Modellieren Nebenläufiger und Verteilter Systeme mit Präemption

Gerald Lüttgen

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23681-2199, USA
luettgen@icase.edu
http://www.icase.edu/~luettgen

Aus unserem modernen Leben sind computerunterstützte Systeme nicht mehr wegzudenken. Wir müssen uns vor allem auf *sicherheitskritische Systeme*, wie sie z.B. in der Medizin oder in der Luftfahrt eingesetzt werden, verlassen können. Um deren Korrektheit hinreichend genau nachzuweisen, genügen ausgiebige Systemtests alleine nicht. Vielmehr wird ein mathematisch präziser Rahmen zur *Systemmodellierung* und *Systemverifikation* benötigt, wie ihn etwa *Prozeßalgebren* für die besonders diffizilen *nebenläufigen* und *verteilten Systeme* bieten.
Traditionelle Prozeßalgebren, z.B. Milners *Calculus of Communicating Systems* (CCS), beschränken sich auf die Modellierung des potentiellen *Nichtdeterminismus*, welcher durch die verzahnte Ausführung der Aktionen paralleler Prozesse hervorgerufen wird. Dabei wird meist von wichtigen Aspekten, wie z.B. *Prioritäten* und *Zeit*, abstrahiert, was ein exaktes Modellieren realer Systeme erschwert. Um diese Schwäche zu beheben, wurde in der Literatur das semantische Konzept der *Präemption* eingeführt, welches es ermöglicht, einigen Aktionen Präzedenz über anderen Aktionen zu geben. Existierende Erweiterungen klassischer Prozeßalgebren eignen sich vorwiegend zur Behandlung zentraler, aber nicht verteilter Systeme. In diesem Artikel skizzieren wir eine Erweiterung von CCS um Prioritäten, welche auch die Verteilung von Systemen berücksichtigt. Die dabei auftretenden semantischen Konzepte werden anhand eines einfachen Beispiels veranschaulicht.

1 Hintergrund

Unser alltägliches Leben wird immer stärker durch den Einfluß von mehr oder weniger komplexen Computersystemen geprägt, die von modernen Küchengeräten über Telefonanlagen bis hin zu Kontrollsystemen für die Flugüberwachung reichen. Gut ausgebildete Ingenieure sichern die Qualität ihrer Produkte durch intensives Testen. Dennoch werden Fehler, die nur selten und in speziellen Kontexten auftreten, oft erst nach der Produkteinführung bemerkt. Diese Situation ist für sicherheitskritische Systeme in der Medizin oder Luftfahrt, bei denen Leib und Leben von Menschen auf dem Spiel steht, untragbar. Testen alleine darf zur Qualitätskontrolle solcher Systeme nicht genügen. Vielmehr wird eine mathematische Analyse und Verifikation benötigt, um festzustellen, ob ein Produkt seinen Anforderungen genügt. *Formale Semantik* und *formale Verifikation* sind die Gebiete der Informatik, die sich mit mathematischen Grundlagen der Modellierung und Analyse von Systemen beschäftigen.
In der Literatur wurde vor allen Dingen die Semantik sequentieller Programme studiert, welche mathematisch mit Hilfe partieller Funktionen, die Eingabewerte auf Ausgabewerte abbilden, beschrieben werden können. Die Semantik *nebenläufiger* und *verteilter Systeme* hingegen steht zur Zeit im Blickpunkt der Forschung. Insbesondere verteilte Systeme gewinnen immer mehr an Bedeutung. Man denke nur an die stürmische Entwicklung des Internets, welches den weltweiten Austausch von Information ermöglicht. Im Unterschied zu sequentiellen Programmen, terminieren viele nebenläufige Systeme, etwa Betriebssysteme oder Kommunikationsprotokolle, nicht, sondern interagieren kontinuierlich mit ihrer Umgebung. Sie werden daher auch *reaktive Systeme* genannt, deren Semantikbeschreibung neue Techniken erfordert.

