_\tau$ ***Die abkürzende Schreibweise hierfür ist:*** $\Phi_\downarrow(w_\alpha) \in \beta_\tau$ ***bzw.*** $\Phi_\uparrow(w_\beta) \in \alpha_\tau$ ***Außerdem gelte:*** $\Phi_\downarrow(\tau) = \Phi_\uparrow(\tau) = \tau$

Elemente der Menge der $\alpha$-Prozeßfunktionen $\mathrm{Func}_\alpha$ dienen dazu, auf dem Medium gesprochene $v \in \alpha$ zu verarbeiten, wohingegen Elemente von $\mathrm{Def}_\alpha$, der Menge der $\alpha$-Prozeßdefinitionen, andere Prozesse repräsentieren, die für sie eingesetzt werden sollen. $\alpha, \beta$-Übersetzer aus $\mathrm{Trans}_{\alpha,\beta}$ repräsentieren die Übersetzer.

Die Funktionen $\mathrm{Ins}$ sind ein Hilfskonstrukt, um jedem Element der entsprechenden Mengen $\mathrm{Func}_\alpha$, $\mathrm{Def}_\alpha$ und $\mathrm{Trans}_{\alpha,\beta}$ die entsprechenden Objekte zuordnen zu können. Die Mengen sind ja Teilmengen des Alphabets, wohingegen die zugeordneten Funktionen, Prozesse bzw. Funktionenpaare andere mathematische Objekte sind. Da die Entsprechung zwischen Bezeichnung (z.B. $A \in \mathrm{Def}_\alpha$) und Bedeutung (dazugehörig: $\mathrm{Ins}_{\mathrm{Def}}(A)$) klar ersichtlich ist, werden im folgenden vorwiegend die abkürzenden Schreibweisen verwendet, sowohl in Definitionen als auch in der Anwendung. Außerdem wird angenommen, daß alle undefinierten Terminalsymbole der Mengen $\mathrm{Func}_\alpha$, $\mathrm{Def}_\alpha$ und $\mathrm{Trans}_{\alpha,\beta}$, um einen ausführbaren TCBS-Prozeß zu erhalten, auf folgende Art und Weise definiert sind:

- Für alle $\alpha \in \Gamma$ sei $f_{\mathbf{0}_\alpha} \in \mathrm{Func}_\alpha$ und für alle $v \in \alpha$ sei $f_{\mathbf{0}_\alpha}(v) := \mathbf{0}_\alpha$
- Für alle $\alpha \in \Gamma$ sei $\mathbf{0}_\alpha \in \mathrm{Def}_\alpha$ und $\mathbf{0}_\alpha :=? f_{\mathbf{0}_\alpha}$
- Alle undefinierten Symbole $f \in \mathrm{Func}_\alpha$ werden mit $f(v) := \mathbf{0}_\alpha$ für alle $v \in \alpha$ definiert
- Alle undefinierten Symbole $A \in \mathrm{Def}_\alpha$ werden mit $A() := \mathbf{0}_\alpha$ definiert
- Alle undefinierten Symbole $\Phi \in \mathrm{Trans}_{\alpha,\beta}$ werden mit $\Phi_\uparrow(w_\alpha) := \tau$ für alle $w_\alpha \in \alpha_\tau$ und $\Phi_\downarrow(w_\beta) := \tau$ für alle $w_\beta \in \beta_\tau$ definiert.

Der Prozeß $\mathbf{0}_\alpha$, der Nil-Operator, wird genauer im Abschnitt 3.6 auf Seite 27 eingeführt. Ab jetzt werden die Mengen $\mathrm{Func}_\alpha$, $\mathrm{Def}_\alpha$, $\mathrm{Trans}_{\alpha,\beta}$ und $Z$ sowie die Funktionen $\mathrm{Ins}_{\mathrm{Func}}$, $\mathrm{Ins}_{\mathrm{Def}}$ und $\mathrm{Ins}_{\mathrm{Trans}}$ nur noch implizit betrachtet. Sie werden als existent und wohldefiniert vorausgesetzt und seien im übrigen ausreichend groß, aber endlich.

Für die einzelnen Konstruktoren in der Syntaxdefinition werden jetzt noch einmal zusammenfassend die Namen und Schreibweisen dargestellt:

- $?f$ mit $f \in \mathrm{Func}_\alpha$ heißt ein hörender Prozeß (Hear-Operator).
- $f\&<v,p>$ mit $f \in \mathrm{Func}_\alpha$, $v \in \alpha$ und $p \in \mathbf{P}_\alpha$ heißt ein sich unterhaltender Prozeß (Talk-Operator).
- $p|q$ mit $p, q \in \mathbf{P}_\alpha$ heißt eine parallele Komposition zweier Prozesse (Parallel Composition-Operator oder kürzer Compose-Operator).
- $A$ mit $A \in \mathrm{Def}_\alpha$ heißt eine Definitionseinsetzung oder auch eine Prozeßkonstante (Define-Operator).
- $\Phi p_\beta$ mit $\Phi \in \mathrm{Trans}_{\alpha,\beta}$ und $p_\beta \in \mathbf{P}_\beta$ heißt ein übersetzter Prozeß. Hierbei heißt $\Phi$ der Übersetzer (Translate-Operator).
- $\delta.p$ mit $p \in \mathbf{P}_\alpha$ und $\delta \in Z$ heißt ein genau $\delta$ wartender Prozeß (Delay-Operator).
- $\delta{:}p$ mit $p \in \mathbf{P}_\alpha$ und $\delta \in Z$ heißt ein höchstens $\delta$ wartender Prozeß (Timeout-Operator).

Diese Bezeichnungen deuten bereits die Semantik der Sprache an, sind aber keine Definition für diese. Im nächsten Abschnitt wird die Semantik dann anschaulich und formal eingeführt.

Jetzt wird das zu Beginn angegebene Beispiel der Bestimmung der ältesten Person noch einmal vollständig als ausführbarer Prozeß dargestellt. Dafür sei $N \subset \mathbb{N}$ eine endliche Menge und $N \in \Gamma$. Außerdem gelte für alle $n \in N$, daß $\mathrm{Cell}_n \in \mathrm{Def}_N$ und $f_{\mathrm{Cell}_n} \in \mathrm{Func}_N$. Nun sei:

$$\mathrm{Cell}_n := f_{\mathrm{Cell}_n}\&<n, \mathbf{0}_N>$$

$$f_{\mathrm{Cell}_n}(m) := \begin{cases} \mathbf{0}_N & m \geq n \\ \mathrm{Cell}_n & \text{sonst} \end{cases}$$

Jetzt ist z.B. $(((\mathrm{Cell}_1 \,|\, \mathrm{Cell}_8)|\, \mathrm{Cell}_6)|\, \mathrm{Cell}_3) \in \mathbf{P}_N$ ein Prozeß, der als letztes die $8$, also die höchste der Zahlen der Menge $\{1, 8, 6, 3\}$ sagt. Dazu mehr, wenn die Semantik eingeführt ist.

In obiger Definition des Beispiels sind die Elemente der Mengen $\mathrm{Func}_\alpha$ und $\mathrm{Def}_\alpha$ noch explizit angegeben worden. Das wird im weiteren weggelassen. Elemente gelten als in den Mengen befindlich, wenn sie definiert und/oder angewendet werden. Die einzige Forderung ist die Endlichkeit und die paarweise Disjunktheit der Mengen. Beide Eigenschaften sind in diesem Kalkül mehr technischer Natur.

## 3.3 Die Semantik von TCBS

Nun werden alle Konstruktoren von TCBS aus Definition 3.7 auf Seite 19 noch einmal zusammenhängend anschaulich erklärt.

- $?f$ ist ein Prozeß, der darauf wartet, daß etwas ($v \in \alpha$) gesagt wird, und der sich dann zu $\mathrm{Ins}_{\mathrm{Func}}(f, v) \in \mathbf{P}_\alpha$ entwickelt.
- $f\&<v', p>$ ist ein Prozeß, der entweder $v' \in \alpha$ sagt und zu $p \in \mathbf{P}_\alpha$ wird, oder der $v \in \alpha$ hört und zu $\mathrm{Ins}_{\mathrm{Func}}(f, v) \in \mathbf{P}_\alpha$ wird.
- $p|q$ ist ein Prozeß, der sich wie $p$ und $q$ gleichzeitig verhält, wobei er dafür sorgt, daß:
  - höchstens einer der beiden Prozesse $p$ und $q$ spricht,
  - beide Prozesse $p$ und $q$ hören, was $p|q$ hört,
  - $p|q$ sagt, was einer der beiden Prozesse $p$ und $q$ sagt,
  - wenn einer der beiden Prozesse $p$ und $q$ etwas sagt, der jeweils andere es hört.
- $A$ ist durch $\mathrm{Ins}_{\mathrm{Def}}(A) \in \mathbf{D}_\alpha$ zu ersetzen. Das geschieht jedoch erst, wenn das Verhalten von $A$ gefragt ist, damit rekursive Definitionen nicht ins Unendliche laufen.
- $\Phi p_\beta$ verhält sich wie $p_\beta$, nur daß alle Daten übersetzt werden, d.h. $\Phi p_\beta$ sagt $\mathrm{Ins}_{\mathrm{Trans}}(\mathrm{Up}, w_\beta) \in \alpha$, wenn $p_\beta$ das Element $w_\beta$ aus $\beta_\tau$ sagt. und $p_\beta$ hört $\mathrm{Ins}_{\mathrm{Trans}}(\mathrm{Down}, w_\alpha) \in \beta$, wenn $\Phi p_\beta$ das Element $w_\alpha$ aus $\alpha_\tau$ hört.
- $\delta.p$ wartet genau $\delta$ Zeiteinheiten und verhält sich dann wie $p$.
- $\delta{:}p$ wartet $\delta$ Zeiteinheiten und verhält sich dann wie $p$. Wenn vor Ablauf der $\delta$ Zeiteinheiten etwas gesagt wird, dann verhält sich $\delta{:}p$ wie $p$, wobei $p$ das, was gesagt wurde, hört.
- Alle Prozesse, die $\tau$ hören, bleiben unverändert.

Die Semantik von TCBS wird als operationale Semantik eingeführt, d.h. man erklärt, wie sich ein Prozeß entwickelt. Dazu wird zuerst der Begriff der Transition eingeführt.

**Definition 3.9 ($\alpha$-Transitionen auf einer Menge)**
***Sei** $P$ **eine Menge. Dann ist die Relation***

$$T \subseteq P \times (\alpha_\tau \times \{?, !\} \cup \mathbb{N} \setminus \{0\} \times \{:\}) \times P$$

***eine Menge von** $\alpha$**-Transitionen auf** $P$**. Die verkürzte Schreibweise dafür lautet:***

- $T \ni (p, w, !, p') \Longleftrightarrow p \xrightarrow{w!} p'$***, man sagt:** $p$ **kann** $w$ **sagen und dabei zu** $p'$ **werden.***
- $T \ni (p, w, ?, p') \Longleftrightarrow p \xrightarrow{w?} p'$***, man sagt:** $p$ **kann** $w$ **hören und dabei zu** $p'$ **werden.***
- $T \ni (p, \delta, :, p') \Longleftrightarrow p \xrightarrow{\delta:} p'$***, man sagt:** $p$ **kann** $\delta$ **Zeiteinheiten vergehen lassen und dabei zu** $p'$ **werden.***

***Dabei ist** $p, p' \in P$**,** $w \in \alpha_\tau$ ***und*** $\delta \in \mathbb{N}$**.***

Es gelten in dieser Arbeit folgende Konventionen für die Benennung von in Transitionen auftretetenden Variablen:

- $w \in \alpha_\tau$
- $v \in \alpha$
- $\delta \in \mathbb{N}$
- $u \in \alpha_\tau \cup \mathbb{N}$

Außerdem treten häufig Ausdrücke der Form $u\ddagger \in \alpha \times \{!, ?\} \cup \mathbb{N} \times \{:\}$ auf. Dabei ist $\ddagger$ ein Platzhalter für den Typ der Transition (z.B. $\ddagger$=!) und $u$ ist entweder aus $\alpha$ oder aus $\mathbb{N}$, abhängig vom Transitionstyp.

Eine Menge von Objekten, auf der Transitionen definiert sind, bekommt nun noch einen bestimmten Namen:

**Definition 3.10 ($\alpha$-Prozeßmenge)**
$P$ ***heißt eine** $\alpha$**-Prozeßmenge, wenn auf** $P$ **eine Menge von** $\alpha$**-Transitionen definiert ist.***

Durch folgende Definition wird $\mathbf{P}_\alpha$ zur $\alpha$-Prozeßmenge, indem auf $\mathbf{P}_\alpha$ bestimmte $\alpha$-Transitionen definiert werden. Dazu wird die Notation der logischen Schlußstriche verwendet. Seien $a_1, a_2, ..., a_n, b$ Aussagen. Dann ist

$$\frac{a_1\ a_2\ \ldots\ a_n}{b}$$

gleichwertig zu

$$a_1 \wedge a_2 \wedge \ldots \wedge a_n \Longrightarrow b$$

Wenn Transitionen in der folgenden Tabelle ohne diese Schlußstriche auftreten, dann werden sie damit definiert, wohingegen solche mit Vorbedingungen nur dann vorhanden sind, wenn ihre Voraussetzungen erfüllt sind. Diese rekursive Form der Bildung der semantischen Bedeutung von Prozessen ist nur deshalb möglich, weil ein TCBS-Prozeß $p \in \mathbf{P}_\alpha$ einen eindeutigen Ableitungsbaum hat.

**Definition 3.11 (operationale Semantik von TCBS)**
***In dieser Definition seien*** $p, p', p'', q, q' \in \mathbf{P}_\alpha$***,*** $p_\beta, p'_\beta \in \mathbf{P}_\beta$***,*** $\delta, \eta \in \mathbb{N}$***,*** $w \in \alpha_\tau$***,*** $v, v' \in \alpha$***,*** $w_\beta \in \beta_\tau$***,*** $f \in \mathrm{Func}_\alpha$***,*** $\Phi \in \mathrm{Trans}_{\alpha,\beta}$ ***sowie*** $A \in \mathrm{Def}_\alpha$***. Dann definieren folgende Regeln für alle*** $\alpha$ ***und*** $\beta$ ***aus*** $\Gamma$ ***die möglichen Transitionen auf*** $\mathbf{P}_\alpha$***:***

| **Operator** | **?** | **!** | **:** |
|---|---|---|---|
| **Allgemein** | $p \xrightarrow{\tau?} p$ | | $\dfrac{p \xrightarrow{\delta:} p' \; p' \xrightarrow{\eta:} p''}{p \xrightarrow{(\delta+\eta):} p''}$ |
| **Talk** | $f\&<v', p> \xrightarrow{v?} f(v)$ | $f\&<v', p> \xrightarrow{v'!} p$ | |
| **Hear** | $?f \xrightarrow{v?} f(v)$ | | $?f \xrightarrow{\delta:} ?f$ |
| **Timeout** | $\dfrac{p \xrightarrow{v?} p'}{\delta{:}p \xrightarrow{v?} p'}$ | $\dfrac{p \xrightarrow{w!} p'}{0{:}p \xrightarrow{w!} p'}$ | $(\delta+\eta){:}p \xrightarrow{\delta:} \eta{:}p$<br>$\dfrac{p \xrightarrow{\delta:} p'}{0{:}p \xrightarrow{\delta:} p'}$ |
| **Delay** | $\delta.p \xrightarrow{v?} \delta.p$<br>$\dfrac{p \xrightarrow{v?} p'}{0.p \xrightarrow{v?} p'}$ | $\dfrac{p \xrightarrow{w!} p'}{0.p \xrightarrow{w!} p'}$ | $(\delta+\eta).p \xrightarrow{\delta:} \eta.p$<br>$\dfrac{p \xrightarrow{\delta:} p'}{0.p \xrightarrow{\delta:} p'}$ |
| **Translate** | $\dfrac{p_\beta \xrightarrow{\Phi_\downarrow(v)?} p'_\beta}{\Phi p_\beta \xrightarrow{v?} \Phi p'_\beta}$ | $\dfrac{p_\beta \xrightarrow{w_\beta!} p'_\beta}{\Phi p_\beta \xrightarrow{\Phi_\uparrow(w_\beta)!} \Phi p'_\beta}$ | $\dfrac{p_\beta \xrightarrow{\delta:} p'_\beta}{\Phi p_\beta \xrightarrow{\delta:} \Phi p'_\beta}$ |
| **Compose** | $\dfrac{p \xrightarrow{v?} p' \; q \xrightarrow{v?} q'}{(p\|q) \xrightarrow{v?} (p'\|q')}$ | $\dfrac{p \xrightarrow{w\ddagger_1} p' \; q \xrightarrow{w\ddagger_2} q'}{(p\|q) \xrightarrow[(\ddagger_1,\ddagger_2)\in\{(!,?),(?,!)\}]{w!} (p'\|q')}$ | $\dfrac{p \xrightarrow{\delta:} p' \; q \xrightarrow{\delta:} q'}{(p\|q) \xrightarrow{\delta:} (p'\|q')}$ |
| **Define** | $\dfrac{A() = p \; p \xrightarrow{v?} p'}{A \xrightarrow{v?} p'}$ | $\dfrac{A() = p \; p \xrightarrow{w!} p'}{A \xrightarrow{w!} p'}$ | $\dfrac{A() = p \; p \xrightarrow{\delta:} p'}{A \xrightarrow{\delta:} p'}$ |

Jetzt folgen einige Erläuterungen zu den einzelnen Transitionsdefinitionen.

- **Allgemein :** Alle Prozesse können $\tau$ hören und bleiben dabei unverändert ($p \xrightarrow{\tau?} p$). Die Zeit ist additiv ($\dfrac{p \xrightarrow{\delta:} p' \; p' \xrightarrow{\eta:} p''}{p \xrightarrow{(\delta+\eta):} p''}$).

- **Talk :** $f\&<v', p>$ kann $v$ hören und zu $f(v) \in \mathbf{P}_\alpha$ werden ($f\&<v', p> \xrightarrow{v?} f(v)$) oder $v'$ sagen und zu $p$ werden ($f\&<v', p> \xrightarrow{v'!} p$). $f\&<v', p>$ kann keine Zeit vergehen lassen ($\nexists q \in \mathbf{P}_\alpha : f\&<v', p> \xrightarrow{\delta:} q$)

- **Hear :** $?f$ kann $v$ hören und zu $f(v)$ werden ($?f \xrightarrow{v?} f(v)$) oder eine beliebige Zeit vergehen lassen und dabei unverändert bleiben ($?f \xrightarrow{\delta:} ?f$).

- **Timeout :** $\delta{:}p$ kann $v$ hören und verhält sich dann wie $p$ ($\dfrac{p \xrightarrow{v?} p'}{\delta{:}p \xrightarrow{v?} p'}$). $0{:}p$ verhält sich genauso wie $p$ ($\dfrac{p \xrightarrow{v?} p'}{\delta{:}p \xrightarrow{v?} p'}$, $\dfrac{p \xrightarrow{w!} p'}{0{:}p \xrightarrow{w!} p'}$, $\dfrac{p \xrightarrow{\delta:} p'}{0{:}p \xrightarrow{\delta:} p'}$). $\delta{:}p$ kann Zeit vergehen lassen, auch Stück für Stück. Die vergangene Zeit wird von $\delta$ abgezogen ($(\delta+\eta){:}p \xrightarrow{\delta:} \eta{:}p$)

- **Delay :** $\delta.p$ kann $v$ hören und bleibt unverändert ($\delta.p \xrightarrow{v?} \delta.p$). $0.p$ verhält sich genauso wie $p$ ($\frac{p \xrightarrow{v?} p'}{0.p \xrightarrow{v?} p'}$, $\frac{p \xrightarrow{w!} p'}{0.p \xrightarrow{w!} p'}$, $\frac{p \xrightarrow{\delta:} p'}{0.p \xrightarrow{\delta:} p'}$). $\delta.p$ kann wiederum analog zu $\delta{:}p$ Zeit vergehen lassen ($(\delta + \eta).p \xrightarrow{\delta:} \eta.p$).

- **Translate :** $\Phi p_\beta$ kann $v \in \alpha$ hören und zu $\Phi p'_\beta$ werden, wenn $p_\beta$, der übersetzte Prozeß, $\Phi_\downarrow(v) \in \beta_\tau$ hören kann und zu $p'_\beta$ wird ($\frac{p_\beta \xrightarrow{\Phi_\downarrow(v)?} p'_\beta}{\Phi p_\beta \xrightarrow{v?} \Phi p'_\beta}$). $\Phi p_\beta$ kann $\Phi_\uparrow(w_\beta) \in \alpha_\tau$ sagen, wenn $p_\beta$, der übersetzte Prozeß $w_\beta \in \beta_\tau$ sagen kann und zu $p'_\beta$ wird ($\frac{p_\beta \xrightarrow{w_\beta!} p'_\beta}{\Phi p_\beta \xrightarrow{\Phi_\uparrow(w_\beta)!} \Phi p'_\beta}$). Außerdem kann $\Phi p_\beta$ die Zeit $\delta$ vergehen lassen und zu $\Phi p'_\beta$ werden , wenn $p_\beta$ die Zeit $\delta$ vergehen lassen kann und zu $p'_\beta$ wird ($\frac{p_\beta \xrightarrow{\delta:} p'_\beta}{\Phi p_\beta \xrightarrow{\delta:} \Phi p'_\beta}$). Der Translate-Operator ist der einzige, der eine originäre Transitionsregel für das Sprechen von $\tau$ hat, nämlich genau dann, wenn $\Phi_\uparrow(w_\beta) = \tau$ ist. Wie man außerdem sieht, bleibt bei jeder Transition das $\Phi$ vor dem jeweiligen Prozeß erhalten. Es kann also nicht von einem Schritt zum nächsten verschwinden. Das ist auch vernünftig, denn $p_\beta \notin \mathbf{P}_\alpha$ ($\beta \neq \alpha$). Aus diesem Grunde heißen der Translate- und der Compose-Operator statische Operatoren, die im Gegensatz zu allen anderen Operatoren bei jeder Transition erhalten bleiben.