2 Klassische und Moderne Prozeßalgebren

Prozeßalgebren sind Formalismen zur *Modellierung* und *Verifikation* reaktiver Systeme. Prominente klassische Vertreter sind Milners *Calculus of Communicating Systems* (CCS) [Mil89] und Hoares *Communicating Sequential Processes* (CSP) [Hoa85]. Prozeßalgebren bestehen aus einer einfachen, termbasierten Beschreibungssprache zusammen mit einer mathematisch exakt definierten, meist *operationellen Semantik*. Ihre Stärken liegen in der Spezifikation von *Kommunikationsverhalten* und *Nichtdeterminismus* nebenläufiger und verteilter Systeme. Nichtdeterminismus entsteht sowohl durch Abstraktion von *Daten*, *Prioritäten* und *zeitlichen Aspekten*, als auch durch das Modellieren der Nebenläufigkeit paralleler Prozesse mittels verzahnter Ausführung, die auch als *Interleaving* bezeichnet wird.

In der Praxis stellt sich allerdings heraus, daß insbesondere Prioritäten und zeitliche Aspekte für die Verifikation des Kommunikationsverhaltens reaktiver Systeme von besonderem Interesse sind. Folglich wurden in der neueren Literatur Prozeßalgebren um diese Konzepte erweitert. Dabei blieb allerdings der Verteilungsaspekt reaktiver Systeme unberücksichtigt; z.B. wurden ausschließlich temporale Prozeßalgebren mit einem einzigen Taktgeber betrachtet. Als Konsequenz ergaben sich Semantiken, die zwar für zentrale aber nicht für verteilte Systeme adäquat sind.
Die diesem Artikel zugrunde liegende Dissertation entwickelt Erweiterungen von CCS um *verteilte Prioritäten* – zur Modellierung von Interrupts und prioritisierter Auswahl – und um *lokale Taktgeber* – zur Modellierung von verteilten Realzeitsystemen. Als Basis dazu dient ein kohärentes Studium des Konzeptes der *Präemption* für zentrale und verteilte Systeme, auf dem die Semantik für Prioritäten und Realzeit basiert. Präemption ermöglicht es auszudrücken, daß eine Kommunikation auf einigen Systemkanälen Vorrang vor Kommunikationen auf anderen Kanälen hat. In temporalen Prozeßalgebren spiegelt sich Präemption in der sogenannten *Synchronisationshypothese* wieder, die das Ticken eines Taktgebers solange verhindert, bis das betrachtete System keine internen Berechnungen mehr abarbeiten kann. Aus Platzgründen werden wir uns im folgenden exemplarisch mit Erweiterungen von CCS um Prioritätenkonzepte beschäftigen.

3 Die Prozeßalgebra CCS

Die klassische Prozeßalgebra CCS wurde von R. Milner in den achtziger Jahren entwickelt [Mil89]. In diesem Abschnitt motivieren wir CCS anhand eines Beispiels, auf das wir noch mehrmals zurückkommen werden.
Das Beispielsystem, dessen Architektur in Abbildung 1 dargestellt ist, modelliert ein typisches Szenario für den direkten Speicherzugriff (*Direct Memory Access*, DMA). Es besteht aus drei nebenläufigen Komponenten, sogenannten *Prozessen*: einem Prozessor (CPU) und zwei Speicherbänken (Bench1 und Bench2). Prozeß CPU kann über *Ports* fetch1 und fetch2 mit seiner Umgebung kommunizieren. Bench1, und analog Bench2, hat ebenfalls zwei Ports, $\overline{\text{fetch1}}$ und dma1. Komplementäre Ports, z.B. fetch1 und $\overline{\text{fetch1}}$, spannen einen *Kommunikationskanal* zwischen den jeweiligen Prozessen auf. Folglich gibt es in unserem Beispielsystem zwei Kanäle, die den Prozessor mit jeder Speicherbank verbinden. Dabei darf man sich vorstellen, daß der Prozeß mit dem „gequerten“ Port der *Sender* und der andere Prozeß der *Empfänger* der über den Kanal transferierten Nachrichten ist. In diesem Zusammenhang spricht man auch von einer *Handshakekommunikation*. In Abbildung 1 symbolisiert das