- **Compose** Der Compose-Operator ist zweifellos der wichtigste aller Operatoren, da er ja erst die Parallelität ins Spiel bringt. Das passiert folgendermaßen: $(p|q)$ kann $v$ hören und zu $(p'|q')$ werden, wenn sowohl $p$ als auch $q$, die Teilprozesse der Komposition, $v$ hören können und zu $p'$ bzw. $q'$ werden ($\frac{p \xrightarrow{v?} p' \; q \xrightarrow{v?} q'}{(p|q) \xrightarrow{v?} (p'|q')}$). Interessanter ist der Fall, in dem $(p|q)$, eine parallele Komposition, $w$ sagen kann und zu $(p'|q')$ wird. Das kann sie genau dann, wenn einer der beiden Teilprozesse $p$ und $q$ auch $w$ sagen kann und der andere $w$ hören kann und sie sich beide zu $p'$ bzw. $q'$ entwickeln ( $\frac{p \xrightarrow{w\ddagger_1} p' \; q \xrightarrow{w\ddagger_2} q'}{(p|q) \xrightarrow{w!} (p'|q')}$; $_{(\ddagger_1,\ddagger_2)\in\{(!,?),(?,!)\}}$). Der Fall, daß beide Prozesse $p$ und $q$ etwas sagen, ist nicht möglich. Dies ist die praktische Implementation dessen, was zu Beginn der Arbeit mit Kollisionsverhütung bezeichnet wurde, daß nämlich in jeder Transition immer nur ein Teilprozeß spricht. Zeit läßt $(p|q)$ dann vergehen, wenn sowohl $p$ als auch $q$ Zeit vergehen lassen. Das heißt, wenn einer der beiden nicht bereit ist, Zeit vergehen zu lassen, ist auch nicht $(p|q)$ ( $\frac{p \xrightarrow{\delta:} p' \; q \xrightarrow{\delta:} q'}{(p|q) \xrightarrow{\delta:} (p'|q')}$). Das ist die semantische Definition für die früher formulierte Vorgabe, Zeit solle nur dann vergehen, wenn keiner der parallel geschalteten Prozesse mehr etwas sagen will. Da $f\&<v', p>$ es nicht zuläßt, daß Zeit vergeht, kann es auch keine parallele Kombination, in der irgendwo ein Talk-Operator vorkommt. Man sieht außerdem auch, daß $|$ ein statischer Operator ist.

- **Define** wenn $A() = p$ ist, verhält sich $A$ wie $p$ ($\frac{A() = p \; p \xrightarrow{v?} p'}{A \xrightarrow{v?} p'}$, $\frac{A() = p \; p \xrightarrow{w!} p'}{A \xrightarrow{w!} p'}$, $\frac{A() = p \; p \xrightarrow{\delta:} p'}{A \xrightarrow{\delta:} p'}$). Das ist nicht überraschend und man mag sich fragen, warum man nicht gleich $p$ anstatt $A$ schreibt. Die Antwort ist, daß in $p$ wieder $A$ enthalten sein kann, was zu

einer unendlich langen Definition von $p$ führen würde, ersetzte man $A$ immer wieder durch $p$. Deshalb dieses Konzept der sogenannten „lazy evaluation“: In einem Ausdruck wird $A$ erst durch seine Definition $A() = p$ ersetzt, wenn es nötig ist, nicht vorher.

Im folgenden werden häufig **abkürzende Schreibweisen** für Transitionen verwendet, diese seien hier angegeben ($u\ddagger \in \alpha_\tau \times \{?,!\} \cup \mathbb{N} \times \{:\}; p \in \mathbf{P}_\alpha$):

- $p \xrightarrow{u\ddagger} \iff \exists p' \in \mathbf{P}_\alpha : p \xrightarrow{u\ddagger} p'$
- $p \not\xrightarrow{u\ddagger} \iff \nexists p' \in \mathbf{P}_\alpha : p \xrightarrow{u\ddagger} p'$
- $p \not\xrightarrow{:} \iff \nexists \delta \in \mathbb{N}, p' \in \mathbf{P}_\alpha : p \xrightarrow{\delta:} p'$
- $p \not\xrightarrow{\ddagger}; \ddagger \in \{!,?\} \iff \nexists w \in \alpha_\tau, p' \in \mathbf{P}_\alpha : p \xrightarrow{w\ddagger} p'$
- $p \xrightarrow{u_1\ddagger_1} \xrightarrow{u_2\ddagger_2} p' \iff \exists p'' \in \mathbf{P}_\alpha : p \xrightarrow{u_1\ddagger_1} p'' \ \wedge \ p'' \xrightarrow{u_2\ddagger_2} p'$
- $p \xrightarrow{[u_1\ddagger_1,\ldots,u_n\ddagger_n]} p' \iff p \xrightarrow{u_1\ddagger_1} p' \ \vee \ \ldots \ \vee \ p \xrightarrow{u_n\ddagger_n} p'$
- $p \xrightarrow{u\ddagger^*} p' \iff \exists p_1, \ldots, p_n \in \mathbf{P}_\alpha : p \xrightarrow{u\ddagger} p_1 \wedge p_1 \xrightarrow{u\ddagger} p_2 \wedge \ldots \wedge p_{n-1} \xrightarrow{u\ddagger} p_n \wedge p_n \xrightarrow{u\ddagger} p'$

Außerdem treten selbsterklärende Kombinationen daraus auf, wie z.B. $p \xrightarrow{[\tau!,1:]^*} \xrightarrow{u\ddagger}$ .

## 3.4 Läufe von Prozessen

Die folgende Definition bestimmt nun den Zusammenhang zwischen Transitionen und einem Lauf eines Prozesses. Dabei ist der Lauf eines Prozesses das Verhalten eines Prozesses, der von einer Umgebung belauscht wird, aber selber nichts von dieser Umgebung hört.

**Definition 3.12 (Läufe eines TCBS-Prozesses)**
***Sei*** $p \in \mathbf{P}_\alpha$***. Dann heißt die Menge*** $\mathrm{Run}(p) \subseteq (\alpha_\tau \times \{!\} \cup \mathbb{N} \times \{:\})^{\mathbb{N}}$ ***die Menge der Läufe des Prozesses*** $p$***. Die Folge*** $\mathrm{Run}^i(p) \in \mathrm{Run}(p)$***;*** $i \in I$ ***heißt ein Lauf des Prozesses*** $p$ ***und die Folge*** $\mathrm{State}^i(p) \in \mathbf{P}_\alpha^{\mathbb{N}}$ ***heißt die zugehörige Folge von Zuständen des Prozesses*** $p$***. Dabei ist*** $I$ ***eine Indexmenge, die nur zur Numerierung dient, und es gilt:***

- $\mathrm{State}_0^i(p) = p$***, d.h. der nullte Zustand eines Prozesses ist der Prozeß selber.***
- $\forall n \in \mathbb{N} : \mathrm{State}_n^i(p) \xrightarrow{\mathrm{Run}_n^i(p)} \mathrm{State}_{n+1}^i(p)$***, d.h. der*** $n$***-te Zustand eines Prozesses geht durch die*** $n$***-te Transition des Laufes in den*** $n+1$***-ten Zustand über.***

Der unbestimmte Artikel (ein Lauf des Prozesses $p$) deutet bereits an, daß es mehrere Läufe geben kann, d.h. daß TCBS nichtdeterministisch arbeitet. Bis jetzt ist aber weder das formal gezeigt, noch ist die obige Definition gerechtfertigt, da noch nicht bekannt ist, ob zu jedem $p$ eine Transition der verlangten Form existiert, die es in einen anderen beliebigen Prozeß überführt. Dieses wird später nachgeholt (siehe Satz 3.4 auf Seite 33).

**Definition 3.13 (abbrechender Lauf)**
***Ein Lauf*** $\mathrm{Run}^i(p)$ ***heißt abbrechend wenn ein*** $n_0 \in \mathbb{N}$ ***existiert, so daß:***

$$\forall n \geq n_0 : \mathrm{Run}^i_n(p) = \delta_n\text{:}\textbf{, wobei } \delta_n \in \mathbb{N}.$$

***Dann kann*** $\mathrm{Run}^i(p)$ ***abkürzend als*** $\left(\mathrm{Run}^i_0(p), ..., Run^i_{n_0-1}(p)\right)$ ***geschrieben werden.***

Die letzte Definition erleichtert den Umgang mit Prozessen, die „absterben“, also von einem gewissen Zustand ab nur noch hören und warten, aber nichts mehr sagen. Das ist z.B. beim Nil-Operator $\mathbf{0}_\alpha$ (siehe Abschnitt 3.6) der Fall.

## 3.5 Gleichheit von Prozessen

In Abschnitt 4.1 auf Seite 35 wird der Begriff der starken Bisimulation eingeführt. Dieser definiert eine Äquivalenzrelation $\sim$ auf $\mathbf{P}_\alpha$. Dadurch werden Prozesse identifiziert, die das gleiche Verhalten zeigen. Im folgenden werden bereits (ohne Beweis, der später erfolgt) folgende Eigenschaften von TCBS bezüglich $\sim$ verwendet:

- $(p|q) \sim (q|p)$, d.h. $|$ ist ein kommutativer Operator.
- $(p|(q|r)) \sim ((q|p)|r)$, d.h. $|$ ist assoziativ. Das rechtfertigt die Schreibweise $p|q|r$ ohne Klammern, die in dieser Arbeit vorwiegend verwendet wird.
- Wenn $p \sim q$, dann $\mathrm{Run}(p) = \mathrm{Run}(q)$
- $\sim$ ist eine Kongruenzrelation.

Im übrigen gelten Prozesse, die stark bisimuliert, also bezüglich $\sim$ äquivalent sind, als gleich. Dieses ist durch die beiden letzten obengenannten Punkte plausibel.

## 3.6 Abgeleitete Operatoren

In diesem Abschnitt werden zwei wichtige abgeleitete Operatoren eingeführt und einige Eigenschaften dieser angegeben. Der erste Operator ist bereits im Abschnitt 3.2 auf Seite 21 verwendet worden. Es ist der Nil-Operator $\mathbf{0}_\alpha$. Die Definition sei hier noch einmal wiederholt:

**Definition 3.14 (Nil-Operator)**
$\mathbf{0}_\alpha \in \mathbf{P}_\alpha$ ***heißt Nil-Operator, wenn:***

$$\mathbf{0}_\alpha :=? f_{\mathbf{0}_\alpha}$$
$$\forall v \in \alpha : f_{\mathbf{0}_\alpha}(v) := \mathbf{0}_\alpha$$

Im folgenden wird häufig der Index weggelassen und nur $\mathbf{0}$ geschrieben, da der Typ $\alpha \in \Gamma$ meist eindeutig aus dem Kontext zu schließen ist. Die Definition von $\mathbf{0}$ bedeutet im Grunde nichts anderes, als daß $\mathbf{0}$ zwar alles hört, aber nicht darauf reagiert, sondern weiterlauscht. Für $\mathbf{0}$ gelten folgende Regeln:

- $\mathbf{0}_\alpha|p \sim p$, d.h. ein parallel zu anderen Prozessen laufendes $\mathbf{0}$ kann man weglassen

- $\mathrm{Run}(\mathbf{0}_\alpha) = \{\epsilon\}$, d.h. $\mathbf{0}_\alpha$ hat nur den leeren Lauf. Das bedeutet, daß der Lauf eines Prozesses, der im Laufe seiner Entwicklung zu $\mathbf{0}_\alpha$ geworden ist, an dieser Stelle abbricht, also der beobachtenden Umgebung nichts mehr sagt.
- $\forall \delta \in \mathbb{N} : \mathbf{0}_\alpha \xrightarrow{\delta:} \mathbf{0}_\alpha$
- $\forall v \in \alpha : \mathbf{0}_\alpha \xrightarrow{v?} \mathbf{0}_\alpha$

Ebensogut könnten die letzten beiden Regeln in der Definition der Semantik und $\mathbf{0}$ als Konstruktor in der Syntax-Definition stehen. Das wäre gleichwertig, würde jedoch in den Beweisen die Fallunterscheidungen verlängern.

Der zweite und letzte abgeleitete Operator ist der Say-Operator $!<v, p>$. Er ist sozusagen der Talk-Operator, der alles, was gesagt wird, ignoriert.

**Definition 3.15 (Say-Operator)**
$!<v, p> \in \mathbf{P}_\alpha$ ***heißt Say-Operator, wenn:***

$$!<v, p> := \mathrm{Say}_{v,p} \,\&<v, p>$$
$$\forall v' \in \alpha : \mathrm{Say}_{v,p}(v') :=!<v, p>$$

***Dabei ist*** $v \in \alpha$ ***und*** $p \in \mathbf{P}_\alpha$

Die Technik ist bei dieser Definition dieselbe wie bei der von $\mathbf{0}$. Die Funktion $\mathrm{Say}_{v,p}$ ignoriert wie $f_{\mathbf{0}_\alpha}$, was gesagt wurde. Dieser „Kunstgriff“ ist nötig, da TCBS, wie später gezeigt wird, die Eigenschaft hat, daß jeder Prozeß bereit ist, etwas zu hören. Auch der Say-Operator könnte als Konstruktor in der Syntax-Definition angegeben werden. Die zugehörige Semantik wäre dann:

- $!<v, p> \xrightarrow{v!} p$
- $\forall v' \in \alpha :!<v, p> \xrightarrow{v'?}!<v, p>$.

## 3.7 Ein Beispiel

Jetzt wird das schon angeführte Beispiel der Bestimmung des höchsten Alters, bzw. der höchsten Zahl näher betrachtet. Die zugehörigen Definitionen werden hier noch einmal angegeben:

$$\mathrm{Cell}_n := f_{\mathrm{Cell}_n} \&<n, \mathbf{0}_N>$$
$$f_{\mathrm{Cell}_n}(m) := \begin{cases} \mathbf{0}_N & m \geq n \\ \mathrm{Cell}_n & \text{sonst} \end{cases}$$
$$p := (\mathrm{Cell}_1 \mid \mathrm{Cell}_{42})$$

Nun muß eine Transition für $p$ gefunden werden: Die einzigen Schlußregeln für den Parallelitätsoperator $|$, den äußersten Operator, setzen die Kenntnis über die möglichen Transitionen der parallel gesetzten Prozesse voraus. Also müssen zuerst die Transitionen für $\mathrm{Cell}_1$ und $\mathrm{Cell}_{42}$ gefunden

werden. Dafür gelten die Transitionsregeln für den Define-Operator, da $\mathrm{Cell}_1, \mathrm{Cell}_{42} \in \mathrm{Def}_N$. Also werden jetzt die Transitionen für $f_{\mathrm{Cell}_1}\&<1, \mathbf{0}_N>$ und $f_{\mathrm{Cell}_{42}}\&<42, \mathbf{0}_N>$ benötigt. Diese sind nach den beiden Schlußregeln für den Talk-Operator wie folgt:

$$f_{\mathrm{Cell}_1}\&<1, \mathbf{0}_N> \xrightarrow{1!} \mathbf{0}_N$$

$$\forall v \in N : f_{\mathrm{Cell}_1}\&<1, \mathbf{0}_N> \xrightarrow{v?} \begin{cases} \mathbf{0}_N & v \geq 1 \\ \mathrm{Cell}_1 & \text{sonst} \end{cases}$$

$$f_{\mathrm{Cell}_{42}}\&<42, \mathbf{0}_N> \xrightarrow{42!} \mathbf{0}_N$$

$$\forall v \in N : f_{\mathrm{Cell}_{42}}\&<42, \mathbf{0}_N> \xrightarrow{v?} \begin{cases} \mathbf{0}_N & v \geq 42 \\ \mathrm{Cell}_{42} & \text{sonst} \end{cases}$$

und also wegen der Schlußregeln für den Define-Operator:

$$\mathrm{Cell}_1 \xrightarrow{1!} \mathbf{0}_N$$

$$\forall v \in N : \mathrm{Cell}_1 \xrightarrow{v?} \begin{cases} \mathbf{0}_N & v \geq 1 \\ \mathrm{Cell}_1 & \text{sonst} \end{cases}$$

$$\mathrm{Cell}_{42} \xrightarrow{42!} \mathbf{0}_N$$

$$\forall v \in N : \mathrm{Cell}_{42} \xrightarrow{v?} \begin{cases} \mathbf{0}_N & v \geq 42 \\ \mathrm{Cell}_{42} & \text{sonst} \end{cases}$$

Im folgenden wird bei Angabe von Transitionsregeln die Forderung $\forall v \in \alpha$ weggelassen. Sie erfolgt trotzdem immer implizit. Das bedeutet, daß anstelle von $\forall v \in \alpha : p \xrightarrow{v?} p'_v$ eigentlich $|\alpha|$ verschiedene Regeln stehen, für jedes $v \in \alpha$ eine. Die Transitionsregeln für den Parallelitätsoperator lassen nun 3 Möglichkeiten für eine Transition zu, und zwar:

$$(\mathrm{Cell}_1 \,|\, \mathrm{Cell}_{42}) \xrightarrow{1!} (\mathbf{0}_N| \,\mathrm{Cell}_{42})$$

$$(\mathrm{Cell}_1 \,|\, \mathrm{Cell}_{42}) \xrightarrow{42!} (\mathbf{0}_N|\mathbf{0}_N)$$

$$(\mathrm{Cell}_1 \,|\, \mathrm{Cell}_{42}) \xrightarrow{v?} \begin{cases} (\mathrm{Cell}_1 \,|\, \mathrm{Cell}_{42}) & v < 1 \\ (\mathbf{0}_N| \,\mathrm{Cell}_{42}) & 1 \leq v < 42 \\ (\mathbf{0}_N|\mathbf{0}_N) & 42 \leq v \end{cases}$$

Für die Bestimmung der ersten Elemente der möglichen Läufe von $(\mathrm{Cell}_1 \,|\, \mathrm{Cell}_{42})$ kommen nur die ersten beiden Transitionen in Betracht, da in der dritten eine Transition der Form $p \xrightarrow{v?} p'$ erscheint. Da $(\mathbf{0}_N|\mathbf{0}_N)$ abbricht, ist ein Lauf des Prozesses schon gefunden und zwar $(42!) \in \mathrm{Run}(p)$. Die möglichen Transitionen für $(\mathbf{0}_N| \,\mathrm{Cell}_{42}) \sim \mathrm{Cell}_{42}$ sind bereits bestimmt:

$$\mathrm{Cell}_{42} \xrightarrow{42!} \mathbf{0}_N$$

$$\forall v \in N : \mathrm{Cell}_{42} \xrightarrow{v?} \begin{cases} \mathbf{0}_N & v \geq 42 \\ \mathrm{Cell}_{42} & \text{sonst} \end{cases}$$

Wieder ist nur der erste Zweig interessant, da nur er die für einen Lauf verlangte Form der Transition aufweist. Übrig bleibt $\mathbf{0}_N$, von dem bereits bekannt ist, daß es den leeren Lauf hat. Der zweite

mögliche Lauf von $(\mathrm{Cell}_1 \mid \mathrm{Cell}_{42})$ ist also $(1!, 42!) \in \mathrm{Run}(p)$. Da es keine anderen Möglichkeiten der Entwicklung mehr gibt, gilt:

$$\mathrm{Run}(\mathrm{Cell}_1 \mid \mathrm{Cell}_{42}) = \{(42!), (1!, 42!)\}.$$

Damit ist für diesen Fall gezeigt, daß der höchste Wert immer als letzter im Lauf steht. Außerdem ist die Ursache des Nichtdeterminismus deutlich geworden: Mehrere Prozesse, die parallel arbeiten, wollen etwas sagen. In diesem Fall wird zur Bestimmung eines Laufs einer der zur Ausgabe fähigen Prozesse willkürlich gewählt und er spricht, während alle anderen zuhören müssen.

## 3.8 Induktive Beweise über TCBS

In dieser Arbeit wird ein Großteil der Beweise immer demselben Schema folgen: Es sind Induktionsbeweise über $\mathrm{Depth}(p)$ ($p \in \mathbf{P}_\alpha$). Dieses Schema soll nun genauer dargestellt und erläutert werden.

Sei $H$ die für alle $p \in \mathbf{P}_\alpha$ zu beweisende Aussage über Transitionen. Alle Induktionsbeweise über $\mathrm{Depth}(p)$ laufen dann in 2 Stufen ab:

(i) $H$ gilt für alle $p \in \mathbf{D}_\alpha \subseteq \mathbf{P}_\alpha$: Dieser Beweis erfolgt mit Hilfe der Induktion über $\mathrm{Depth}(p)$. Als Induktionsanfang sind also alle Prozesse $p \in \mathbf{D}_\alpha$ mit $\mathrm{Depth}(p) = 1$ zu betrachten. Diese sind aber alle von der Form $?f$, da nur im Hear-Zweig der Definition von $P$ ein $\mathbf{d}_\alpha$ ausschließlich durch Terminalsymbole ersetzt wird (siehe Definition 3.7 auf Seite 19). Nachdem $H(?f)$ bewiesen wurde, kommt nun die Induktionsvoraussetzung: Für alle $\alpha \in \Gamma; p \in \mathbf{D}_\alpha$ mit $\mathrm{Depth}\, p \leq n$ gelte $H(p)$. Der Induktionsschritt beweist nun, daß unter dieser Voraussetzung $p \in \mathbf{D}_\alpha$ mit $\mathrm{Depth}(p) = n + 1$ auch $H$ erfüllt. Dazu erfolgt eine Fallunterscheidung über die Konstruktoren der Syntax, die $\mathbf{d}_\alpha$ ersetzen, also Talk, Compose, Translate, Timeout und Delay. Deren Subprozesse $p'$ haben natürlich immer eine kleinere Tiefe ($\mathrm{Depth}(p') \leq n$). Damit ist gezeigt:

$$\forall p \in \mathbf{D}_\alpha : H(p).$$

(ii) $H$ gilt für alle $p \in \mathbf{P}_\alpha$: Auch dieser Beweis erfolgt mit Hilfe der Induktion über $\mathrm{Depth}(p)$. Als Induktionsanfang sind hier alle $p \in \mathbf{P}_\alpha$ mit $\mathrm{Depth}(p) = 1$ zu betrachten. Diese sind alle von der Form $A \in \mathrm{Def}_\alpha$. $A$ hat die gleichen Transitionen wie $\mathrm{Ins}_{\mathrm{Def}}(A) \in \mathbf{D}_\alpha$ und da es in der Aussage $H$ immer nur um Eigenschaften von Transitionen geht, und $H$ für alle $p \in \mathbf{D}_\alpha$ gilt (siehe Schritt (i)), gilt sie auch hierfür. Jetzt kommt wieder die Induktionsvoraussetzung: Für alle $\alpha \in \Gamma; p \in \mathbf{P}_\alpha$ mit $\mathrm{Depth}\, p \leq n$ gelte $H(p)$. Der Induktionsschritt beweist nun, daß unter dieser Voraussetzung $p \in \mathbf{P}_\alpha$ mit $\mathrm{Depth}(p) = n + 1$ auch $H$ erfüllt. Dafür ist auch eine Fallunterscheidung notwendig, aber diesmal über die Konstruktoren Compose, Translate, Timeout und Delay, sowie für den Fall, daß $\mathbf{p}_\alpha$ durch $\mathbf{d}_\alpha$ ersetzt wird. Dafür jedoch gilt $H$ trivialerweise, denn es gilt ja für alle $p \in \mathbf{D}_\alpha$. Bleiben die Fälle Compose, Translate, Timeout und Delay. Diese laufen jedoch analog zu den Beweisen im 1. Schritt, denn dort wird für den Beweis, das z.B. $H(p|q)$ gilt, nur benötigt, daß $H(p)$ und $H(q)$ gilt, nicht jedoch, daß $p, q \in \mathbf{D}_\alpha$

Um also eine Aussage $H$ über Transitionen für $p \in \mathbf{P}_\alpha$ zu beweisen, muß folgendes gezeigt werden:

- **Hear :** $H(?f)$.

- **Talk :** $H(f\&<v,p>)$
- **Compose :** $H(p) \wedge H(q) \Longrightarrow H(p|q)$
- **Translate :** $H(p_\beta) \Longrightarrow H(\Phi p_\beta)$
- **Delay :** $H(p) \Longrightarrow H(\delta.p)$
- **Timeout :** $H(p) \Longrightarrow H(\delta{:}p)$

Als einfaches Beispiel wird folgender Satz bewiesen:

**Satz 3.1**
***Sei*** $p \in \mathbf{P}_\alpha$ ***und*** $\delta \in \mathbb{N}$***. Wenn*** $p \xrightarrow{\delta:}$***, dann gilt für alle*** $\eta \in \mathbb{N}; \eta \leq \delta$***, daß*** $p \xrightarrow{\eta:}$

**Beweis**
Hier folgt jetzt die Fallunterscheidung:

- **Hear :** Es gilt für alle $\eta \in \mathbb{N}$:
$$?f \xrightarrow{\eta:} .$$
- **Talk :** Es gilt
$$f\&<v,p> \not\xrightarrow{:} .$$
- **Compose :** Aus $p|q \xrightarrow{\delta:}$ folgt wegen der Transitionsregel für den Parallelitätsoperator | bzgl. der Transition $\xrightarrow{\delta:}$, daß $p \xrightarrow{\delta:}$ und $q \xrightarrow{\delta:}$. Für $p$ und $q$ gilt aber $H$, also gilt $\forall \eta \leq \delta : p \xrightarrow{\eta:} \wedge\, q \xrightarrow{\eta:}$ und also wegen selbiger Transitionsregel
$$\forall \eta \leq \delta : p|q \xrightarrow{\eta:} .$$
- **Translate :** Aus $\Phi p_\beta \xrightarrow{\delta:}$ folgt $p_\beta \xrightarrow{\delta:}$ und weil $p_\beta$ die Aussage $H$ erfüllt, daß $\forall \eta \leq \delta : p_\beta \xrightarrow{\eta:}$ und also
$$\forall \eta \leq \delta : \Phi p_\beta \xrightarrow{\eta:} .$$
- **Delay :** Aus $\delta'.p \xrightarrow{\delta:}$ folgt, wenn $\delta' \geq \delta$, daß
$$\forall \eta \leq \delta : \delta'.p \xrightarrow{\eta:} (\delta' - \eta).p.$$
Wenn hingegen $\delta' < \delta$, dann muß die allgemeine Regel $\dfrac{\delta'.p \xrightarrow{\delta':} p \; p \xrightarrow{(\delta-\delta'):} p'}{\delta'.p \xrightarrow{\delta:} p'}$ zutreffen, und aus $p \xrightarrow{(\delta-\delta'):}$ folgt, da die Aussage für $p$ ja gilt, daß $\forall \eta \leq \delta : p \xrightarrow{(\eta-\delta'):}$. Daraus folgt aber wieder, daß $\forall \eta \leq \delta : \delta'.p \xrightarrow{\delta':} \xrightarrow{(\eta-\delta'):}$ und also
$$\forall \eta \leq \delta : \delta'.p \xrightarrow{\eta:} .$$

- **Timeout :** Die Regeln für Timeout und Delay sind die Transition $\xrightarrow{\delta:}$ betreffend analog, und somit ist es auch dieser Beweis.

■

Es ist hier deutlich zu sehen, wie solche Beweise vonstatten gehen: man sieht in die dem Operator entsprechende Zeile und die der gewünschten Transition entsprechende Spalte der Tabelle in Definition 3.11 auf Seite 24 und schaut, welche Folgerungen daraus zu ziehen sind. Dabei gilt: Wenn in der entsprechenden Zelle der Tabelle keine Eintragung steht und auch die allgemeinen Regeln in Zeile Eins nicht zutreffen, dann existiert keine solche Transition. Ähnlich einfach läßt sich auch der folgende Satz beweisen:

**Satz 3.2**
***Sei*** $p \in \mathbf{P}_\alpha$ ***ein TCBS-Prozeß und*** $\delta \in \mathbb{N}$***. Wenn*** $p \xrightarrow{\delta:}$***, dann*** $p \not\xrightarrow{!}$***. Wenn*** $p \xrightarrow{w!}$***, dann*** $p \not\xrightarrow{:}$***.***

Dieser Satz zeigt, daß der Zeitbegriff nach der hier vorgegeben Semantik der zu Anfang gestellten Bedingung entspricht, Zeit solle nur vergehen, wenn der Prozeß und damit keiner seiner Teilprozesse etwas zu sagen hat. Warten und Sprechen schließen sich also aus. Dabei ist die zweite Aussage des Satzes natürlich nur die Kontraposition der ersten.

## 3.9 Hörbereitschaft, Determinismus und Aktionsbereitschaft

In diesem Abschnitt wird die Berechtigung für die Wohldefiniertheit der Sprache TCBS und für die Definition des Laufes gegeben. Als erstes wird bewiesen, daß jeder Prozeß jedes $w \in \alpha_\tau$ zu hören bereit ist, und daß er sich in diesem Fall deterministisch entwickelt. Der zweite Satz sichert, daß jeder Prozeß mindestens eine der zwei für den Lauf eines Prozesses nötigen Transitionen $\xrightarrow{w!}$ und $\xrightarrow{\delta:}$ hat.

**Satz 3.3 (Hörbereitschaft und Determinismus)**
***Sei*** $p \in \mathbf{P}_\alpha$ ***und*** $w \in \alpha_\tau$***. Dann existiert genau ein*** $p' \in \mathbf{P}_\alpha$***, so daß*** $p \xrightarrow{w?} p'$

**Beweis**
Dieser Beweis läuft nach dem in Abschnitt 3.8 auf Seite 30 angegebenen Schema: Im folgenden wird aber der Fall $p \xrightarrow{\tau?} p$ stillschweigend weggelassen, da für ihn die Aussage des Satzes offensichtlich immer gilt.

- **Hear :** Für $?f$ gilt: $?f \xrightarrow{v?} f(v)$ für $v \in \alpha$. Da $f$ eine totale Funktion ist, folgt, daß die Behauptung gilt.

- **Talk :** Es gilt: $f\&<v, p'> \xrightarrow{v'?} f(v')$ und es gibt keine andere Regel $f\&<v, p'> \xrightarrow{v'?} p'$. Also ist auch hier die Aussage des Satzes erfüllt.

- **Compose :** Die einzige Regel aus Definition 3.11 auf Seite 24, die Aussagen für $|$ bezüglich $\xrightarrow{v?}$ macht, ist

$$\frac{p \xrightarrow{v?} p' \; q \xrightarrow{v?} q'}{(p|q) \xrightarrow{v?} (p'|q')}.$$

  Da für $p$ und $q$ die Aussage wahr ist, gilt also:$\forall v \in \alpha \; \exists! p', q' : p \xrightarrow{v?} p' \; \wedge \; q \xrightarrow{v?} q'$. Daraus folgt aber sofort, daß $\forall v \in \alpha \; \exists! r : (p|q) \xrightarrow{v?} r$ und zwar $r = (p'|q')$.

- **Translate :** Die einzig in Frage kommende Semantikregel ist

$$\frac{p_\beta \xrightarrow{\Phi_\downarrow(v)?} p'_\beta}{\Phi p_\beta \xrightarrow{v?} \Phi p'_\beta}.$$

  $p_\beta$ erfüllt die Aussage und also $\forall w_\beta \in \beta_\tau\ \exists! p'_\beta : p_\beta \xrightarrow{w_\beta?} p'_\beta$. Da $\Phi_\downarrow(v) \in \beta_\tau$ für alle $v \in \alpha$, gilt: $\exists! p' : \Phi p_\beta \xrightarrow{v?} p'$ und zwar ist $p' = \Phi p'_\beta$.

- **Timeout :** Für diesen Konstruktor kommt nur folgende Semantikregel in Frage:

$$\frac{p \xrightarrow{v?} p'}{\delta{:}p \xrightarrow{v?} p'}.$$

  $p$ erfüllt die Aussage und also $\exists! p' : p \xrightarrow{v?} p'$. Daraus folgt unmittelbar, daß $\exists! p' : \delta{:}p \xrightarrow{v?} p'$.

- **Delay :** Für diesen Konstruktor kommen zwei Semantikregeln in Frage:

$$\delta.p \xrightarrow{v?} \delta.p; \delta \neq 0$$
$$\frac{p \xrightarrow{v?} p'}{0.p \xrightarrow{v?} p'}$$

  Für $\delta = 0$ ist der Fall analog zum Timeout-Konstruktor. Sonst bleibt $p$ für alle $v \in \alpha$ unverändert, also $\delta.p \xrightarrow{v?} p; \delta \neq 0$

■

Jetzt ist auch noch einmal deutlich die Ursache für die Einführung zweier Nichtterminalsymbolsorten $(\mathbf{d}_\alpha, \mathbf{p}_\alpha)$ in Definition 3.7 auf Seite 19 zu sehen. Würden für die Definition eines $A \in \mathrm{Def}_\alpha$ auch $p \in \mathbf{P}_\alpha$ zugelassen sein[3], wäre eine Definition wie $A = A$ zulässig. Dafür würde aber der Satz nicht gelten, denn man kann keine Aussage über die Reaktion von $A$ machen. Die einzigen Operatoren, die dafür sorgen, daß eine Aufnahme möglich und das Ergebnis deterministisch ist, sind Hear und Talk, sowie Compose, Translate, Delay und Timeout, die vier letztgenannten aber nur unter der Voraussetzung, daß die in ihnen aufgehenden Prozesse schon aus $\mathbf{P}_\alpha$ sind.[4] Deswegen heißt auch $\mathbf{D}_\alpha$ die Menge aller überwachten Prozesse, da für $p \in \mathbf{D}_\alpha$ unabhängig von Definitionseinsetzungen die direkten Entwicklungsmöglichkeiten bestimmt werden können.

Im nächsten Satz wird gezeigt, daß die Definition 3.12 auf Seite 26 wohlgeformt ist, d.h. das mindestens ein solcher Lauf existiert.

**Satz 3.4 (Aktionsbereitschaft)**
***Sei*** $p \in \mathbf{P}_\alpha$***. Dann existiert mindestens ein*** $p' \in \mathbf{P}_\alpha$***, so daß*** $p \xrightarrow{w!} p'$ ***(für ein*** $w \in \alpha_\tau$***) oder*** $p \xrightarrow{\delta:} p'$ ***(für ein*** $\delta \in \mathbb{N}$***).***

**Beweis**
Auch dieser Beweis läuft nach dem in Abschnitt 3.8 auf Seite 30 angegebenen Schema:

[3] Das ist die einzige Verwendung von $\mathbf{d}_\alpha$ außerhalb der Definition 3.7 auf Seite 19, ohne sie könnte $\mathbf{d}_\alpha$ dort einfach durch $\mathbf{p}_\alpha$ ersetzt werden. Ohne die dann redundanten Regeln wäre die Definition viel einfacher.

[4] Für den Delay-Operator steht die Sache etwas anders, er ist eigentlich auch ein Operator, der für $\delta \neq 0$ den Satz erfüllt. Er wird jedoch trotzdem in Analogie zum Timeout-Operator behandelt.

- **Hear :** Es gilt:
$$?f \xrightarrow{\delta:} ?f.$$
- **Talk :** Es gilt:
$$f\&<v, p'> \xrightarrow{v!} p'$$
- **Compose :** Das ist der einzig interessante Fall in diesem Beweis, da hier auch der vorhin bewiesene Satz vonnöten ist. Im ersten Fall wird angenommen, daß sowohl $p$ als auch $q$ Zeit vergehen lassen können, und zwar $\delta_p$ bzw. $\delta_q$. Daraus folgt, daß beide auch $\delta = \min(\delta_p, \delta_q)$ Zeiteinheiten vergehen lassen können (siehe Satz 3.1 auf Seite 31). Dann kann wegen:
$$\frac{p \xrightarrow{\delta:} p' \; q \xrightarrow{\delta:} q'}{p|q \xrightarrow{\delta:} p'|q'}$$
der Prozeß $p|q$ auch Zeit vergehen lassen, erfüllt also die Behauptung. Nun sei o.B.d.A. $p$ nicht in der Lage, Zeit vergehen zu lassen. Darum und weil $p$ die Aussage des Satzes erfüllt, gilt: $\exists p' \in \mathbf{P}_\alpha, w \in \alpha_\tau : p \xrightarrow{w!} p'$. Nun gilt aber nach Satz 3.3 auf Seite 32: $\exists! q' \in \mathbf{P}_\alpha : q \xrightarrow{w?} q'$. Daraus und aus der Transitionsregel
$$\frac{p \xrightarrow{w!} p' \; q \xrightarrow{w?} q'}{(p|q) \xrightarrow{w!} (p'|q')}$$
folgt nun direkt die Behauptung des Satzes.
- **Translate :** Da für $p_\beta$ die Aussage stimmt, gilt entweder: $p_\beta \xrightarrow{w_\beta!} p'_\beta$ oder $p_\beta \xrightarrow{\delta:} p'_\beta$. Im ersten Fall gilt wegen
$$\frac{p_\beta \xrightarrow{w_\beta!} p'_\beta}{\Phi p_\beta \xrightarrow{\Phi_\uparrow(w_\beta)!} \Phi p'_\beta}$$
und im zweiten Fall wegen
$$\frac{p_\beta \xrightarrow{\delta:} p'_\beta}{\Phi p_\beta \xrightarrow{\delta:} \Phi p'_\beta}$$
die Behauptung des Satzes.
- **Timeout :** Wegen
$$(\delta + \eta){:}p \xrightarrow{\delta:} \eta{:}p$$
gilt die Behauptung.
- **Delay :** Wegen
$$(\delta + \eta).p \xrightarrow{\delta:} \eta.p$$
gilt auch hier die Behauptung.

■

Mit Hilfe dieses Satzes ist nun die Existenz eines Laufes eines Prozesses gewährleistet. Das ist im Grunde erst die Bestätigung für die Wohldefiniertheit der operationalen Semantik, denn ein Prozeß, der keine Aktion, also keine Entwicklung, keinen Lauf haben kann, wäre natürlich nicht besonders sinnvoll.

# 4 Bisimulationen

Wie bereits angedeutet, ist es bei solchen formalen Kalkülen sinnvoll, gewisse Prozesse auf Grund ihres Verhaltens in bestimmte Klassen einzusortieren. Dabei gibt es verschiedene Möglichkeiten. Sind einerseits die Klassen zu klein, sind nur sehr wenige Aussagen über die gleiche Zugehörigkeit verschiedener Prozesse möglich, sind andererseits die Klassen zu groß, so sind die gewonnenen Aussagen meist nicht relevant. Im allgemeinen sind Klassen nur dann von Interesse, wenn es Äquivalenzklassen sind, sie also durch Äquivalenzrelationen gebildet werden. Das ist auch hier der Fall. Es werden in diesem Kapitel die zwei am häufigsten im Zusammenhang mit parallelen Kalkülen angewendeten Äquivalenzrelationen, die starke und die schwache Bisimulation, eingeführt und einige Eigenschaften bewiesen.

Zunächst werden zwei Begriffe definiert, um mit den Bisimulationen umgehen zu können.

**Definition 4.1 (Äquivalenzrelation)**
***Eine Relation*** $\mathcal{R} \subseteq P \times P$ ***heißt Äquivalenzrelation, wenn gilt:***

- **Reflexivität :** $\forall p \in P : (p,p) \in \mathcal{R}$
- **Symmetrie :** $\forall p,q \in P : (p,q) \in \mathcal{R} \Longrightarrow (q,p) \in \mathcal{R}$
- **Transitivität :** $\forall p,q,r \in P : (p,q),(q,r) \in \mathcal{R} \Longrightarrow (p,r) \in \mathcal{R}$

Eine Äquivalenzrelation auf $\mathbf{P}_\alpha$ erlaubt es, den Begriff der Äquivalenz zweier Prozesse genauer zu definieren. Es kann verschiedene solche Begriffe geben, wie ja auch bei den natürlichen Zahlen: $1 = 1$ aber auch $5 = 1 \bmod(4)$.

**Definition 4.2 (Kongruenzrelation)**
***Eine Relation*** $\mathcal{R} \subseteq P \times P$ ***heißt Kongruenzrelation bezüglich der*** $s$***-stelligen Funktion*** $f : P^s \to P$***, wenn folgendes gilt:***

$$(\forall 1 \leq k \leq s : p_k \mathcal{R} p'_k) \Longrightarrow f(p_1,...,p_s)\mathcal{R} f(p'_1,...,p'_s)$$

Eine Kongruenzrelation $\mathcal{R}$ bezüglich aller Konstruktoren von TCBS auf $\mathbf{P}_\alpha$, die auch eine Äquivalenzrelation ist, definiert eine starke Form der Gleichheit, denn man kann, wenn $p\mathcal{R}p'$ gilt, $p$ in jedem Prozeß $q$ durch $p'$ ersetzen und für das Ergebnis $q'$ gilt: $q\mathcal{R}q'$. Dies ist natürlich eine wünschenswerte Eigenschaft für eine Äquivalenz.