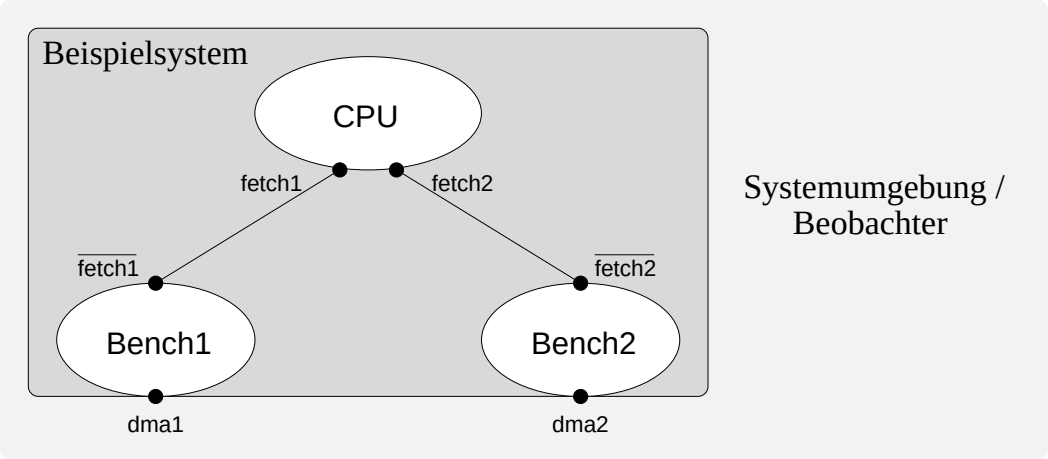


Abbildung 1: Direkter Speicherzugriff (DMA)

graue Rechteck, daß lediglich die Ports dma1 und dma2 für den außenstehenden *Beobachter* bzw. die *Systemumgebung* sichtbar sind; alle anderen Ports sind systemintern. Das erwünschte Systemverhalten ist intuitiv wie folgt. Prozeß CPU fordert abwechselnd ein Datum von Bench1 und Bench2 an. Jede Speicherbank kann entweder das entsprechende Datum über Port $\overline{\mathsf{fetch}i}$ senden, oder ein Datum über Port dma$i$ einlesen. Letztere Ports können etwa von einem DMA-Controller, welchen wir hier nicht explizit modellieren sondern in der Systemumgebung annehmen, angesprochen werden. In der Realität sollte eine Kommunikation auf Port fetch$i$ Vorrang vor einer Kommunikation auf Port dma$i$ haben, da man aus Effizienzgründen den Prozessor nicht warten lassen sollte. Solche Präferenzen können in CCS nicht ausgedrückt werden, da in dieser Algebra per Definition alle Ports gleichwertig sind. In Abschnitt 4 werden wir CCS um geeignete Prioritätenkonzepte erweitern.

3.1 Syntax und Semantik

CCS ist parametrisiert in der Menge Λ der Ports. Diese spannt die Menge $\mathcal{A} = \Lambda \cup \overline{\Lambda} \cup \{\tau\}$ der *Aktionen* auf, welche Ports, die zugehörigen komplementären Ports, sowie die ausgezeichnete *interne*, *unbeobachtbare Aktion* τ umfaßt. Letztere entsteht beim Nachrichtenaustausch zweier Prozesse mittels einer Kommunikation auf einem Kanal. Die Syntax von CCS ist wie folgt definiert, wobei $\alpha \in \mathcal{A}$, $L \subseteq \Lambda \cup \overline{\Lambda}$ und C ein Element aus einer abzählbaren Menge von *Prozeßvariablen* ist:

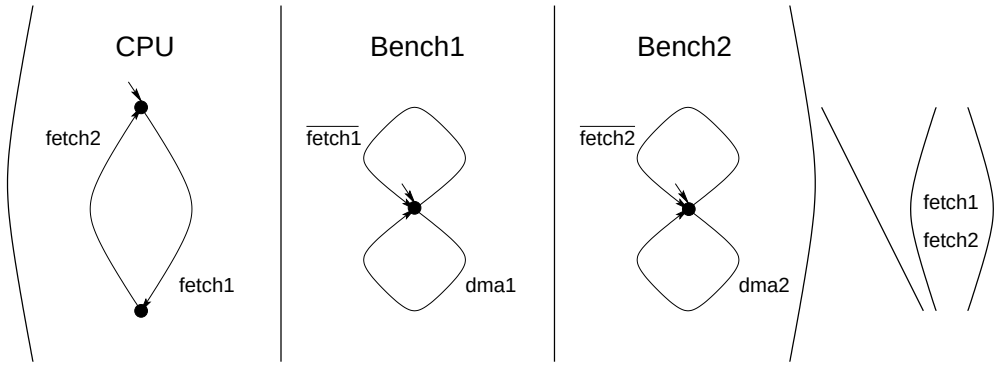
$$P \quad ::= \quad \mathbf{0} \quad | \quad \alpha.P \quad | \quad P + P \quad | \quad P \,|\, P \quad | \quad P \setminus L \quad | \quad C$$