## 4.1 Die starke Bisimulation

**Definition 4.3 (starke Bisimulation)**
***Seien*** $P_1$ ***und*** $P_2$ ***zwei*** $\alpha$***-Prozeßmengen. Eine Relation*** $\mathcal{R} \subseteq P_1 \times P_2$ ***ist eine starke Bisimulation, wenn für alle*** $u\ddagger \in \alpha_\tau \times \{?,!\} \cup \mathbb{N} \times \{:\}$ ***gilt:***

(i) ***Wenn*** $p\mathcal{R}q$ ***und*** $p \xrightarrow{u\ddagger} p'$***, dann*** $\exists q'$***, so daß*** $q \xrightarrow{u\ddagger} q'$ ***und*** $p'\mathcal{R}q'$***,***

(ii) ***Wenn*** $p\mathcal{R}q$ ***und*** $q \xrightarrow{u\ddagger} q'$***, dann*** $\exists p'$***, so daß*** $p \xrightarrow{u\ddagger} p'$ ***und*** $p'\mathcal{R}q'$***.***

***Die größte starke Bisimulation wird mit*** $\sim_{P_1,P_2}$ ***bezeichnet.***

Im folgenden wird häufig der Index von $\sim$ weggelassen, da aus dem Kontext meist zu schließen ist, welche Mengen betrachtet werden. Im übrigen ist bis auf wenige Ausnahmen $P_1 = P_2 = \mathbf{P}_\alpha$. Es ist offensichtlich, daß es eine größte starke Bisimulation gibt, da die Vereinigung von zwei starken Bisimulationen $\mathcal{R}_1$ und $\mathcal{R}_2$ wieder eine starke Bisimulation ist, und andererseits jede starke Bisimulation $\mathcal{R} \subseteq P_1 \times P_2$ sein muß. Andererseits ist es für den Nachweis, daß zwei Prozesse $p_1 \in P_1$ und $p_2 \in P_2$ die Eigenschaft $p_1 \sim p_2$ erfüllen, ausreichend, die Existenz einer starken Bisimulation $\mathcal{R}$ mit $(p_1, p_2) \in \mathcal{R}$ zu zeigen. Da die zweite geforderte Eigenschaft die symmetrische Entsprechung der ersten ist, folgt aus $p_1 \mathcal{R} p_2 \Longrightarrow p_2 \mathcal{R} p_1$ und dem Beweis der Eigenschaft (i) für $\mathcal{R}$ schon, daß $\mathcal{R}$eine starke Bisimulation ist.

Jetzt wird auch der Grund für die Namensgebung für die starke Bisimulation klar: man führt beide Prozesse nebeneinander aus (man bisimuliert sie) und schaut, wie sie sich verhalten; wenn es für jede Transition des einen Prozesses eine Entsprechung beim anderen Prozeß gibt und umgekehrt, dann gelten die Prozesse als äquivalent, man sagt auch, als stark bisimuliert.

**Satz 4.1**
***Die starke Bisimulation $\sim_{\mathbf{P}_\alpha, \mathbf{P}_\alpha}$ ist eine Äquivalenzrelation und eine Kongruenzrelation bezüglich aller Konstruktoren von $\mathbf{P}_\alpha$ (TCBS).***

**Beweis**
**$\sim_{\mathbf{P}_\alpha, \mathbf{P}_\alpha}$ ist Äquivalenzrelation :**

- **Reflexivität :** Zum Beweis wird die Relation

$$\mathcal{R} := \left\{ (p, p) \;\middle|\; p \in \mathbf{P}_\alpha \right\}$$

  definiert und bewiesen, daß sie eine starke Bisimulation ist. Dann gilt $\mathcal{R} \subseteq \sim$ und damit auch $\forall p \in P : (p, p) \in \sim$. $\mathcal{R}$ ist eine starke Bisimulation, da $p$ sich wie $p$ verhalten und sich dabei natürlich auch gleich (zu $p'$) entwickelt, und da $(p', p') \in \mathcal{R}$ gilt.

- **Symmetrie :** Diese ist offensichtlich erfüllt, da der zweite Zweig der Definition der starken Bisimulation die symmetrische Entsprechung zum ersten ist.

- **Transitivität :** Nun sei

$$\mathcal{R} := \left\{ (p, r) \;\middle|\; (p, q), (q, r) \in \sim_{\mathbf{P}_\alpha, \mathbf{P}_\alpha} \right\}.$$

  Um zu zeigen, daß $\mathcal{R}$ eine starke Bisimulation ist, werden alle Paare $(p, r) \in \mathcal{R}$ betrachtet: $p$ kann sich wie ein $q$ verhalten und das Paar der resultierenden Prozesse ist in $\sim$. Analoges gilt für dasselbe $q$ und $r$. Also kann sich $p$ wie $r$ verhalten, und da $(p', q'), (q', r') \in \sim$, gilt für die resultierenden Prozesse $p'$ und $r'$: $(p', r') \in \mathcal{R}$. Damit ist $\mathcal{R}$ eine starke Bisimulation.

**$\sim_{\mathbf{P}_\alpha, \mathbf{P}_\alpha}$ ist Kongruenzrelation :** Nun muß für alle Konstruktoren, die auf Elementen aus $\mathbf{P}_\alpha$ operieren (Talk, Translate, Compose, Timeout, Delay) bestimmt werden, ob ihre Anwendung die Relation $\sim_{\mathbf{P}_\alpha, \mathbf{P}_\alpha}$ erhält. Das läuft immer daraus hinaus, eine geeignete Relation $\mathcal{R} \in \mathbf{P}_\alpha \times \mathbf{P}_\alpha$ zu finden, so daß z.B. für den Translate-Operator alle Paare $(\Phi p, \Phi q)$ mit $p \sim q$ Element von $\mathcal{R}$ sind. Diese Beweise sind bis auf den Fall des Compose-Operators ziemlich offensichtlich aus den Transitionsregeln abzuleiten. Das liegt daran, daß sie alle einstellig sind und damit die Transitionen gleich sein müssen, denn die Transitionen, die die jeweiligen Operatoren ausführen, sind

ja, da die Operatoren identisch sind, gleich und die in ihren enthaltenen Prozesse sind stark bisimuliert. Etwas interessanter ist der Fall des Compose-Operators, da er binär ist, und damit das Zusammenspiel von zwei verschiedenen Prozessen untersucht wird. Dazu ist zu beweisen, daß

$$\mathcal{R} := \left\{ (p_1|p_2, q_1|q_2) \;\middle|\; p_1 \sim q_1 \;\wedge\; p_2 \sim q_2 \right\}$$

eine starke Bisimulation ist. Dazu reicht es zu zeigen, daß:

$$p_1|p_2 \mathcal{R} q_1|q_2 \;\wedge\; p_1|p_2 \xrightarrow{u\ddagger} p_1'|p_2' \Longrightarrow \exists q_1', q_2' : q_1|q_2 \xrightarrow{u\ddagger} q_1'|q_2' \;\wedge\; p_1'|p_2' \mathcal{R} q_1'|q_2'.$$

Dafür ist hinreichend, daß:

$$p_1 \sim q_1 \;\wedge\; p_2 \sim q_2 \;\wedge\; p_1|p_2 \xrightarrow{u\ddagger} p_1'|p_2' \Longrightarrow \exists q_1', q_2' : q_1|q_2 \xrightarrow{u\ddagger} q_1'|q_2' \;\wedge\; p_1' \sim q_1' \;\wedge\; p_2' \sim q_2'.$$

Jetzt erfolgt eine Fallunterscheidung über die drei Möglichkeiten der Belegung von $u\ddagger$:

- $u\ddagger = \delta :$ . Es reicht zu zeigen, daß:

  $$p_1 \sim q_1 \;\wedge\; p_2 \sim q_2 \;\wedge\; p_1 \xrightarrow{\delta:} p_1' \;\wedge\; p_2 \xrightarrow{\delta:} p_2' \Longrightarrow \exists q_1', q_2' : q_1 \xrightarrow{\delta:} q_1' \;\wedge\; q_2 \xrightarrow{\delta:} q_2' \;\wedge\; p_1' \sim q_1' \;\wedge\; p_2' \sim q_2'.$$

  Das aber folgt aus der Voraussetzung, daß $p_1 \sim q_1$ und $p_2 \sim q_2$.

- $u\ddagger = w?$ . vollkommen analog zum Fall $u\ddagger = \delta :$.

- $u\ddagger = w!$ . Hier gibt es zwei analoge Fälle. O.B.d.A sei $(\ddagger_1, \ddagger_2) = (!, ?)$ (siehe Transitionsregel für den Compose-Operator in Definition 3.11 auf Seite 24). Dann reicht es, zu zeigen, daß:

  $$p_1 \sim q_1 \;\wedge\; p_2 \sim q_2 \;\wedge\; p_1 \xrightarrow{w!} p_1' \;\wedge\; p_2 \xrightarrow{w?} p_2' \Longrightarrow \exists q_1', q_2' : q_1 \xrightarrow{w!} q_1' \;\wedge\; q_2 \xrightarrow{w?} q_2' \;\wedge\; p_1' \sim q_1' \;\wedge\; p_2' \sim q_2'.$$

  Das folgt wiederum aus der Voraussetzung, daß $p_1 \sim q_1$ und $p_2 \sim q_2$.

■

Jetzt folgen einige Eigenschaften der starken Bisimulation, die teilweise bereits angegeben wurden, aber hier nun bewiesen werden:

**Satz 4.2**

(i) $(p|q) \sim (q|p)$

(ii) $(p|(q|r)) \sim ((q|p)|r)$

(iii) ***Wenn*** $p \sim q$***, dann*** $\mathrm{Run}(p) = \mathrm{Run}(q)$

(iv) $\mathbf{0}_\alpha|p \sim p$

(v) $0{:}p \sim p$

(vi) $0.p \sim p$

(vii) $\delta{:}?f \sim ?f$

**Beweis**
Für die Punkte (i), (ii), (iv), (v), (vi) und (vii) reicht die Angabe der entsprechenden Bisimulationen, da der Beweis, daß es starke Bisimulationen sind, dann ein einfaches Einsetzen der Regeln ist. Die Bisimulation sind aber auch sehr einfach und lauten:

$$\begin{aligned}
\mathcal{R}_{(i)} &:= \{(p|q, q|p) \mid p, q \in \mathbf{P}_\alpha\} \\
\mathcal{R}_{(ii)} &:= \{(p|(q|r), (p|q)|r) \mid p, q, r \in \mathbf{P}_\alpha\} \\
\mathcal{R}_{(iv)} &:= \{(\mathbf{0}_\alpha|p, p) \mid p \in \mathbf{P}_\alpha\} \\
\mathcal{R}_{(v)} &:= \{(0{:}p, p) \mid p \in \mathbf{P}_\alpha\} \\
\mathcal{R}_{(vi)} &:= \{(0.p, p) \mid p \in \mathbf{P}_\alpha\} \\
\mathcal{R}_{(vii)} &:= \{(\delta{:}?f, ?f) \mid f \in \mathrm{Func}_\alpha\}
\end{aligned}$$

Da aus $p \sim q$ automatisch $q \sim p$ folgt, reicht es für Punkt (iii) aus, zu zeigen, daß gilt:

$$p \sim q \Longrightarrow \mathrm{Run}(p) \subseteq \mathrm{Run}(q)$$

und damit reicht es auch, zu zeigen, daß gilt:

$$p \sim q \ \wedge \ \mathrm{Run}^i(p) \in \mathrm{Run}(p) \Longrightarrow \mathrm{Run}^i(p) \in \mathrm{Run}(q)$$

Wenn $\mathrm{Run}^i$ aber ein Lauf von $p$ ist, dann existiert eine Folge von Zuständen von $p$, nennen wir sie $\mathrm{State}^i(p)$, für die gilt:

- $\mathrm{State}^i_0(p) = p$,
- $\forall n \in \mathbb{N} : \mathrm{State}^i_n(p) \xrightarrow{\mathrm{Run}^i_n(p)} \mathrm{State}^i_{n+1}(p)$.

Es gilt nun $p \sim q$, also $\mathrm{State}^i_0(p) \sim \mathrm{State}^j_0(q)$. Wenn nun $\mathrm{State}^i_n(p) \sim \mathrm{State}^j_n(q)$ gilt, folgt daraus, daß es für die Transition $\mathrm{State}^i_n(p) \xrightarrow{\mathrm{Run}^i_n(p)} \mathrm{State}^i_{n+1}(p)$ die Entsprechung $\mathrm{State}^j_n(q) \xrightarrow{\mathrm{Run}^i_n(p)} q_{n+1}$ gibt. Nun sei $\mathrm{Run}^j_n(q)$ durch $\mathrm{Run}^i_n(p)$ und $\mathrm{State}^j_{n+1}(q)$ durch $q_{n+1}$ definiert. Dann gilt $\mathrm{State}^i_{n+1}(p) \sim \mathrm{State}^j_{n+1}(q)$. Daraus folgt, daß $\mathrm{Run}^j(q) = \mathrm{Run}^i(p)$ in $\mathrm{Run}(q)$ enthalten ist. ■

## 4.2 Die schwache Bisimulation

**Definition 4.4 (schwache Bisimulation)**
***Seien** $P_1$ **und** $P_2$ **zwei** $\alpha$**-Prozeßmengen. Eine Relation** $\mathcal{R} \subseteq P_1 \times P_2$ **ist eine schwache Bisimulation, wenn für alle** $u\ddagger \in \alpha \times \{?, !\} \cup \mathbb{N} \times \{:\}$ **gilt:***

(i) ***Wenn** $p\mathcal{R}q$ **und** $p \xrightarrow{u\ddagger} p'$**, dann** $\exists q'$**, so daß** $q \xrightarrow{(\tau!)^*} \xrightarrow{u\ddagger} q'$ **und** $p'\mathcal{R}q'$**,**

(ii) ***Wenn** $p\mathcal{R}q$ **und** $p \xrightarrow{\tau!} p'$**, dann** $\exists q'$**, so daß** $q \xrightarrow{(\tau!)^*} q'$ **und** $p'\mathcal{R}q'$**,**

(iii) ***Wenn** $p\mathcal{R}q$ **und** $q \xrightarrow{u\ddagger} q'$**, dann** $\exists p'$**, so daß** $p \xrightarrow{(\tau!)^*} \xrightarrow{u\ddagger} p'$ **und** $p'\mathcal{R}q'$**,**

(iv) ***Wenn*** $p\mathcal{R}q$ ***und*** $q \xrightarrow{\tau!} q'$***, dann*** $\exists p'$***, so daß*** $p \xrightarrow{(\tau!)^*} p'$ ***und*** $p'\mathcal{R}q'$***.***

***Die größte schwache Bisimulation wird mit*** $\approx_{P_1,P_2}$ ***bezeichnet, der Index kann weggelassen werden, wenn er aus dem Kontext eindeutig hervorgeht.***

Analog zur starken Bisimulation ist die Wohldefiniertheit von $\approx_{P_1,P_2}$ gesichert. Um nachzuweisen, daß $p_1 \approx p_2$ reicht es wieder aus, zu zeigen, daß $\mathcal{R}$ mit $(p_1, p_2) \in \mathcal{R}$ eine schwache Bisimulation ist, außerdem folgt aus $p_1\mathcal{R}p_2 \implies p_2\mathcal{R}p_1$ und dem Beweis der Eigenschaften (i) und (ii) für $\mathcal{R}$ auch schon, daß $\mathcal{R}$ eine schwache Bisimulation ist, da die Eigenschaften (iii) und (iv) die symmetrische Entsprechung der Eigenschaften (i) bzw. (ii) sind.

Bei der schwachen Bisimulation wird im Gegensatz zur starken Bisimulation von den beiden Prozessen nicht erwartet, daß sie sich identisch verhalten, sondern nur, daß sie bis auf interne Aktionen (diese finden statt, wenn ein Übersetzer ein bestimmtes Element aus $\alpha$ nach außen hin zu $\tau$ übersetzt, welches von allen Prozessen ignoriert wird) dasselbe tun. Dabei ist der erste Zweig der Definition in (i) bzw. (iii) klar: wenn $p$ sich zu $p'$ entwickeln kann, wobei es $v$ hört bzw. sagt, muß das auch $q$ mit $q\mathcal{R}p$ können, nur das ihm vorher noch erlaubt ist, beliebig oft $\tau$ zu sagen. $q$ entwickelt sich dabei zu $q'$ und es muß gelten, daß $p'\mathcal{R}q'$. Interessanter ist da schon der zweite Zweig in (ii) bzw. (iv): wenn $p$ das Wort $\tau$ sagt und zu $p'$ wird, muß auch $q$ eine beliebige Anzahl von $\tau$ sagen können und zu $q'$ mit $p'\mathcal{R}q'$ werden. Auf den ersten Blick scheint das analog zum ersten Fall zu sein. Aber im zweiten Fall ist es auch möglich, daß $q$ Null mal $\tau$ sagen kann, d.h. daß $q$ garnichts sagt und $q$ bleibt, daß also $q$ identisch zu $q'$ ist. Würde hingegen der zweite Fall zum ersten dazugeschlagen, dann müßte $q$ mindestens einmal $\tau$ sagen. Verlangte man dies, so würde der internen Aktion $\tau$ aber eine zu starke Bedeutung zugemessen. Es ist aber gerade die Absicht, daß die internen Aktionen unbeachtet bleiben.

**Satz 4.3**
***Die schwache Bisimulation*** $\approx_{\mathbf{P}_\alpha,\mathbf{P}_\alpha}$ ***ist eine Äquivalenzrelation und eine Kongruenzrelation bezüglich aller Konstruktoren von*** $\mathbf{P}_\alpha$ ***(TCBS).***

**Beweis**
Daß die schwache Bisimulation eine Äquivalenzrelation ist, wird analog zum Beweis für die starke Bisimulation bewiesen. Auch zum Beweis, daß $\approx_{\mathbf{P}_\alpha,\mathbf{P}_\alpha}$ eine Kongruenzrelation ist, sind die Betrachtungen ähnlich wie bei der starken Bisimulation. So sind die Beweise für die Operatoren Talk, Timeout und Delay so einfach, daß sie weggelassen werden. Hier ist jetzt aber der Beweis für den Translate-Operator interessant, da er ja das in der schwachen Bisimulation ausgezeichnete $\tau$ umwandeln könnte. Außerdem ist wieder der Beweis für den Compose-Operator zu führen. Dieser ist hier etwas komplexer, aber auch interessanter als beim Beweis für die starke Bisimulation. Als erstes folgt nun der Beweis, daß der Translate-Operator die schwache Bisimulation erhält. Es wird gezeigt, daß für ein festes $\Phi$ die Relation

$$\mathcal{R} := \left\{(\Phi p_\beta, \Phi q_\beta) \;\middle|\; p_\beta \approx q_\beta\right\}$$

eine schwache Bisimulation ist. Dazu reicht es, folgendes zu zeigen ($u\ddagger \in \alpha \times \{?, !\} \cup \mathbb{N} \times \{:\}$):

(i) Wenn $\Phi p_\beta \mathcal{R} \Phi q_\beta$ und $\Phi p_\beta \xrightarrow{u\ddagger} \Phi p'_\beta$, dann $\exists q'_\beta$, so daß $\Phi q_\beta \xrightarrow{(\tau!)^*} \xrightarrow{u\ddagger} \Phi q'_\beta$ und $\Phi p'_\beta \mathcal{R} \Phi q'_\beta$,

(ii) Wenn $\Phi p_\beta \mathcal{R} \Phi q_\beta$ und $\Phi p_\beta \xrightarrow{\tau!} \Phi p'_\beta$, dann $\exists q'_\beta$, so daß $\Phi q_\beta \xrightarrow{(\tau!)^*} \Phi q'_\beta$ und $\Phi p'_\beta \mathcal{R} \Phi q'_\beta$,

Aus den Transitionsregeln für den Translate-Operator und aus $\Phi_\uparrow(\tau) = \tau$ folgt, daß es reicht, zu zeigen, daß gilt:

(i) Sei $v_\beta \in \Phi_\uparrow^{-1}(v)$: Wenn $p_\beta \approx q_\beta$ und $p_\beta \xrightarrow{v_\beta!} p'_\beta$, dann $\exists q'_\beta$, so daß $q_\beta \xrightarrow{(\tau!)^*} \xrightarrow{v_\beta!} q'_\beta$ und $p'_\beta \approx q'_\beta$,

(ii) Sei $\Phi_\downarrow(v) \neq \tau$: Wenn $p_\beta \approx q_\beta$ und $p_\beta \xrightarrow{\Phi_\downarrow(v)?} p'_\beta$, dann $\exists q'_\beta$, so daß $q_\beta \xrightarrow{(\tau!)^*} \xrightarrow{\Phi_\downarrow(v)?} q'_\beta$ und $p'_\beta \approx q'_\beta$,

(iii) Sei $\Phi_\downarrow(v) = \tau$: Wenn $p_\beta \approx q_\beta$ und $p_\beta \xrightarrow{\tau?} p'_\beta$, dann $\exists q'_\beta$, so daß $q_\beta \xrightarrow{\tau?} q'_\beta$ und $p'_\beta \approx q'_\beta$,

(iv) Wenn $p_\beta \approx q_\beta$ und $p_\beta \xrightarrow{\delta:} p'_\beta$, dann $\exists q'_\beta$, so daß $q_\beta \xrightarrow{(\tau!)^*} \xrightarrow{\delta:} q'_\beta$ und $p'_\beta \approx q'_\beta$,

(v) Sei $w_\beta \in \Phi_\uparrow^{-1}(\tau)$: Wenn $p_\beta \approx q_\beta$ und $p_\beta \xrightarrow{w_\beta!} p'_\beta$, dann dann $\exists q'_\beta$, so daß $q_\beta \xrightarrow{(\tau!)^*} \xrightarrow{w_\beta!} q'_\beta$ und $p'_\beta \approx q'_\beta$,

Die Punkte (i), (ii) und (iv) folgen sofort aus der Voraussetzung, daß $p_\beta \approx q_\beta$. Punkt (iii) gilt, weil $p_\beta = p'_\beta$ und $q_\beta = q'_\beta$. Für Punkt (i) ist zu beachten, daß $\tau \notin \Phi_\uparrow^{-1}(v)$. Für Punkt (v) gibt es zwei Fälle, erstens kann $w_\beta = \tau$ sein, dann aber folgt die Aussage sofort aus $p_\beta \approx q_\beta$ (Punkt (ii) der Definition). Zweitens kann $w_\beta \neq \tau$ sein, aber auch dann folgt die Aussage sofort aus der Voraussetzung (Punkt (i) der Definition 4.4 auf Seite 38).