Die Menge aller Prozeßterme wird mit $\mathcal{P}$ bezeichnet. Die Semantik eines Prozesses P ist ein mit Aktionen *beschriftetes Transitionssystem* $\langle \mathcal{P}, \mathcal{A}, \longrightarrow, P \rangle$, wobei $\mathcal{P}$ die Menge aller *Zustände*, $\mathcal{A}$ das *Alphabet*, $\longrightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ die *Transitionsrelation* und P der *Anfangszustand* ist. Die Transitionsrelation ist mit Hilfe *strukturierter operationeller Semantikregeln* induktiv definiert und spezifiziert das Verhalten von Prozessen. Wir schreiben $P \xrightarrow{\alpha} P'$ für eine Transition $\langle P, \alpha, P' \rangle \in \longrightarrow$ und sagen, daß sich *Prozeß P in Aktion α engagieren kann und sich anschließend wie Prozeß P' verhält.*
Prozeß $\mathbf{0}$ symbolisiert ein sich in einem *Deadlock* befindendes System ohne Transitionen. Prozeß $\alpha.P$ kann zunächst eine Kommunikation auf Port α anbieten, falls $\alpha \neq \tau$ gilt, oder im Falle $\alpha = \tau$ eine interne Berechnung ausführen, bevor er sich wie Prozeß P verhält. Prozeß $P + Q$ beschreibt *nichtdeterministische Auswahl* und kann sich entweder wie P oder wie Q verhalten. Prozeß $P \mid Q$ steht für die *parallele Komposition* von P und Q mittels einer *Interleavingsemantik* und synchroner Handshakekommunikation auf komplementären Ports. Der *Restriktionsoperator* $\backslash L$ verbietet in $P \backslash L$ alle mit einer Aktion aus $L \cup \overline{L}$ beschrifteten Transitionen. Darüber hinaus erlaubt CCS die Spezifikation *rekursiver* Prozesse mittels Gleichungsdefinitionen der Form $C \stackrel{\text{def}}{=} P$, wobei C eine in P auftretende Prozeßvariable ist.
Man sieht leicht, daß die Transitionssemantik für CCS „sehr fein“ ist, d.h. sie unterscheidet beispielsweise zwischen den Prozessen $P \mid Q$ und $Q \mid P$, obwohl diese isomorphe Transitionssysteme besitzen. Daher wird in der Regel eine *Verhaltensäquivalenz* auf Prozessen definiert, so daß dann die Semantik eines Prozesses die Äquivalenzklasse ist, welche ihn umschließt. In der Literatur wurden zahlreiche Äquivalenzen – von Isomorphie bis hin zu der aus der Automatentheorie bekannten Sprachäquivalenz – untersucht. Eine geeignete Äquivalenz sollte die folgenden Eigenschaften besitzen: (1) *Adäquatheit* für reaktive Systeme, (2) *Deadlocksensitivität*, d.h. äquivalente Prozesse besitzen das gleiche Deadlockverhalten, (3) *Kongruenz*, d.h. in einem System dürfen Prozesse durch äquivalente Prozesse ersetzt werden, (4) effiziente *Berechenbarkeit*. Eine Äquivalenz, die diese Eigenschaften erfüllt, ist die *Bisimulation* [Mil89]. Bisimulation ist die größte symmetrische Relation $\sim$ auf Prozessen, für die gilt: $P \sim Q \ \wedge \ P \xrightarrow{\alpha} P'$ impliziert $Q \xrightarrow{\alpha} Q' \ \wedge \ P' \sim Q'$ für einen Prozeß Q'.