Jetzt folgt der Beweis, daß der Compose-Operator die schwache Bisimulation erhält. Dazu ist zu beweisen, daß

$$\mathcal{R} := \left\{ (p_1|p_2, q_1|q_2) \;\middle|\; p_1 \approx q_1 \;\wedge\; p_2 \approx q_2 \right\}$$

eine schwache Bisimulation ist. Dafür reicht es zu zeigen, daß:

(i) Wenn $p_1|p_2 \mathcal{R} q_1|q_2$ und $p_1|p_2 \xrightarrow{u\ddagger} p'_1|p'_2$, dann $\exists q'_1, q'_2$, so daß $q_1|q_2 \xrightarrow{(\tau!)^*} \xrightarrow{u\ddagger} q'_1|q'_2$ und $p'_1|p'_2 \mathcal{R} q'_1|q'_2$,

(ii) Wenn $p_1|p_2 \mathcal{R} q_1|q_2$ und $p_1|p_2 \xrightarrow{\tau!} p'_1|p'_2$, dann $\exists q'_1, q'_2$, so daß $q_1|q_2 \xrightarrow{(\tau!)^*} q'_1|q'_2$ und $p'_1|p'_2 \mathcal{R} q'_1|q'_2$.

Dafür ist hinreichend, daß gilt:

(i) Wenn $p_1 \approx q_1$ und $p_2 \approx q_2$ und $p_1|p_2 \xrightarrow{u\ddagger} p'_1|p'_2$, dann $\exists q'_1, q'_2$, so daß $q_1|q_2 \xrightarrow{(\tau!)^*} \xrightarrow{u\ddagger} q'_1|q'_2$ sowie $p'_1 \approx q'_1$ und $p'_2 \approx q'_2$,

(ii) Wenn $p_1 \approx q_1$ und $p_2 \approx q_2$ und $p_1|p_2 \xrightarrow{\tau!} p'_1|p'_2$, dann $\exists q'_1, q'_2$, so daß $q_1|q_2 \xrightarrow{(\tau!)^*} q'_1|q'_2$ sowie $p'_1 \approx q'_1$ und $p'_2 \approx q'_2$.

Aus der Voraussetzung für Punkt (ii) folgt (o.B.d.A sagt $p_1$ und hört $p_2$ das Wort $\tau$):

$$p_1 \approx q_1 \;\wedge\; p_2 \approx q_2 \;\wedge\; p_1 \xrightarrow{\tau!} p'_1 \;\wedge\; p_2 \xrightarrow{\tau?} p_2.$$

Daraus folgt wegen der Definition von $\approx$ und weil jeder Prozeß beliebig viele $\tau$ hören kann, ohne sich zu verändern, daß:

$$\exists q'_1 : q_1 \xrightarrow{(\tau!)^*} q'_1 \;\wedge\; q_2 \xrightarrow{(\tau?)^*} q_2 \;\wedge\; p'_1 \approx q'_1 \;\wedge\; p_2 \approx q_2.$$

Daraus folgt direkt die Behauptung von Punkt (ii). Jetzt erfolgt zum Beweis von Punkt (i) eine Fallunterscheidung über die drei Möglichkeiten der Belegung von $u\ddagger$:

- $u\ddagger = \delta :$ . Aus der Voraussetzung folgt wegen der Definition von $\approx$ und weil jeder Prozeß beliebig viele $\tau$ hören kann, daß:

$$\exists q_1', q_2' : q_1 \xrightarrow{(\tau?)^*} \xrightarrow{(\tau!)^*} \xrightarrow{\delta:} q_1' \ \wedge \ q_2 \xrightarrow{(\tau!)^*} \xrightarrow{(\tau?)^*} \xrightarrow{\delta:} q_2' \ \wedge \ p_1' \approx q_1' \ \wedge \ p_2' \approx q_2'.$$

  Daraus folgt unmittelbar die Behauptung.

- $u\ddagger = w?$ . vollkommen analog zum Fall $u\ddagger = \delta :$.

- $u\ddagger = w!$ . Hier gibt es zwei analoge Fälle. O.B.d.A sei $(\ddagger_1, \ddagger_2) = (!, ?)$. Dann folgt, daß:

$$\exists q_1', q_2' : q_1 \xrightarrow{(\tau?)^*} \xrightarrow{(\tau!)^*} \xrightarrow{w!} q_1' \ \wedge \ q_2 \xrightarrow{(\tau!)^*} \xrightarrow{(\tau?)^*} \xrightarrow{w?} q_2' \ \wedge \ p_1' \approx q_1' \ \wedge \ p_2' \approx q_2'.$$

  Daraus folgt unmittelbar die Behauptung.

■

# 5 Äquivalenzen und Korrektheit

Ein bedeutender Aspekt der Betrachtung formaler paralleler Kalküle ist der Nachweis der Korrektheit von Algorithmen. Ein Programm gilt im intuitiven Sinne dann als korrekt, wenn seine Implementation das erfüllt, was seine Spezifikation fordert. Die Implementation eines Algorithmus ist in TCBS wohldefiniert. Von einem Programm in TCBS ist bekannt, wie es sich verhält. Anders sieht es bei der Spezifikation aus. Diese ist oftmals nur verbal gegeben.

Nun wird eine spezielle Algorithmenklasse betrachtet, für die die in dieser Arbeit vorgestellten Korrektheitsbeweistechniken besonders geeignet sind, und zwar die der Protokolle.

Protokolle dienen dazu, Daten über ein potentiell unsicheres Medium, zum Beispiel ein Ethernet, zu übertragen. Nun mag es paradox klingen, das Ethernet als unsicheres Medium zu bezeichnen. Schließlich funktioniert ja bei Benutzung eines solchen Mediums (die meisten LANs sind derzeit auf Grundlage eines Ethernets implementiert) jede Übertragung. Das aber liegt nur an dem verwendeten Netzwerkprotokoll (zunehmend TCP/IP), welches die auftretenden Fehler abfängt.

Die verbale Spezifikation eines einfachen Punkt-zu-Punkt-Übertragungprotokolls, das Daten vom Punkt A zum Punkt B überträgt[5], könnte etwa so lauten:

> Jede von der Seite A ausgehende Information erreicht mit Sicherheit die Seite B.

Der nun verfügbare theoretische Apparat über TCBS legt es nahe, die verbale Spezifikation in diesen Kalkül zu übertragen und zu prüfen, ob Spezifikation ($\in \mathbf{P}_\alpha$) und Implementation ($\in \mathbf{P}_\alpha$) bezüglich einer noch zu bestimmenden Bisimulation äquivalent sind.

Auf den ersten Blick sieht es so aus, als wenn das Übersetzen der Spezifikation nach TCBS genauso kompliziert ist wie die Implementierung, daß also der Vorgang des Übersetzens von Problemstellung zur Spezifikation genauso anfällig für Fehler (um deren Ausschluß es bei der Korrektheitsprüfung geht) ist wie die Programmierung, also die Übertragung des Problems zur Implementation. Das ist für viele Algorithmenklassen auch richtig, aber gerade Protokolle sind für den hier vorgestellten Ansatz sehr gut geeignet, denn obwohl sie selbst sehr kompliziert sein können, ist die obige verbale Spezifikation sehr einfach und damit auch die im folgenden vorgestellte formale Spezifikation in TCBS.

[5] Es gibt auch andere Netzprotokolle, z.B. für die Verständigung von $n$ Rechnern

## 5.1 Vorbetrachtungen zur Definition der Korrektheit

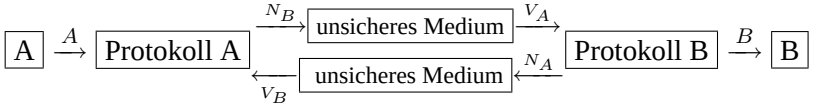
Protokolle sind natürlich nicht von ihrer Außenwelt abgeschnitten, sie sind im Gegenteil ohne mit ihnen kommunizierende Programme völlig sinnlos. Hier werden der Einfachheit halber nur Zweipunkt-Einwegprotokolle betrachtet, also Protokolle, die ausschließlich Informationen von Punkt A nach Punkt B übertragen:


Ein Protokollalgorithmus besteht also aus zwei Seiten, für jeden der beteiligten Partner ist eine Seite zuständig.

TCBS hat aber ein sicheres Medium, auf dem zudem noch alle hören können, was gesagt wird. Um hier ein passendes Modell zu finden, werden die bisher etwas zu kurz gekommenen Übersetzer (die Translate-Operatoren) verwendet. Es werden alle Daten ausgefiltert, die der jeweilige Prozeß aus physikalischen Gründen nicht direkt hören kann, weil er zum Beispiel auf der anderen Seite des unsicheren Mediums ist, oder aber weil z.B. $A$ ja nicht mit $B$ direkt kommuniziert, sondern mit dem Protokoll A.

Seien nun den beteiligten Partnern A und B die TCBS-Prozesse $a$ bzw. $b$ zugeordnet. Dann wird um $a \in \mathbf{P}_\alpha$ der Übersetzer $\Phi_a$ gelegt, der an alle Daten, die $a$ sagt, das Symbol $A$ anhängt und $a$ nichts von außen hören läßt und um $b \in \mathbf{P}_\alpha$ wird der Übersetzer $\Phi_b$ gelegt, der alle Daten, an die das Symbol $B$ angehängt ist, zu lokalen Daten macht, und nichts vom Innenleben von $b$ nach außen dringen läßt außer $\tau$. Nun werden für die anderen direkten Verbindungen (zwischen Protokoll A und dem Medium, sowie zwischen Protokoll B und dem Medium) noch mehr solche Marken benötigt. Außerdem ist damit zu rechnen, daß ein Protokoll zusätzlich zu den Daten des Typs $\alpha$ auch noch andere interne Informationen überträgt, um die einwandfreie Übertragung zu gewährleisten. Dieser vom Protokoll benötigte Datentyp, der sicherlich $\alpha$ in der einen oder anderen Form enthält, heiße $\beta$.

Auf dem Medium wird also die Sprache $\gamma := \{A, B\} \times \alpha \cup \{V_A, V_B, N_A, N_B\} \times \beta$ gesprochen. In der nächsten Darstellung werden die Richtungen der Datenübertragungen dargestellt. Dabei liefert ein Prozeß nur dann mit $X$ markierte Daten, wenn es einen mit $X$ markierter Pfeil gibt, der die entsprechende Box verlaßt. Gleiches gilt für die Aufnahme. Das wird teilweise mit Übersetzern erreicht, oft jedoch über die Definition der entsprechenden Prozesse, die ignorieren, was nicht für sie bestimmt ist.




Dabei soll $N_B$ „nach B“ und $V_A$ „von A“ bedeuten. Nun muß natürlich noch das unsichere Medium in TCBS programmiert werden, und es muß, wie oben zu sehen, in 2 Richtungen Daten übertragen, aus $N_B$ macht es $V_A$ und aus $N_A$ macht es $V_B$, oder eben auch nicht; es überträgt nichts; die Daten gehen auf dem Medium verloren.

## 5.2 Das unsichere Medium

Das unsichere Medium, welches mit $N$ gekennzeichnete Daten mit $V$ kennzeichnet und eventuell überträgt, wird durch folgenden Prozeß implementiert:

$$
\begin{aligned}
\mathrm{Medium}_{N,V} &:=? f_{\mathrm{Medium}_{N,V}} \\
f_{\mathrm{Medium}_{N,V}}(v) &:= \begin{cases} \ddot{\mathrm{U}}\mathrm{bertrage}_{V,d} \,|\, \mathrm{Medium}_{N,V} & \text{falls } v = (N,d) \\ \mathrm{Medium}_{N,V} & \text{sonst} \end{cases} \\
\ddot{\mathrm{U}}\mathrm{bertrage}_{V,d} &:= \Phi_{\ddot{\mathrm{U}}\mathrm{bertrage}}(f_{\mathbf{0}}\&{<}(V,d),\mathbf{0}{>}|f_{\mathbf{0}}\&{<}Ignore,\mathbf{0}{>}) \\
\Phi_{\ddot{\mathrm{U}}\mathrm{bertrage}\uparrow}(v) &:= \begin{cases} v & \text{falls } v = (V,d) \\ \tau & \text{sonst} \end{cases} \\
\Phi_{\ddot{\mathrm{U}}\mathrm{bertrage}\downarrow}(v) &:= \tau
\end{aligned}
$$

Dabei ist die innerhalb des Übersetzers gesprochene Sprache gegenüber der außerhalb gesprochenen Sprache um das Symbol $Ignore$ erweitert, daß vom Übersetzer $\Phi_{\ddot{\mathrm{U}}\mathrm{bertrage}}$ nach außen zu $\tau$ übersetzt wird, da es nicht dem Schema $v = (V,d)$ entspricht. $\Phi_{\ddot{\mathrm{U}}\mathrm{bertrage}}$ übersetzt nach innen hin alles zu $\tau$. $f_{\mathbf{0}}$ wird unabhängig vom Gesagten zu $\mathbf{0}$. Das Ergebnis ist, daß $\ddot{\mathrm{U}}\mathrm{bertrage}_{V,d}$ entweder $\tau$ oder $(V,d)$ sagt und dann zu $\mathbf{0}$ wird, d.h. die Daten werden übertragen oder sie verschwinden. Dieses Medium ist nicht nur unsicher in dem Sinne, daß es nicht gewährleistet, daß überhaupt Daten ankommen. Wenn Daten ankommen, so ist nicht einmal gewährleistet, daß die Reihenfolge gewahrt bleibt. Das liegt daran, daß das Medium, sobald es Daten gehört hat, wieder aufnahmebereit ist (erster Zweig der Definition von $f_{\mathrm{Medium}_{N,V}}$), und die zweite Instanz des Mediums vor der zuerst gestarteten mit der Übertragung der Daten zum Zuge kommen könnte, da ja in TCBS der als nächstes sprechende Prozeß willkürlich gewählt werden kann. Diese Medium ist aber sicher dahingehend, daß Daten nicht verstümmelt werden und das keine falschen Daten vom Medium „erzeugt“ werden. Falls ein Mediums beides nicht sichern kann, muß eine Art von Prüfsumme zur Nachricht hinzugefügt werden, damit man fehlerhafte Nachrichten von fehlerfreien unterscheiden kann. Nun ist aber, mit welcher Prüfsumme auch immer, nicht mit absoluter Sicherheit beweisbar, daß nur fehlerfreie Daten ankommen. Mit einer gewissen (wenn auch sehr kleinen) Wahrscheinlichkeit sind Daten auf solche Art fehlerhaft, daß ihre Prüfsumme wieder stimmt. Deshalb kann man auf dieses Phänomen nicht mit Hilfe dieses Kalküls eingehen. Dafür wäre eine stochastische Betrachtung des spezielles Prüfsummenverfahrens unter Beachtung der möglichen Fehlerquellen und ihrer Eigenschaften erforderlich.

## 5.3 Das perfekte Medium

Nun können wir auch endlich die Spezifikation für ein perfektes Medium angeben. Diese wird gegeben durch:

$$
\begin{aligned}
\mathrm{Perfekt} &:=? f_{\mathrm{Perfekt}} \\
f_{\mathrm{Perfekt}}(v) &:= \begin{cases} !{<}(B,d), \mathrm{Perfekt}{>} & \text{falls } v = (A,d) \\ \mathrm{Perfekt} & \text{sonst} \end{cases}
\end{aligned}
$$

Dieses Medium ist nur perfekt in Hinsicht auf die Korrektheit der übertragenen Daten und die Bewahrung der Ordnung. Es bietet keine Flow Control, d.h. es fängt keine überschnellen Datenströme ab (es kann erst wieder etwas aufnehmen, wenn es selbst seine Daten abgeliefert hat)

und sichert auch nicht, daß abgelieferte Daten auch wirklich schon vom Zielprozeß verarbeitetet werden können. Da eine Flow Control aber durchaus über Mittel der sequentiellen Programmierung gesichert werden kann, wird in dieser Arbeit im weiteren davon abgesehen. Dieses Medium überträgt ohne Mithilfe von weiteren Protokollen, kann also direkt mit $A$ markierte Daten in mit $B$ markierte wandeln. Außerdem überträgt es nur in eine Richtung, da wir uns ja nur für Einwegprotokolle interessieren. Das unsichere Medium jedoch muß in zwei Richtungen übertragen, denn der sendende Protokollprozeß ist auf Rückmeldungen vom empfangenden Protokollprozeß angewiesen.

## 5.4 Korrektheit von Protokollen

Jetzt wird ein erster Versuch gestartet, die Korrektheit von Protokollen im obigen Sinne zu beschreiben. Dazu sei $p_A$ das Protokoll A und $p_B$ das Protokoll B. Dann scheint es plausibel, die Prozesse $\mathrm{Perfekt}$ und $\Phi_{\mathrm{Prot}}(p_A|\,\mathrm{Medium}_{N_B,V_A}\,|\,\mathrm{Medium}_{N_A,V_B}\,|p_B)$ auf Äquivalenz bezüglich der schwachen Bisimulation zu testen. Das hätte den Vorteil, daß man das perfekte Medium $\mathrm{Perfekt}$ in jedem Zusammenhang durch $\Phi_{\mathrm{Prot}}(p_A|\,\mathrm{Medium}_{N_B,V_A}\,|\,\mathrm{Medium}_{N_A,V_B}\,|p_B)$ ersetzen kann, ohne das Ergebnis mehr als von der schwachen Bisimulation erlaubt zu ändern. Es wäre also durchaus angemessen, in diesem Zusammenhang von einem korrekten Protokoll zu sprechen. Dabei sorgt der Übersetzer $\Phi_{\mathrm{Prot}}$ dafür, daß alle internen Aktionen des Protokolls nach außen zu $\tau$ übersetzt werden und so nur das nach außen dringt, was für den Prozeß $B$ bestimmt ist. Außerdem kommt nur das von außen nach innen, was vom Prozeß $A$ kommt und für den Prozeß $B$ bestimmt ist. $\Phi_{\mathrm{Prot}}$ wird also folgendermaßen definiert:

$$\Phi_{\mathrm{Prot}\uparrow}(w) := \begin{cases} w & \text{falls } w = (B,d); d \in \alpha \\ \tau & \text{sonst} \end{cases}$$

$$\Phi_{\mathrm{Prot}\downarrow}(w) := \begin{cases} w & \text{falls } w = (A,d); d \in \alpha \\ \tau & \text{sonst} \end{cases}$$

Das heißt, alles, was außen gesagt wird und mit A markiert ist, wird unverändert reingelassen, alles andere wird zu $\tau$ übersetzt und damit innen ignoriert. Und alles was innen gesagt wird und mit B markiert ist, wird unverändert herausgelassen, alles andere wird nach außen hin zu $\tau$ übersetzt und damit dort ignoriert.

Das Problem ist nun, daß $\mathrm{Perfekt}$, während es die Nachrichten übermittelt, keine Zeit vergehen läßt, was bei den beiden bisher eingeführten Bisimulationen dazu führte, daß auch das zu testende Protokoll das nicht dürfte. Dieses ist natürlich (auch wenn das unsichere Medium instantan überträgt) ein unbefriedigender Zustand, denn viele Protokolle funktionieren erst, weil sie bei nicht erfolgter Übertragung nach einer bestimmten Zeit die Daten nochmals übertragen, die Übertragung also verzögern.

Es ist nicht nur sinnvoll sondern unabdingbar, davon abzusehen, daß ein Protokoll auf einem unsicheren Medium mehr Zeit braucht als ein perfektes Medium, denn die Bandbreite des perfekten Mediums steht voll für Daten zur Verfügung, wohingegen auf dem unsicheren Medium erstens Bandbreite durch Fehler verloren geht, andererseits aber auch durch zusätzlich übertragene protokollinterne Daten.

Das alles führt zu der Idee, Möglichkeiten zu suchen, um von der Zeit, die ein Prozeß zur Erledigung seiner Aufgabe braucht, abzusehen. Dieses wird im nächsten Kapitel getan, bevor am Ende

der Arbeit eine genaue Definition für die Korrektheit eines Punkt-zu-Punkt-Einwegprotokolls angegeben wird.

# 6 Ein erweiterter Kalkül

## 6.1 Einführung

Wie man im letzten Kapitel gesehen hat, wäre es wünschenswert, über eine Bisimulation zu verfügen, die von der Zeit so wie die schwache Bisimulation von $\tau$ abstrahieren würde. Ich werde zur Plausibilisierung der im nächsten Kapitel vorgestellten Bisimulation nun den bisher verwendeten Kalkül TCBS erweitern. Im Prinzip geht es darum, eine neue Art Übersetzer einzuführen, der alle Zeit, die in seinem Inneren vergeht, nicht außen vergehen läßt. Das bedeutete aber in letzter Konsequenz, daß außerhalb des Übersetzers gar keine Zeit mehr vergehen könnte, was wiederum die Verwendung des Kalküls TCBS ad absurdum führen würde.

Es gilt also, die Zeit in TCBS in verschiedene Klassen zu zerlegen, die je nach Typ vom Übersetzer herausgelassen werden oder aber zu $\tau$ übersetzt werden, womit die Zeit die innerhalb des Übersetzers vergeht, außen als Hintergrundrauschen ignoriert wird. Eine solche Einteilung fällt aber bei genauerer Betrachtung nicht schwer:

- Es gibt Zeit, die, wenn ein Prozeß sie vergehen läßt, den Prozeß **unverändert** läßt. Diese Zeit werde wie bisher mit der Transition $\xrightarrow{\delta:}$ bezeichnet. Ein Beispiel dafür ist $?f \xrightarrow{\delta:} ?f$
- Aber es gibt auch eine Zeit, die, wenn ein Prozeß sie vergehen läßt, den Prozeß ändert. Dieser Zeit wird nun die Transition $\xrightarrow{\delta::}$ zugeordnet. Ein Beispiel dafür ist $1.p \xrightarrow{1::} p$.

Es erscheint logisch, daß man die Zeit des ersten Typs, die der übersetzte Prozeß vergehen läßt, unverändert nach außen dringen läßt, da ja sonst der Prozeß inklusive Übersetzer dauernd $\tau$ sagen würde, ohne sich zu verändern. Dies wäre ein in diesem Zusammenhang unerwünschtes Resultat, da $\tau$ ja gerade die internen Aktionen repräsentiert, die zur Erreichung eines bestimmten Zieles durchgeführt werden, aber die Außenwelt nicht interessieren.

## 6.2 Die Syntax von TCBS´

Nun wird der erweiterte Kalkül $\mathrm{TCBS}'$ eingeführt. Die Syntax entspricht im großen und ganzen der von TCBS. In diesem Kalkül werden jedoch noch zwei weitere Konstruktoren zur Syntax hinzugenommen werden, nämlich $L$ als der oben erwähnte Übersetzer, sowie $N$, ein Übersetzer, der die Aufgabe erfüllt, die zwei verschiedenen Zeittypen von $\mathrm{TCBS}'$ ($\xrightarrow{\delta:}$,$\xrightarrow{\delta::}$) auf den originalen von $\mathrm{TCBS}$ ($\xrightarrow{\delta:}$) zu übersetzen.

Um $\mathrm{TCBS}'$ formal zu definieren, müßte vieles von der Definition von $\mathrm{TCBS}$ wiederholt werden. Da es nur um eine kleine Ergänzung geht, seien hier nur die beiden Regeln angegeben, die zur Definition der Syntax von $\mathrm{TCBS}$ hinzugenommen werden müßten, um die Syntax von $\mathrm{TCBS}'$ zu erhalten:

| **Operator** | $\mathbf{d}_\alpha$ | $\mathbf{p}_\alpha$ |
|---|---|---|
| **Localize** | $\mathbf{d}_\alpha \underset{P}{\longrightarrow} L\,\mathbf{d}_\alpha$ | $\mathbf{p}_\alpha \underset{P}{\longrightarrow} L\,\mathbf{p}_\alpha$ |
| **Normalize** | $\mathbf{d}_\alpha \underset{P}{\longrightarrow} N\,\mathbf{d}_\alpha$ | $\mathbf{p}_\alpha \underset{P}{\longrightarrow} N\,\mathbf{p}_\alpha$ |

Die den Mengen $\mathbf{P}_\alpha$, $\mathbf{D}_\alpha$, $\mathrm{Func}_\alpha$, $\mathrm{Trans}_{\alpha,\beta}$ und $\mathrm{Def}_\alpha$ in $\mathrm{TCBS}'$ entsprechenden Mengen heißen $\mathbf{P}'_\alpha$, $\mathbf{D}'_\alpha$, $\mathrm{Func}'_\alpha$, $\mathrm{Trans}'_{\alpha,\beta}$ und $\mathrm{Def}'_\alpha$. Jetzt sei nur noch die Funktion $\mathrm{TCBS}'()$ erwähnt, die auf kanonische Weise jedem TCBS-Prozeß $p$ seine $\mathrm{TCBS}'$-Entsprechung $\mathrm{TCBS}'(p)$ zuordnet. Das ist möglich, da die Syntax von $\mathrm{TCBS}'$ alle Konstruktoren der Syntax von TCBS enthält.

## 6.3 Die Semantik von TCBS´

Da jetzt in dem Kalkül $\mathrm{TCBS}'$ neue Transitionen der Form $\xrightarrow{\delta::}; \delta \in \mathbb{N}$ vorkommen, müssen diese noch zu den ursprünglichen 3 Formen hinzugenommen werden:

**Definition 6.1 (erweiterte $\alpha$-Transitionen auf einer Menge)**
***Sei $P$ eine Menge. Dann ist die Relation***

$$T \subseteq P \times (\alpha_\tau \times \{?,!\} \cup \mathbb{N} \setminus \{0\} \times \{:,::\}) \times P$$

***eine Menge von erweiterten $\alpha$-Transitionen auf $P$. Die verkürzte Schreibweise dafür lautet:***

- $T \ni (p, w, !, p') \Longleftrightarrow p \xrightarrow{w!} p'$***, man sagt: $p$ kann $w$ sagen und dabei zu $p'$ werden.***
- $T \ni (p, w, ?, p') \Longleftrightarrow p \xrightarrow{w?} p'$***, man sagt: $p$ kann $w$ hören und dabei zu $p'$ werden.***
- $T \ni (p, \delta, :, p') \Longleftrightarrow p \xrightarrow{\delta:} p'$***, man sagt: $p$ kann $\delta$ unverändernde Zeiteinheiten vergehen lassen und dabei zu $p'$ werden.***
- $T \ni (p, \delta, ::, p') \Longleftrightarrow p \xrightarrow{\delta::} p'$***, man sagt: $p$ kann $\delta$ verändernde Zeiteinheiten vergehen lassen und dabei zu $p'$ werden.***

***Dabei ist $p, p' \in P$, $w \in \alpha_\tau$ und $\delta \in \mathbb{N}$.***

**Definition 6.2 (erweiterte $\alpha$-Prozeßmenge)**
***$P$ heißt dann eine erweiterte $\alpha$-Prozeßmenge, wenn auf $P$ eine Menge von erweiterten $\alpha$-Transitionen definiert ist.***

Als nächstes folgt nun, und das wieder formal, die Definition der Semantik von $\mathrm{TCBS}'$. Dabei sind grau unterlegte Bereiche Änderungen zur Semantikdefinition von TCBS. Die beiden durchgestrichenen Regeln fallen im Gegensatz zum Original weg.

**Definition 6.3 (operationale Semantik von $\mathrm{TCBS}'$)**
***In dieser Definition seien $p, p', p'', q, q' \in \mathbf{P}'_\alpha$, $p_\beta, p'_\beta \in \mathbf{P}'_\beta$, $\delta, \eta \in \mathbb{N}$, $w \in \alpha_\tau$, $v, v' \in \alpha$, $w_\beta \in \beta_\tau$, $f \in \mathrm{Func}'_\alpha$, $\Phi \in \mathrm{Trans}'_{\alpha,\beta}$ sowie $A \in \mathrm{Def}'_\alpha$. Dann definieren folgende Regeln für alle $\alpha, \beta \in \Gamma$ die möglichen erweiterten Transitionen auf $\mathbf{P}'_\alpha$:***

| Operator | ? | ! | : | :: |
|---|---|---|---|---|
| **Allgemein** | $p \xrightarrow{\tau?} p$ | | $\dfrac{p \xrightarrow{\delta:} p' \; p' \xrightarrow{\eta:} p''}{p \xrightarrow{(\delta+\eta):} p''}$ | $\dfrac{p \xrightarrow{\delta::} p' \; p' \xrightarrow{\eta::} p''}{p \xrightarrow{\delta+\eta)::} p''}$ |
| **Talk** | $f\&<v',p> \xrightarrow{v?} f(v)$ | $f\&<v',p> \xrightarrow{v'!} p$ | | |
| **Hear** | $?f \xrightarrow{v?} f(v)$ | | $?f \xrightarrow{\delta:} ?f$ | |
| **Timeout** | | | $\cancel{(\delta+\eta):p \xrightarrow{\delta:} \eta:p}$ | $(\delta+\eta):p \xrightarrow{\delta::} \eta:p$ |
| | $\dfrac{p \xrightarrow{v?} p'}{\delta:p \xrightarrow{v?} p'}$ | $\dfrac{p \xrightarrow{w!} p'}{0:p \xrightarrow{w!} p'}$ | $\dfrac{p \xrightarrow{\delta:} p'}{0:p \xrightarrow{\delta:} p'}$ | $\dfrac{p \xrightarrow{\delta::} p'}{0:p \xrightarrow{\delta::} p'}$ |
| **Delay** | $\delta.p \xrightarrow{v?} \delta.p$ | | $\cancel{(\delta+\eta).p \xrightarrow{\delta:} \eta.p}$ | $(\delta+\eta).p \xrightarrow{\delta::} \eta.p$ |
| | $\dfrac{p \xrightarrow{v?} p'}{0.p \xrightarrow{v?} p'}$ | $\dfrac{p \xrightarrow{w!} p'}{0.p \xrightarrow{w!} p'}$ | $\dfrac{p \xrightarrow{\delta:} p'}{0.p \xrightarrow{\delta:} p'}$ | $\dfrac{p \xrightarrow{\delta::} p'}{0.p \xrightarrow{\delta::} p'}$ |
| **Translate** | $\dfrac{p_\beta \xrightarrow{\Phi_\downarrow(v)?} p'_\beta}{\Phi p_\beta \xrightarrow{v?} \Phi p'_\beta}$ | $\dfrac{p_\beta \xrightarrow{w_\beta!} p'_\beta}{\Phi p_\beta \xrightarrow{\Phi_\uparrow(w_\beta)!} \Phi p'_\beta}$ | $\dfrac{p_\beta \xrightarrow{\delta:} p'_\beta}{\Phi p_\beta \xrightarrow{\delta:} \Phi p'_\beta}$ | $\dfrac{p_\beta \xrightarrow{\delta::} p'_\beta}{\Phi p_\beta \xrightarrow{\delta::} \Phi p'_\beta}$ |
| **Compose** | $\dfrac{p \xrightarrow{v?} p' \; q \xrightarrow{v?} q'}{(p\|q) \xrightarrow{v?} (p'\|q')}$ | $\dfrac{p \xrightarrow{w\ddagger_1} p' \; q \xrightarrow{w\ddagger_2} q'}{(p\|q) \xrightarrow{w!} (p'\|q')}$ $(\ddagger_1,\ddagger_2)\in\{(!,?),(?,!)\}$ | $\dfrac{p \xrightarrow{\delta:} p' \; q \xrightarrow{\delta:} q'}{(p\|q) \xrightarrow{\delta:} (p'\|q')}$ | $\dfrac{p \xrightarrow{\delta\ddagger_1} p' \; q \xrightarrow{\delta\ddagger_2} q'}{p\|q \xrightarrow{\delta::} p'\|q'}$ $(\ddagger_1,\ddagger_2)\in\{(::,:),(:,::),(::,::)\}$ |
| **Define** | $\dfrac{A() = p \; p \xrightarrow{v?} p'}{A \xrightarrow{v?} p'}$ | $\dfrac{A() = p \; p \xrightarrow{w!} p'}{A \xrightarrow{w!} p'}$ | $\dfrac{A() = p \; p \xrightarrow{\delta:} p'}{A \xrightarrow{\delta:} p'}$ | $\dfrac{A() = p \; p \xrightarrow{\delta::} p'}{A \xrightarrow{\delta::} p'}$ |
| **Normalize** | $\dfrac{p \xrightarrow{w?} p'}{Np \xrightarrow{w?} Np'}$ | $\dfrac{p \xrightarrow{w!} p'}{Np \xrightarrow{w!} Np'}$ | $\dfrac{p \xrightarrow{\delta\ddagger} p'}{Np \xrightarrow{\delta:} Np'}$ $\ddagger\in\{:,::\}$ | |
| **Localize** | $\dfrac{p \xrightarrow{w?} p'}{Lp \xrightarrow{w?} Lp'}$ | $\dfrac{p \xrightarrow{w!} p'}{Lp \xrightarrow{w!} Lp'}$ $\dfrac{p \xrightarrow{1::} p'}{Lp \xrightarrow{\tau!} Lp'}$ | $\dfrac{p \xrightarrow{\delta:} p'}{Lp \xrightarrow{\delta:} Lp'}$ | |

Es ist ganz gut zu sehen, welche Operatoren die beiden Zeiten vergehen lassen, und wie die beiden Operatoren $L$ und $N$ arbeiten. Interessant ist aber, daß sowohl $L$ als auch $N$ im Gegensatz zum herkömmlichen Übersetzer $\Phi$ nicht symmetrisch sind. Für sie ist „Innen“ und „Außen“ ein echter Unterschied, wohingegen für $\Phi$ beides gleichwertig ist.

Der Parallelitätsoperator $|$ läßt nur dann die nichtändernde Zeit $\xrightarrow{\delta:}$ vergehen, wenn beide Subprozesse das tun. Wenn nur einer der beiden ändernde Zeit vergehen läßt, muß das natürlich auch der Parallelitätsoperator $|$ tun.

Es ist zu beachten, daß alle $\mathrm{TCBS}'$-Prozesse der Formen $Np$ und $Lp; p \in \mathbf{P}'_\alpha$ nicht nur, wie $\mathbf{P}'_\alpha$ selber, eine erweiterte $\alpha$-Prozeßmenge ausmachen, sondern sogar eine (einfache) $\alpha$-Prozeßmenge, d.h. sie haben keine Transitionen der Form $\xrightarrow{\delta::}$.

Es scheint nun für den Vergleich von zwei Protokollen nun sinnvoll, sie gekapselt von $L$ auf Äquivalenz bezüglich der schwachen Bisimulation zu testen.

Das folgende Lemma sichert nun, daß die zu Anfang des Kapitels gewünschte Eigenschaft von $\mathrm{TCBS}'$ bezüglich der zwei vorhandenen Zweitbegriffe auch erfüllt ist.

**Lemma 6.1**
***Sei** $p \in \mathbf{P}_\alpha$ **ein Prozeß. Dann gilt:***

$$\mathrm{TCBS}'(p) \xrightarrow{\delta:} \mathrm{TCBS}'(p') \Longrightarrow p \sim p' \ \wedge \ \mathrm{TCBS}'(p) \not\xrightarrow{::} \ \wedge \ \mathrm{TCBS}'(p) \not\xrightarrow{!}$$

**Beweis**
Dieser Beweis erfolgt nach dem im Abschnitt 3.8 auf Seite 30 gegebenen Beweisschema einzeln für die Operatoren von TCBS. Das ist möglich, da in $\mathrm{TCBS}'(p)$ keine anderen Operatoren als in TCBS vorkommen und auch die zur Rechtfertigung des Schemas verwendeten Transitionsregeln für den Define-Operator den neuen Gegebenheiten angepaßt sind. Nun die einzelnen Beweise:

- **Hear :** Es gilt: $\mathrm{TCBS}'(?f) \xrightarrow{\delta:} \mathrm{TCBS}'(?f)$. Es ist $?f \sim ?f$ und außerdem $\mathrm{TCBS}'(?f) \not\xrightarrow{::}$ und $\mathrm{TCBS}'(?f) \not\xrightarrow{!}$.
- **Talk :** Es gilt: $\mathrm{TCBS}'(f\&<v,p>) \not\xrightarrow{:}$
- **Compose :** Aus $\mathrm{TCBS}'(p|q) \xrightarrow{\delta:} \mathrm{TCBS}'(p'|q')$ folgt: $\mathrm{TCBS}'(p) \xrightarrow{\delta:} \mathrm{TCBS}'(p')$ und $\mathrm{TCBS}'(q) \xrightarrow{\delta:} \mathrm{TCBS}'(q')$ und damit nach Voraussetzung (für $p$ und $q$ gilt die Aussage) $p \sim p', q \sim q'$ und $\mathrm{TCBS}'(p) \not\xrightarrow{::} \ \wedge \ \mathrm{TCBS}'(p) \not\xrightarrow{!}$, sowie $\mathrm{TCBS}'(q) \not\xrightarrow{::} \ \wedge \ \mathrm{TCBS}'(q) \not\xrightarrow{!}$. Aus den beiden letztgenannten folgt sofort, daß $\mathrm{TCBS}'(p|q) \not\xrightarrow{::} \ \wedge \ \mathrm{TCBS}'(p|q) \not\xrightarrow{!}$. Weil $\sim$ ein Kongruenzoperator ist, gilt auch: $(p|q) \sim (p'|q')$
- **Translate :** Aus $\mathrm{TCBS}'(\Phi p_\beta) \xrightarrow{\delta:} \mathrm{TCBS}'(\Phi p'_\beta)$ folgt: $\mathrm{TCBS}'(p_\beta) \xrightarrow{\delta:} \mathrm{TCBS}'(p'_\beta)$. Für $p_\beta$ gilt die Aussage und so folgt: $p_\beta \sim p'_\beta$ und $\mathrm{TCBS}'(p_\beta) \not\xrightarrow{::} \ \wedge \ \mathrm{TCBS}'(p_\beta) \not\xrightarrow{!}$. Weil $\sim$ eine Kongruenzrelation ist, und wegen der Transitionsregeln für den Translate-Operator folgt nun $\Phi p_\beta \sim \Phi p'_\beta$ sowie $\mathrm{TCBS}'(\Phi p_\beta) \not\xrightarrow{::} \ \wedge \ \mathrm{TCBS}'(\Phi p_\beta) \not\xrightarrow{!}$.
- **Timeout :** Aus $\mathrm{TCBS}'(\eta{:}p) \xrightarrow{\delta:} \mathrm{TCBS}'(p')$ folgt: $\eta = 0$ und damit $\eta{:}p \sim p$. Da die Aussage für $p$ gilt, gilt sie auch für $\eta{:}p$.
- **Delay :** analog zum Beweis für den Timeout-Operator.

■

Die gewünschte Eigenschaft, daß ein Prozeß, der unverändernde Zeit (Transitionen der Form $\xrightarrow{\delta:}$) vergehen läßt, unverändert bleibt, ist hiermit bewiesen. Außerdem gilt, daß ein solcher Prozeß

- keine verändernde Zeit (Transitionen der Form $\xrightarrow{\delta::}$) vergehen lassen kann und
- nicht in der Lage ist, etwas zu sagen.

Aus beidem kann man folgern, daß Prozeß aus $\mathrm{TCBS}'$ genau dann unverändernde Zeit vergehen lassen kann, wenn er ausschließlich auf Eingaben wartet. Damit wird die Eignung dieses Kalküls für den Test auf Korrektheit von Protokollen deutlich. Denn zwei Protokolle (durch den Operator $L$ gekapselt) sind dann im gleichen Zustand, wenn sie beide auf Eingaben warten. Dann können sie auch beide beliebig viel Zeit vergehen lassen. Das ist wichtig, damit sie beide schwach bisimuliert sind. Während die beiden Protokolle jedoch Daten übertragen, ändern sich ihre Zustände solange, bis sie die Daten an den Empfänger geleitet haben und auf neue Eingaben warten. In dieser Zeit jedoch läßt der Operator $L$ keine Zeit vergehen, da sich der Zustand ändert, sondern er sagt nur $\tau$. Davon kann die schwache Bisimulation aber absehen.

**Satz 6.2**
***Sei*** $p \in \mathbf{P}_\alpha$ ***ein*** $\mathrm{TCBS}$***-Prozeß und*** $p_E = \mathrm{TCBS}'(p) \in \mathbf{P}'_\alpha$ ***der entsprechende*** $\mathrm{TCBS}'$***-Prozeß. Dann gilt:***

$$N(p_E) \sim p$$

**Beweis**
Als erstes wird nun die Relation $\mathcal{R}$ naheliegend als

$$\mathcal{R} = \left\{ (p, N(\mathrm{TCBS}'(p))) \;\middle|\; p \in \mathbf{P}_\alpha \right\}$$

definiert. Nun muß gezeigt werden, daß $\mathcal{R}$ eine starke Bisimulation ist, also daß für alle $u\ddagger \in \alpha_\tau \times \{?, !\} \cup \mathbb{N} \times \{:\}$ gilt:

- Wenn $p \xrightarrow{u\ddagger} p'$, dann gilt: $N(\mathrm{TCBS}'(p)) \xrightarrow{u\ddagger} N(\mathrm{TCBS}'(p'))$.
- Wenn $p_E = \mathrm{TCBS}'(p)$ und $N(p_E) \xrightarrow{u\ddagger} N(p'_E)$, dann gilt: $p \xrightarrow{u\ddagger} p'$ und $p'_E = \mathrm{TCBS}'(p')$.