3.2 Formalisierung des Beispielsystems

Abbildung 2 präsentiert die Spezifikation unseres Beispielsystems in CCS und veranschaulicht die Semantik der sequentiellen Prozesse CPU, Bench1, und Bench2. Mit Hilfe des Parallel- und Restriktionsoperators kann die Architektur aus Abbildung 1 eins zu eins reflektiert werden. Dabei drückt der Parallel-

operator die Nebenläufigkeit der Prozesse CPU, Bench1 und Bench2 aus. Der Restriktionsoperator $\backslash\{\mathsf{fetch1}, \mathsf{fetch2}\}$ spezifiziert, daß Ports fetch1 und fetch2 systemintern und daher von außen unbeobachtbar sind.



$$\mathsf{CPU} \stackrel{\text{def}}{=} \mathsf{fetch1.fetch2.CPU} \quad \mathsf{Bench1} \stackrel{\text{def}}{=} \overline{\mathsf{fetch1}}.\mathsf{Bench1} + \mathsf{dma1.Bench1} \quad \mathsf{Bench2} \stackrel{\text{def}}{=} \overline{\mathsf{fetch2}}.\mathsf{Bench2} + \mathsf{dma2.Bench2}$$

Abbildung 2: CCS-Spezifikation des Beispielsystems

Die Semantik des Beispielsystems ist in Abbildung 3, links, dargestellt. Das Transitionssystem besteht aus zwei Zuständen, die durch τ-Transitionen miteinander verbunden sind. Diese symbolisieren die zyklische Kommunikation zwischen CPU und Bench1 bzw. Bench2 über Kanäle fetch1 bzw. fetch2. Handshakekommunikationen sind in CCS unbeobachtbar und daher mit der internen Aktion τ beschriftet. In jedem der beiden Zustände bietet das System eine Kommunikation auf Ports dma1 und dma2 an. Leider erlaubt es uns CCS nicht auszudrücken, daß für eine Speicherbank eine Kommunikation mit dem Prozessor Vorrang vor einer Kommunikation mit der Systemumgebung haben sollte. Etwa kann im Startzustand Bench1 eine Kommunikation auf Port dma1 anbieten, obwohl CPU mit Bench1 auf Kanal fetch1 kommunizieren kann. Im folgenden wollen wir zeigen, wie sich diese Unzulänglichkeit beheben läßt.

4 CCS mit Prioritäten

Prioritäten spielen in Hardwaresystemen sowie in Spezifikations- und Programmiersprachen eine wichtige Rolle, da sie die Modellierung von *Interruptmechanismen* und *prioritisierter Auswahl* erlauben. Prozeßalgebren mit Prioritäten [CLN99] ermöglichen im Vergleich zu klassischen Prozeßalgebren eine präzisere Modellierung realer Systeme und damit eine verläßlichere Systemverifikation. Um Prioritäten in CCS zu berücksichtigen, unterteilen wir die

Abbildung 3: Semantik in CCS ohne bzw. mit Prioritäten

Menge Λ in „unwichtige“ und „wichtige“ Ports bzw. Aktionen. Dieses zweistufige *Prioritätenschema* ist für unser Beispiel ausreichend. Wir wollen die Ports $\underline{\mathsf{fetch}i}$ als wichtig und die übrigen Ports als unwichtig betrachten. Wichtige Ports werden hierbei durch Unterstreichen gekennzeichnet.

4.1 Semantik mit Globaler Präemption

Prioritäten in CCS wurden zuerst von R. Cleaveland und M. Hennessy eingeführt [CH90]. Ihre Semantik ist durch die folgende Designentscheidung geprägt. Wann immer ein Prozeß zwischen einer Kommunikation auf einem wichtigen und einer Kommunikation auf einem unwichtigen Port auswählen kann, entscheidet er sich für die erstere. Dabei werden Handshakekommunikationen auf wichtigen Ports durch die wichtige interne Aktion $\underline{\tau}$ und alle anderen Kommunikationen durch τ gekennzeichnet. Technisch gesehen bedeutet die obige Designentscheidung, daß von einem Prozeß, der sich in $\underline{\tau}$ engagieren kann, keine mit einer unwichtigen Aktion beschriftete Transition ausgehen darf; d.h. unwichtige Transitionen werden unabhängig davon abgeschnitten, von welcher Teilkomponente des betrachteten Systems sie ausgehen. Dieses Prinzip wird auch als *globale Präemption* bezeichnet.