Für $\ddagger \in \{!, ?\}$ ist wegen $\dfrac{p \xrightarrow{w?} p'}{Np \xrightarrow{w?} Np'}$ und $\dfrac{p \xrightarrow{w!} p'}{Np \xrightarrow{w!} Np'}$ also folgendes zu zeigen:

- Wenn $p \xrightarrow{w\ddagger} p'$, dann gilt: $\mathrm{TCBS}'(p) \xrightarrow{w\ddagger} \mathrm{TCBS}'(p')$.
- Wenn $p_E = \mathrm{TCBS}'(p)$ und $p_E \xrightarrow{w\ddagger} p'_E$, dann gilt: $p \xrightarrow{w\ddagger} p'$ und $p'_E = \mathrm{TCBS}'(p')$.

Das ist jedoch klar, da alle für die Transitionen $\xrightarrow{w!}$ und $\xrightarrow{w?}$ geltenden Regeln in der Semantik des erweiterten Kalküls unverändert geblieben sind (siehe Definition 6.3 auf Seite 46). Für $\ddagger =:$ ist wegen $\dfrac{p \xrightarrow{\delta\ddagger} p'}{Np \xrightarrow{\delta:} Np'}$; $\ddagger \in \{:, ::\}$ folgendes zu zeigen:

- Wenn $p \xrightarrow{\delta:} p'$, dann gilt: $\mathrm{TCBS}'(p) \xrightarrow{\delta:} \mathrm{TCBS}'(p')$ oder $\mathrm{TCBS}'(p) \xrightarrow{\delta::} \mathrm{TCBS}'(p')$.
- Wenn $p_E = \mathrm{TCBS}'(p)$ und $p_E \xrightarrow{\delta:} p'_E$, dann gilt: $p \xrightarrow{\delta:} p'$ und $p'_E = \mathrm{TCBS}'(p')$.
- Wenn $p_E = \mathrm{TCBS}'(p)$ und $p_E \xrightarrow{\delta::} p'_E$, dann gilt: $p \xrightarrow{\delta:} p'$ und $p'_E = \mathrm{TCBS}'(p')$.

Diese 3 Aussagen werden jetzt im einzelnen jeweils induktiv über $\mathrm{Depth}(p)$ nach dem Schema aus Abschnitt 3.8 auf Seite 30 bewiesen. Dabei sind die Beweise für die Operatoren Hear und Talk klar, da für sie alle Transitionsregeln in $\mathrm{TCBS}'$ denen von $\mathrm{TCBS}$ entsprechen. Außerdem sind die

Regeln für Timeout und Delay die die Transitionen der Form $\xrightarrow{\delta:}$ bzw. $\xrightarrow{\delta::}$ betreffen, identisch. Also braucht der Beweis bloß für z.B. Delay durchgeführt zu werden und gilt dann automatisch auch für Timeout. Im folgenden werden die Aussagen deshalb ausschließlich für die Operatoren Compose, Translate und Delay bewiesen:

(i) Wenn $p \xrightarrow{\delta:} p'$, dann gilt: $\mathrm{TCBS}'(p) \xrightarrow{\delta:} \mathrm{TCBS}'(p')$ oder $\mathrm{TCBS}'(p) \xrightarrow{\delta::} \mathrm{TCBS}'(p')$:

- **Compose :** Vorausgesetzt ist $p|q \xrightarrow{\delta:} p'|q'$ und also $p \xrightarrow{\delta:} p'$ und $q \xrightarrow{\delta:} q'$. Angenommen es gilt: $\mathrm{TCBS}'(p) \xrightarrow{\delta:} \mathrm{TCBS}'(p')$ und $\mathrm{TCBS}'(q) \xrightarrow{\delta:} \mathrm{TCBS}'(q')$. Dann gilt auch:

  $$\mathrm{TCBS}'(p|q) \xrightarrow{\delta:} \mathrm{TCBS}'(p'|q').$$

  Ansonsten gilt, weil $p$ und $q$ die Behauptung des Satzes ja erfüllen o.B.d.A, daß $\mathrm{TCBS}'(p) \xrightarrow{\delta::} \mathrm{TCBS}'(p')$ und entweder $\mathrm{TCBS}'(q) \xrightarrow{\delta::} \mathrm{TCBS}'(q')$ oder $\mathrm{TCBS}'(q) \xrightarrow{\delta:} \mathrm{TCBS}'(q')$. Daraus folgt aber, daß

  $$\mathrm{TCBS}'(p|q) \xrightarrow{\delta::} \mathrm{TCBS}'(p'|q').$$

- **Translate :** Vorausgesetzt ist $\Phi p_\beta \xrightarrow{\delta:} \Phi p'_\beta$ und also $p_\beta \xrightarrow{\delta:} p'_\beta$. Da für $p_\beta$ die Behauptung gilt, folgt: $\mathrm{TCBS}'(p_\beta) \xrightarrow{\delta:} \mathrm{TCBS}'(p'_\beta)$ oder $\mathrm{TCBS}'(p_\beta) \xrightarrow{\delta::} \mathrm{TCBS}'(p'_\beta)$ und damit entweder

  $$\mathrm{TCBS}'(\Phi p_\beta) \xrightarrow{\delta:} \mathrm{TCBS}'(\Phi p'_\beta)$$

  oder

  $$\mathrm{TCBS}'(\Phi p_\beta) \xrightarrow{\delta::} \mathrm{TCBS}'(\Phi p'_\beta).$$

- **Delay :** Vorausgesetzt ist $\eta.p \xrightarrow{\delta:} p'$. Entweder gilt $\eta = 0$ und damit $p \xrightarrow{\delta:} p'$ oder $\eta \neq 0$ und damit $\eta.p \xrightarrow{\delta:} (\eta - \delta).p$. Im ersten Fall folgt die Behauptung unmittelbar, da sie für $p$ gilt. Im zweiten Fall gilt:

  $$\mathrm{TCBS}'(\eta.p) \xrightarrow{\delta::} \mathrm{TCBS}'((\eta - \delta).p).$$

(ii) Wenn $p_E = \mathrm{TCBS}'(p)$ und $p_E \xrightarrow{\delta:} p'_E$, dann gilt: $p \xrightarrow{\delta:} p'$ und $p'_E = \mathrm{TCBS}'(p')$:

- **Compose :** Vorausgesetzt ist: $p_E|q_E = \mathrm{TCBS}'(p|q)$ und $p_E|q_E \xrightarrow{\delta:} p'_E|q'_E$. Das heißt, daß $p_E = \mathrm{TCBS}'(p)$ und $q_E = \mathrm{TCBS}'(q)$ sowie $p_E \xrightarrow{\delta:} p'_E$ und $q_E \xrightarrow{\delta:} q'_E$ und also, weil die Aussage für $p$ und $q$ gilt, $p \xrightarrow{\delta:} p'$ und $q \xrightarrow{\delta:} q'$ sowie $p'_E = \mathrm{TCBS}'(p')$ und $q'_E = \mathrm{TCBS}'(q')$. Daraus folgt:

  $$p|q \xrightarrow{\delta:} p'|q'$$

  und $p'_E|q'_E = \mathrm{TCBS}'(p'|q')$.

- **Translate :** Vorausgesetzt ist: $\Phi p_{\beta,E} = \mathrm{TCBS}'(\Phi p_\beta)$ und $\Phi p_{\beta,E} \xrightarrow{\delta:} \Phi p'_{\beta,E}$. Das heißt, daß $p_{\beta,E} \xrightarrow{\delta:} p'_{\beta,E}$. Da die Behauptung für $p_\beta$ gilt, folgt: $p_\beta \xrightarrow{\delta:} p'_\beta$ und $p'_{\beta,E} = \mathrm{TCBS}'(p'_\beta)$. Daraus wiederum folgt:

  $$\Phi p_\beta \xrightarrow{\delta:} \Phi p'_\beta$$

  und $\Phi p'_{\beta,E} = \mathrm{TCBS}'(\Phi p'_\beta)$.

- **Delay :** Vorausgesetzt ist: $\eta.p_E = \mathrm{TCBS}'(\eta.p)$ und $\eta.p_E \xrightarrow{\delta:} p'_E$. Daraus folgt: $p_E = \mathrm{TCBS}'(p)$ sowie $\eta = 0$ und damit $p_E \xrightarrow{\delta:} p'_E$. Da die Behauptung für $p$ gilt, folgt: $p \xrightarrow{\delta:} p'$ sowie $p'_E = \mathrm{TCBS}'(p')$. Daraus folgt:

  $$0.p \xrightarrow{\delta:} p'$$

  und $p'_E = \mathrm{TCBS}'(p')$.

(iii) Wenn $p_E = \mathrm{TCBS}'(p)$ und $p_E \xrightarrow{\delta::} p'_E$, dann gilt: $p \xrightarrow{\delta:} p'$ und $p'_E = \mathrm{TCBS}'(p')$:

- **Compose :** Vorausgesetzt ist: $p_E|q_E = \mathrm{TCBS}'(p|q)$ und $p_E|q_E \xrightarrow{\delta::} p'_E|q'_E$. Das heißt, daß $p_E = \mathrm{TCBS}'(p)$ und $q_E = \mathrm{TCBS}'(q)$ sowie $p_E \xrightarrow{\delta\ddagger_1} p'_E$ und $q_E \xrightarrow{\delta\ddagger_2} q'_E$; $(\ddagger_1, \ddagger_2) \in \{(::,:),(:,::),(::,::)\}$ und also, weil die hier zu beweisende bzw. die vorhergehende Aussage für $p$ und $q$ gilt, $p \xrightarrow{\delta:} p'$ und $q \xrightarrow{\delta:} q'$ sowie $p'_E = \mathrm{TCBS}'(p')$ und $q'_E = \mathrm{TCBS}'(q')$. Daraus folgt:

  $$p|q \xrightarrow{\delta:} p'|q'$$

  und $p'_E|q'_E = \mathrm{TCBS}'(p'|q')$.
- **Translate :** Vollkommen analog zum Beweis von Aussage (ii) für den Translate-Operator.
- **Delay :** Vorausgesetzt ist: $\eta.p_E = \mathrm{TCBS}'(\eta.p)$ und $\eta.p_E \xrightarrow{\delta:} p'_E$. Daraus folgt $p_E = \mathrm{TCBS}'(p)$ sowie entweder $\eta = 0$ und damit $p_E \xrightarrow{\delta:} p'_E$ bzw. $p_E \xrightarrow{\delta::} p'_E$ oder $\eta \neq 0$ und $\eta.p_E \xrightarrow{\delta::} (\eta - \delta).p'_E$. Da die hier zu beweisende bzw. die vorhergehende Aussage für $p$ gilt, folgt für den ersten Fall: $0.p \xrightarrow{\delta:} p'$ sowie $p'_E = \mathrm{TCBS}'(p')$. Daraus folgt:

  $$0.p \xrightarrow{\delta:} p'$$

  und $p'_E = \mathrm{TCBS}'(p')$. Im zweiten Fall folgt:

  $$\eta.p \xrightarrow{\delta:} (\eta - \delta).p'$$

  sowie $(\eta - \delta).p'_E = \mathrm{TCBS}'((\eta - \delta).p')$

■

Dieser Satz zeigt, daß die Semantik von TCBS, um $\mathrm{TCBS}'$ zu erhalten, nur so wenig abgeändert wurde, daß durch einfaches Ersetzen von $\xrightarrow{\delta::}$ durch $\xrightarrow{\delta:}$ alles beim Alten bleibt. Diese Eigenschaft ist natürlich wichtig, denn jetzt können alle Sätze für TCBS auch problemlos auf $\mathrm{TCBS}'$ übertragen werden.

# 7 Korrektheit von Protokollen

## 7.1 Allgemeine Bedingung

Nachdem ich nun im letzten Kapitel einen Kalkül eingeführt habe, der die von mir gewünschte Eigenschaft besitzt, von der Zeit, die ein Prozeß zur Erledigung seiner Aufgabe benötigt, absehen zu können, kann ich nun die im vorletzten Kapitel vorläufig angegebene Definition, wann ein Protokoll als korrekt gilt, endgültig spezifizieren.

**Definition 7.1 (korrektes Protokoll)**
***Ein in*** $\mathrm{TCBS}$ ***implementiertes Punkt-zu-Punkt-Einweg-Protokoll mit dem Sendeprozeß*** $p_A$ ***und dem Empfangsprozeß*** $p_B$***, heißt dann korrekt im Sinne der Freiheit von Fehlern und der Bewahrung der Reihenfolge, wenn gilt:***

$$\mathrm{Perfekt} \approx L(\Phi_{\mathrm{Prot}}(\mathrm{TCBS}'(p_A|\,\mathrm{Medium}_{N_B,V_A}\,|\,\mathrm{Medium}_{N_A,V_B}\,|p_B)))$$

$\mathrm{Perfekt}$ muß deshalb nicht von $L$ gekapselt sein, weil es keine Timeout- oder Delay-Operatoren enthält, die einzigen, die verändernde Zeit vergehen lassen können.

Wie weist man nun die Korrektheit eines solchen Protokolls nach? Man muß eine schwache Bisimulation finden. Dies ist in diesem Falle oft nicht so einfach, wie in vielen vorangegangenen Beweisen. Das bedeutet, daß man die Menge, die eine schwache Bisimulation werden soll, erst konstruieren muß. Das macht man aber nach einem ziemlich einfachen Algorithmus, wenn man die schwache Bisimuliertheit von $p$ und $q$ beweisen will:

(1) Setze $\mathcal{R} := \emptyset$ und $\mathcal{R}_{\mathrm{Try}} := \{(p,q)\}$.

(2) Ist $\mathcal{R}_{\mathrm{Try}} = \emptyset$, dann ist $\mathcal{R}$ die gesuchte schwache Bisimulation. Dann Abbruch.

(3) Ansonsten nimm ein Paar $(p,q)$ aus $\mathcal{R}_{\mathrm{Try}}$ heraus und zu $\mathcal{R}$ hinzu. Dann nimm solche Paare $(p',q')$ zu $\mathcal{R}_{\mathrm{Try}}$ hinzu, daß die Bedingungen für die schwache Bisimulation bezüglich $(p,q)$ (siehe Definition 4.4 auf Seite 38) erfüllt sind. Es muß also für alle $u\ddagger \in \alpha \times \{?,!\} \cup \mathbb{N} \times \{:\}$ gelten, daß:

- Wenn $p \xrightarrow{u\ddagger} p'$, dann $\exists q'$, so daß $q \xrightarrow{(\tau!)^*} \xrightarrow{u\ddagger} q'$ und $p' \mathcal{R}_{\mathrm{Try}} q'$ oder $p' \mathcal{R} q'$,
- Wenn $p \xrightarrow{\tau!} p'$, dann $\exists q'$, so daß $q \xrightarrow{(\tau!)^*} q'$ und $p' \mathcal{R}_{\mathrm{Try}} q'$ oder $p' \mathcal{R} q'$,
- Wenn $q \xrightarrow{u\ddagger} q'$, dann $\exists p'$, so daß $p \xrightarrow{(\tau!)^*} \xrightarrow{u\ddagger} p'$ und $p' \mathcal{R}_{\mathrm{Try}} q'$ oder $p' \mathcal{R} q'$,
- Wenn $q \xrightarrow{\tau!} q'$, dann $\exists p'$, so daß $p \xrightarrow{(\tau!)^*} p'$ und $p' \mathcal{R}_{\mathrm{Try}} q'$ oder $p' \mathcal{R} q'$.

(4) Ist eine der Bedingungen nicht zu erfüllen, sind die beiden Prozesse nicht schwach bisimuliert. Dann Abbruch.

(5) Weiter in Schritt (2).

Es kann unter Umständen passieren, daß dieser Algorithmus nicht abbricht. Das heißt jedoch auch, daß jede schwache Bisimulation, in der $(p,q)$ enthalten ist, unendlich groß ist.[6] Solche Bisimulation muß man auf andere Art und Weise suchen. Allerding ist es so, daß für viele Fälle dieser Algorithmus gute Dienste leistet. Er läßt sich natürlich auch sehr gut automatisieren. Genau hierin liegt aber die Stärke dieses formalen Kalküls und des formalen Korrektheitsbegriffs. Man braucht für den Beweis keine Intuition mehr (sie hilft natürlich), sondern er funktioniert nach einem festen einfachen Algorithmus und nur deswegen kann man ihn automatisieren.

Obwohl solche Beweise durch ihre Schematik nur von bedingtem Interesse sind, wird zur Erhellung des Algorithmus und zur Darstellung der Bedeutung der Arbeit im nächsten Abschnitt eine Implementation des Alternating-Bit-Protokolls vorgestellt, erläutert und schließlich der Beweis der Korrektheit im obigen Sinne an den entscheidenden Punkten vorgeführt.

[6] Es ist ein offenes Problem, ob solche Bisimulationen bei der von mit gegebenen Definition über Grammatiken unendlich groß sein können. Wenn man jedoch für die Sprachen $\alpha$ auch unendliche Mengen zuläßt, gibt es solche unendlichen Bisimulationen.

## 7.2 Das Alternating-Bit-Protokoll ist korrekt

Das Alternating-Bit-Protokoll (AB-Protokoll) ist eines der einfachsten Protokolle, die Daten über ein unsicheres Medium[7] transportieren können. Es arbeitet auf folgende Weise. Der sendende und der empfangende Prozeß sind jeweils mit einem Bit (also Eins oder Null) versehen. Zu Beginn sind beide mit demselben Bit markiert.

Der sendende Prozeß wartet auf Daten, markiert diese mit seinem aktuellen Bit, sendet sie zusammen los und wartet auf die Bestätigung des empfangenden Prozesses, daß die Daten angekommen sind. Diese Bestätigung muß mit demselben Bit markiert sein. Wenn nicht innerhalb einer gewissen Zeit eine Bestätigung erfolgt, dann werden die Daten nochmal übertragen. Wenn die Bestätigung erfolgt ist, dann wird das mit dem sendenden Prozeß verbundene Bit negiert und der Vorgang beginnt von Neuem.

Der empfangende Prozeß wartet auf eingehende Daten. Dann sendet er eine Bestätigung mit dem empfangenen Bit zurück. Sind die Daten mit seinem aktuellen Bit markiert gewesen, dann leitet er sie weiter und negiert sein aktuelles Bit. Danach wartet er wieder auf eingehende Daten.

Man kann sich leicht vorstellen, daß dieses Verfahren funktioniert. Aber es bleiben Zweifel, ob es nicht vielleicht zu Konstellationen kommen kann, die das Protokoll zum Versagen bringen könnten. Diese Zweifel sind bei komplizierteren Protokollen (die meisten Protokolle sind wesentlich komplizierter) noch stärker. Die formale Bestätigung der Korrektheit mit Hilfe von Definition 7.1 auf der vorherigen Seite gibt aber die gewünschte Sicherheit.

Dazu muß das AB-Protokoll jedoch als ein Paar von TCBS-Prozessen $p_A$ und $p_B$ implementiert werden. Dieses geschieht nun. Der Leser möge die Definition mit der Beschreibung von oben vergleichen:

Die von dem Protokoll intern verwendete Sprache ist

$$\beta := \{0,1\} \times \alpha \cup \{0,1\}.$$

und die auf dem Medium gesprochene Sprache ist, wie schon im Kapitel 5 ausgeführt wurde :

$$\gamma := \{A,B\} \times \alpha \cup \{V_A, V_B, N_A, N_B\} \times \beta.$$

Der sendende Prozeß und sein Folgeprozeß sind wie folgt definiert:

$$\mathrm{Sender}_{\mathrm{Bit}} :=? f_{\mathrm{Sender}_{\mathrm{Bit}}}$$

$$f_{\mathrm{Sender}_{\mathrm{Bit}}}(v) := \begin{cases} !{<}(N_B, \mathrm{Bit}, d), \mathrm{Timeout}_{\mathrm{Bit},d}{>} & \text{falls } v = (A,d) \\ \mathrm{Sender}_{\mathrm{Bit}} & \text{sonst} \end{cases}$$

$$\mathrm{Timeout}_{\mathrm{Bit},d} := 1{:}f_{\mathrm{Timeout}_{\mathrm{Bit},d}}\&{<}(N_B, \mathrm{Bit}, d), \mathrm{Timeout}_{\mathrm{Bit},d}{>}$$

$$f_{\mathrm{Timeout}_{\mathrm{Bit},d}} := \begin{cases} \mathrm{Sender}_{\neg\,\mathrm{Bit}} & \text{falls } v = (V_B, \mathrm{Bit}) \\ \mathrm{Timeout}_{\mathrm{Bit},d} & \text{sonst} \end{cases}$$

Der empfangende Prozeß ist wie folgt definiert:

$$\mathrm{Empfänger}_{\mathrm{Bit}} :=? f_{\mathrm{Empfänger}_{\mathrm{Bit}}}$$

$$f_{\mathrm{Empfänger}_{\mathrm{Bit}}}(v) := \begin{cases} !{<}(N_A, \mathrm{Bit}), !{<}(B,d), \mathrm{Empfänger}_{\neg\,\mathrm{Bit}}{>}{>} & \text{falls } v = (V_A, \mathrm{Bit}, d) \\ !{<}(N_A, \neg\,\mathrm{Bit}), \mathrm{Empfänger}_{\mathrm{Bit}}{>} & \text{falls } v = (V_A, \neg\,\mathrm{Bit}, d) \\ \mathrm{Empfänger}_{\mathrm{Bit}} & \text{sonst} \end{cases}$$

[7] In diesem Fall darf das Medium Nachrichten vertauschen, verschlucken oder generieren. Das Verstümmeln von Nachrichten ist nicht erlaubt.

Dann gilt:

$$p_A := \mathrm{Sender}_{\mathrm{Bit}}$$
$$p_B := \mathrm{Empfänger}_{\mathrm{Bit}}$$

Um die Korrektheit des AB-Protokolls nachzuweisen, ist also folgendes zu zeigen:

$$\mathrm{Perfekt} \approx L(\Phi_{\mathrm{Prot}}(\mathrm{Sender}_{\mathrm{Bit}} \,|\, \mathrm{Medium}_{N_B,V_A} \,|\, \mathrm{Medium}_{N_A,V_B} \,|\, \mathrm{Empfänger}_{\mathrm{Bit}}))$$

Dabei sei jetzt $\mathrm{Sender}_{\mathrm{Bit}} \,|\, \mathrm{Medium}_{N_B,V_A} \,|\, \mathrm{Medium}_{N_A,V_B} \,|\, \mathrm{Empfänger}_{\mathrm{Bit}}$ ein TCBS′-Prozeß. Zum Nachweis der schwachen Bisimuliertheit verwenden wir oben angegebenen Algorithmus. Als erstes gilt also:

$$\mathcal{R} := \emptyset,$$
$$\mathcal{R}_{\mathrm{Try}} := \{(\mathrm{Perfekt}, L(\Phi_{\mathrm{Prot}}(\mathrm{Sender}_{\mathrm{Bit}} \,|\, \mathrm{Medium}_{N_B,V_A} \,|\, \mathrm{Medium}_{N_A,V_B} \,|\, \mathrm{Empfänger}_{\mathrm{Bit}})))\}.$$

Nun nehmen wir ein Element $(p, q)$ aus $\mathcal{R}_{\mathrm{Try}}$ heraus und zu $\mathcal{R}$ hinzu und testen, ob die Bedingungen für die schwache Bisimulation durch Hinzunahme von Elementen zu $\mathcal{R}_{\mathrm{Try}}$ erfüllbar sind. Das sind ($u\ddagger \in \alpha \times \{?, !\} \cup \mathbb{N} \times \{:\}$):

- Wenn $p \xrightarrow{u\ddagger} p'$, dann $\exists q'$, so daß $q \xrightarrow{(\tau!)^*} \xrightarrow{u\ddagger} q'$ und $p' \mathcal{R}_{\mathrm{Try}} q'$ oder $p' \mathcal{R} q'$,
- Wenn $p \xrightarrow{\tau!} p'$, dann $\exists q'$, so daß $q \xrightarrow{(\tau!)^*} q'$ und $p' \mathcal{R}_{\mathrm{Try}} q'$ oder $p' \mathcal{R} q'$,
- Wenn $q \xrightarrow{u\ddagger} q'$, dann $\exists p'$, so daß $p \xrightarrow{(\tau!)^*} \xrightarrow{u\ddagger} p'$ und $p' \mathcal{R}_{\mathrm{Try}} q'$ oder $p' \mathcal{R} q'$,
- Wenn $q \xrightarrow{\tau!} q'$, dann $\exists p'$, so daß $p \xrightarrow{(\tau!)^*} p'$ und $p' \mathcal{R}_{\mathrm{Try}} q'$ oder $p' \mathcal{R} q'$.

Das einzige Element in $\mathcal{R}_{\mathrm{Try}}$ ist im Moment

$$(p, q) = (\mathrm{Perfekt}, L(\Phi_{\mathrm{Prot}}(\mathrm{Sender}_{\mathrm{Bit}} \,|\, \mathrm{Medium}_{N_B,V_A} \,|\, \mathrm{Medium}_{N_A,V_B} \,|\, \mathrm{Empfänger}_{\mathrm{Bit}}))).$$

$p = \mathrm{Perfekt}$ hat folgende Transitionen

$$\mathrm{Perfekt} \xrightarrow{[\delta:,v?]} \mathrm{Perfekt} \text{ für } v \neq (A, d)$$
$$\mathrm{Perfekt} \xrightarrow{(A,d)?} !{<}(B, d), \mathrm{Perfekt} >$$

Diese entsprechen den folgenden Regeln für $q$:

$$L(\Phi_{\mathrm{Prot}}(\mathrm{Sender}_{\mathrm{Bit}} \,|\, \mathrm{Medium}_{N_B,V_A} \,|\, \mathrm{Medium}_{N_A,V_B} \,|\, \mathrm{Empfänger}_{\mathrm{Bit}}))$$
$$\xrightarrow{[\delta:,v?]} L(\Phi_{\mathrm{Prot}}(\mathrm{Sender}_{\mathrm{Bit}} \,|\, \mathrm{Medium}_{N_B,V_A} \,|\, \mathrm{Medium}_{N_A,V_B} \,|\, \mathrm{Empfänger}_{\mathrm{Bit}})) \text{ mit } v \neq (A, d)$$

$$L(\Phi_{\mathrm{Prot}}(\mathrm{Sender}_{\mathrm{Bit}} \,|\, \mathrm{Medium}_{N_B,V_A} \,|\, \mathrm{Medium}_{N_A,V_B} \,|\, \mathrm{Empfänger}_{\mathrm{Bit}}))$$
$$\xrightarrow{(A,d)?} L(\Phi_{\mathrm{Prot}}(!{<}(N_B, \mathrm{Bit}, d), \mathrm{Timeout}_{\mathrm{Bit},d} > | \, \mathrm{Medium}_{N_B,V_A} \,|\, \mathrm{Medium}_{N_A,V_B} \,|\, \mathrm{Empfänger}_{\mathrm{Bit}}))$$

Für diese gefundene Übereinstimmung der Transitionen muß nun noch das Element

$$(!<(B,d),\text{Perfekt}>, L(\Phi_{\text{Prot}}(!<(N_B,\text{Bit},d),\text{Timeout}_{\text{Bit},d}>| \\ \text{Medium}_{N_B,V_A} | \text{Medium}_{N_A,V_B} | \text{Empfänger}_{\text{Bit}})))$$

zu $\mathcal{R}_{\text{Try}}$ hinzugenommen werden.

Jetzt beginnt der Algorithmus von vorne. Dabei ist:

$$\mathcal{R}_{\text{Try}} := \{(!<(B,d),\text{Perfekt}>, \\ L(\Phi_{\text{Prot}}(!<(N_B,\text{Bit},d),\text{Timeout}_{\text{Bit},d}>| \text{Medium}_{N_B,V_A} | \text{Medium}_{N_A,V_B} | \text{Empfänger}_{\text{Bit}})))\}$$
$$\mathcal{R} := \{(\text{Perfekt}, \\ L(\Phi_{\text{Prot}}(\text{Sender}_{\text{Bit}} | \text{Medium}_{N_B,V_A} | \text{Medium}_{N_A,V_B} | \text{Empfänger}_{\text{Bit}})))\}$$

Da $\mathcal{R}$ immer größer wird, wird es im folgenden nicht nach jedem Schritt angegeben. Nun betrachten wir $(p,q) \in \mathcal{R}_{\text{Try}}$, und nehmen es zu $\mathcal{R}$ hinzu und aus $\mathcal{R}_{\text{Try}}$ heraus. Die möglichen Transitionen für $p$ und $q$ sind:

$$p \xrightarrow{v?} p$$
$$p \xrightarrow{(B,d)!} \text{Perfekt}$$
$$q \xrightarrow{v?} q$$
$$q \xrightarrow{\tau!} L(\Phi_{\text{Prot}}(\text{Timeout}_{\text{Bit},d} | \ddot{\text{U}}\text{bertrage}_{V_A,\text{Bit},d} | \text{Medium}_{N_B,V_A} | \text{Medium}_{N_A,V_B} | \text{Empfänger}_{\text{Bit}}))\ (= q')$$

Die jeweils ersten Zweige für $p$ bzw. $q$ entsprechen sich, ohne daß man ein neues Paar $(p',q')$ zu $\mathcal{R}_{\text{Try}}$ hinzunehmen müßte. Da $p$ überhaupt nicht $\tau$ sagen kann, muß $(p,q')$ zu $\mathcal{R}_{\text{Try}}$ hinzugenommen werden. Dann müssen wir noch nach einer mit $\tau!^*$ beginnenden Transitionsfolge für $q$ suchen, die zum Schluß $(B,d)!$ sagt, damit dieser Fall $(p,q)$ fertig abgehandelt ist. Diese Folge ist hier zur besseren Verständlichkeit ohne die Übersetzer $L$ und $\Phi_{\text{Prot}}$ angegeben, die alle bis auf die letzte Transition in $\tau!$ übersetzen, wie man leicht nachprüfen kann:

$$!<(N_B,\text{Bit},d),\text{Timeout}_{\text{Bit},d}>| \text{Medium}_{N_B,V_A} | \text{Medium}_{N_A,V_B} | \text{Empfänger}_{\text{Bit}}$$
$$\xrightarrow{(N_B,\text{Bit},d)!} \text{Timeout}_{\text{Bit},d} | \ddot{\text{U}}\text{bertrage}_{V_A,\text{Bit},d} | \text{Medium}_{N_B,V_A} | \text{Medium}_{N_A,V_B} | \text{Empfänger}_{\text{Bit}}$$
$$\xrightarrow{(V_A,\text{Bit},d)!} \text{Timeout}_{\text{Bit},d} | \text{Medium}_{N_B,V_A} | \text{Medium}_{N_A,V_B} |!<(N_A,\text{Bit}),!<(B,d),\text{Empfänger}_{\neg\,\text{Bit}}>>$$
$$\xrightarrow{(N_A,\text{Bit})!} \text{Timeout}_{\text{Bit},d} | \text{Medium}_{N_B,V_A} | \ddot{\text{U}}\text{bertrage}_{V_B,\text{Bit}} | \text{Medium}_{N_A,V_B} |!<(B,d),\text{Empfänger}_{\neg\,\text{Bit}}>$$
$$\xrightarrow{(V_B,\text{Bit})!} \text{Sender}_{\neg\,\text{Bit}} | \text{Medium}_{N_B,V_A} | \text{Medium}_{N_A,V_B} |!<(B,d),\text{Empfänger}_{\neg\,\text{Bit}}>$$
$$\xrightarrow{(B,d)!} \text{Sender}_{\neg\,\text{Bit}} | \text{Medium}_{N_B,V_A} | \text{Medium}_{N_A,V_B} | \text{Empfänger}_{\neg\,\text{Bit}}$$

Das Paar

$$(\text{Perfekt}, \text{Sender}_{\neg\,\text{Bit}} | \text{Medium}_{N_B,V_A} | \text{Medium}_{N_A,V_B} | \text{Empfänger}_{\neg\,\text{Bit}})$$

ist jedoch schon in $\mathcal{R}$ enthalten, da $\text{Bit}$ ja eine Variable ist.

Nun folgt der nächste Schritt, dabei ist das aktuell zu untersuchende Paar $(p, q)$ gleich

$$(!<(B,d), \text{Perfekt} >, L(\Phi_{\text{Prot}}(\text{Timeout}_{\text{Bit},d} \,|\, \ddot{\text{U}}\text{bertrage}_{V_A,\text{Bit},d} \,| \\ \text{Medium}_{N_B,V_A} \,|\, \text{Medium}_{N_A,V_B} \,|\, \text{Empfänger}_{\text{Bit}}))).$$

Die Transitionen sind nun für $p$ wie im letzten Schritt und für $q$ sind sie:

$$q \xrightarrow{v?} q$$
$$q \xrightarrow{\tau!} L(\Phi_{\text{Prot}}(\text{Timeout}_{\text{Bit},d} \,|\, \text{Medium}_{N_B,V_A} \,|\, \text{Medium}_{N_A,V_B} \,| \\ !<(N_A, \text{Bit}), !<(B,d), \text{Empfänger}_{\neg\,\text{Bit}} >>)) \; (= q')$$
$$q \xrightarrow{\tau!} L(\Phi_{\text{Prot}}(\text{Timeout}_{\text{Bit},d} \,|\, \text{Medium}_{N_B,V_A} \,|\, \text{Medium}_{N_A,V_B} \,|\, \text{Empfänger}_{\text{Bit}})) \; (= q'')$$

Es ist also analog zum letzten Fall wieder $(p, q')$ und $(p, q'')$ zu $\mathcal{R}_{\text{Try}}$ hinzuzunehmen, und zu testen, ob es eine Transitionsfolge, die $q$ wieder $(B, d)$ sagen läßt. Diese ist analog zu der oben gefundenen.

Im nächsten Schritt nun ist $\mathcal{R}_{\text{Try}}$ zum ersten Mal zweielementig. Wir nehmen für $(p, q)$ das Paar

$$(!<(B,d), \text{Perfekt} >, L(\Phi_{\text{Prot}}(\text{Timeout}_{\text{Bit},d} \,| \\ \text{Medium}_{N_B,V_A} \,|\, \text{Medium}_{N_A,V_B} \,|!<(N_A, \text{Bit}), !<(B,d), \text{Empfänger}_{\neg\,\text{Bit}} >>)).$$

Die Transitionen für $q$ sind nun:

$$q \xrightarrow{v?} q$$
$$q \xrightarrow{\tau!} L(\Phi_{\text{Prot}}(\text{Timeout}_{\text{Bit},d} \,| \\ \text{Medium}_{N_B,V_A} \,|\, \ddot{\text{U}}\text{bertrage}_{V_B,\text{Bit}} \,|\, \text{Medium}_{N_A,V_B} \,|!<(B,d), \text{Empfänger}_{\neg\,\text{Bit}} >)) \; (= q')$$

$(p, q')$ kommt wieder zu $\mathcal{R}_{\text{Try}}$ hinzu. Die Transitionsfolge für $q$, die mit $(B, d)!$ endet, findet man analog zu oben.

Diese Methode führt, wie man zwar zeitaufwendig, aber leicht nachprüfen kann, zu guter Letzt zum gewünschten Ergebnis, zur einer schwachen Bisimulation $\mathcal{R}$, in der das nach Definition benötigte Paar

$$(\text{Perfekt}, L(\Phi_{\text{Prot}}(\text{Sender}_{\text{Bit}} \,|\, \text{Medium}_{N_B,V_A} \,|\, \text{Medium}_{N_A,V_B} \,|\, \text{Empfänger}_{\text{Bit}})))$$

enthalten ist.

Ein weiterer interessanter Fall sei aber noch angeführt. Er demonstriert die Vorteile des erweiterten Kalküls für diese Methode des Beweisens.

Wie oben gesehen, ist auch $(p, q)$ gleich

$$(!<(B,d), \text{Perfekt} >, L(\Phi_{\text{Prot}}(\text{Timeout}_{\text{Bit},d} \,|\, \text{Medium}_{N_B,V_A} \,|\, \text{Medium}_{N_A,V_B} \,|\, \text{Empfänger}_{\text{Bit}})))$$

im Laufe des Verfahrens einmal in $\mathcal{R}_{\text{Try}}$.

Nun muß nachgewiesen werden, daß eine Transitionsfolge für $q$ gibt, die mit $\tau!^*$ beginnt und mit $(B, d)!$ endet. Diese sei hier wieder ohne die Übersetzer gegeben.

$$\begin{aligned}&\mathrm{Timeout}_{\mathrm{Bit},d} \,|\, \mathrm{Medium}_{N_B,V_A} \,|\, \mathrm{Medium}_{N_A,V_B} \,|\, \mathrm{Empfänger}_{\mathrm{Bit}}\\ \xrightarrow{1::} &!{<}(N_B, \mathrm{Bit}, d), \mathrm{Timeout}_{\mathrm{Bit},d}{>} |\, \mathrm{Medium}_{N_B,V_A} \,|\, \mathrm{Medium}_{N_A,V_B} \,|\, \mathrm{Empfänger}_{\mathrm{Bit}}\end{aligned}$$

Für diesen Prozeß haben wir aber schon gezeigt, daß er nach dem er viermal $\tau$ gesagt hat, auch $(B, d)$ sagt und der Ergebnisprozeß zu $\mathrm{Perfekt}$ in Relation $\mathcal{R}$ steht. Und durch den Übersetzer $L$ wird $1$ :: zu $\tau!$ übersetzt. Ohne den erweiterten Kalkül müßte $\mathrm{Perfekt}$ in diesem Fall auch Zeit vergehen lassen, was es nicht tut. Damit ist der Nachweis der schwachen Bisimuliertheit mit $\mathrm{Perfekt}$ nur für das von $L$ gekapselte Protokoll (inklusive unsicherer Medien) möglich.

## 7.3 Automatische Beweiser

Wie dieses Beispiel noch einmal deutlich demonstriert, sind für solche Aufgaben automatische Beweiser vonnöten. Ansonsten ist es mit zunehmender Problemgröße sehr schnell praktisch unmöglich, solche Beweise zu führen.

Als Beispiel betrachten wir kurz ein Send-Window-Protokoll. Dieses arbeitet ähnlich wie das AB-Protokoll, nur daß es nicht nur einen Wert lossendet, sondern halb so viele, wie sein Sendefenster groß ist, bevor es auf eine Bestätigung vom Empfänger wartet. Alle Daten sind mit einer Sequenznummer numeriert, um Fehler zu vermeiden und die Ordnung zu erhalten. Diese entspricht dem Bit im AB-Protokoll.

Wenn man z.B. ein Send-Window von nur 20 (ein vergleichsweise kleiner Wert) annimmt, dann werden 10 Werte nacheinander übertragen. Sie könnten alle verloren gehen oder ankommen. Das gibt zusammen $2^{10}$ Möglichkeiten. Das heißt, es treten mindestens $1024$ verschiedene Zustände des Prozesses $L(\Phi_{\mathrm{Prot}}(\mathrm{TCBS}'(p_A|\, \mathrm{Medium}_{N_B,V_A} \,|\, \mathrm{Medium}_{N_A,V_B} \,|p_B)))$ auf. Dadurch ist dieses Problem per Hand nicht mehr lösbar. Ein solcher automatischer Beweiser ist sicherlich nicht allzu kompliziert, man muß aber doch auf einige Punkte achten:

- Man müßte mit Daten (im obigen Beweis $d$ und $\mathrm{Bit}$) arbeiten können. Das ist aber durchaus kompliziert, da man ja nichts über die Berechnungen in den Funktionen $f$ des Talk- bzw. Hear-Operators sagen kann. Sie dürften also nicht allzu kompliziert sein, da ja der Beweiser alle Möglichkeiten der Entwicklung von Prozessen, die etwas hören, betrachten muß.
- Es ist schwer zu entscheiden, wann eine Bisimulation unendlich groß wird[8], und es deswegen keinen Sinn mehr hat, weiter zu testen.
- Im Zusammenhang mit den oberen beiden Punkten steht das Problem, daß es wünschenswert wäre, wenn Paare in $\mathcal{R}$ noch parametrisiert werden könnten, um zu verhindern, daß man für verschiedene Werte in den lokalen Speichern der Prozesse dieselben Rechnungen immer wieder ausführt. Im obigen Beweis waren $\mathrm{Bit}$ und $d$ eine solche Parametrisierung.

Es bleibt also auf dem Gebiet der formalen Schlüsse auf parallelen Kalkülen noch ein weiter Weg bis zum automatischen Beweis der Korrektheit des Internet-Protokolls. Weltweit beschäftigen sich jedoch sehr viele Forscher mit Problemen solcher Art. Es ist also davon auszugehen, daß aus diesem Gebiet der theoretischen Informatik bald sehr viele Anwendungen erwachsen, die sich als unentbehrlich erweisen werden.

[8] falls es solche unendliche Bisimulationen gibt

# Literatur

[BB93] Ludwig Balke and Karl Heinz Böhling. *Einführung in die Automatentheorie und Theorie formaler Sprachen*. Reihe Informatik. BI-Wissenschaftsverlag, Mannheim, Leipzig, Wien, Zürich, 1993.

[Hoa78] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 8(21):666–677, 1978.

[Mil80] Robin Milner. A calculus of communicating systems. *Lecture Notes in Computer Science*, Vol 92, 1980. Springer Verlag.

[Mil89] Robin Milner. *Communication and Concurrency*. Series in Computer Sience. Prentice Hall, Hertfordshire, 1989.

[Pra93] K. V. S. Prasad. Programming with broadcasts. *Lecture Notes in Computer Science*, Vol. 715, CONCUR, 1993. Springer Verlag.

[Pra95] K. V. S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25, 1995.

[Pra96] K. V. S. Prasad. Broadcasting in time. *Lecture Notes in Computer Science*, Vol. 1061, COORDINATION, 1996. Springer Verlag.

[Sha94] Robin Sharp. *Principles of Protocol Design*. Series in Computer Science. Prentice Hall, Hertfordshire, 1994.

[Wil95a] Sebastian Wilhelmi. A Simulated Network of Weather Stations. A Laboratory for the Course ”Dkom”, Chalmers University of Technology, URL: http://www.math-inf.uni-greifswald.de/˜wilhelmi/english/cbs/network.ps, 1995.

[Wil95b] Sebastian Wilhelmi. A TCBS-Implementation on C++. Project Report for the Course ”Parallelism”, Chalmers University of Technology, URL: http://www.math-inf.uni-greifswald.de/˜wilhelmi/english/cbs/tcbs.ps, 1995.