Falls wir diese Semantik mit globaler Präemption auf unser Beispiel anwenden, stellen wir fest, daß das resultierende Transitionssystem nur noch aus den zyklischen wichtigen Kommunikationen zwischen dem Prozessor und den beiden Speicherbänken besteht. Alle mit dma1 und dma2 beschrifteten Schleifen wurden aufgrund globaler Präemption abgeschnitten (siehe Abbildung 3, rechts). Leider ist dies nicht die intuitiv erwünschte Semantik. Eigentlich soll gelten, daß im Startzustand Transition dma2 erhalten bleibt, da CPU nur mit Bench1 kommunizieren kann und nicht mit Bench2. Während Bench1 auf eine Kommunikation auf Port $\underline{\mathsf{fetch1}}$ eingehen kann und daher eine potentielle Kommunikation auf Port dma1 zurückstellen muß, sollte Bench2 eine Kommunikation auf Port dma2 anbieten dürfen. Für den zweiten Zustand des Gesamtsystems

gilt Analoges. Das Konzept der globalen Präemption aus der klassischen Literatur ist demnach für verteilte Systeme ungeeignet. Um die Verteilung eines Systems zu beachten, müssen bei der Präemption *Lokalitäten* von Transitionen berücksichtigt werden. Dies führt auf das Konzept der *lokalen Präemption*.

4.2 Semantik mit Lokaler Präemption

Wir führen zunächst einen geeigneten Lokalitätsbegriff ein. Technisch gesehen ist die Lokalität einer Transition ein Zeiger auf den Teilterm des betrachteten Prozesses, von dem die Transition ausgeht. Wir definieren dann die Relation $\bowtie$ auf Lokalitäten, welche genau solche Lokalitäten bzw. Transitionen miteinander in Verbindung setzt, die logisch auf demselben „Prozessor“ ausgeführt werden. Lokalitäten auf verschiedenen Seiten des nichtdeterministischen Auswahloperators $+$ sind vergleichbar, und solche auf verschiedenen Seiten des Paralleloperators $|$ sind unvergleichbar. Wir nutzen $\bowtie$ aus, um ein lokales Präemptionskonzept, wie in Abbildung 4, links, angedeutet, zu definieren. In dieser Skizze sind nur die Lokalitäten o und p und die Lokalitäten m und n vergleichbar. Folglich darf die wichtige interne $\underline{\tau}$-Transition nur die unwichtige δ-Transition verhindern, nicht aber die unwichtige α-Transition wie bei der globalen Präemption, da m und p unvergleichbar sind. Das lokale Präemptionskonzept führt auf eine neue Semantik für CCS, die für unser Beispiel in Abbildung 4, rechts, dargestellt ist. Im Startzustand wird die dma1-Transition und im anderen Zustand die dma2-Transition abgeschnitten. Diese Semantik reflektiert unsere Intuition hinsichtlich verteilter Systeme besser.

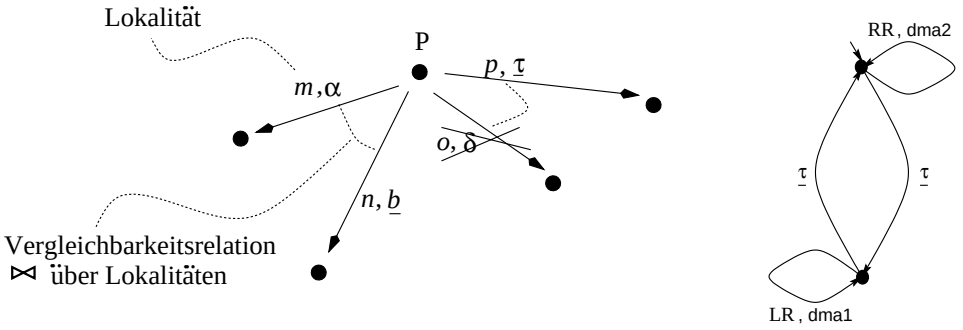


Abbildung 4: Lokale Präemption und Semantik des Beispielsystems

Zum Abschluß betrachten wir die Konsequenzen lokaler Präemption für die Bisimulation. Falls wir die in Abschnitt 3.1 vorgestellte Bisimulation $\sim$ in naiver Weise adaptieren, so ist die gewonnene Äquivalenz $\simeq$, welche Prioritäten ignoriert, keine Kongruenz mehr. Als Beispiel betrachte man die Prozesse $a.\underline{b}.\mathbf{0} + \underline{b}.a.\mathbf{0}$ und $a.\mathbf{0} \,|\, \underline{b}.\mathbf{0}$, die nach dem Interleavingprinzip für parallele

Komposition bezüglich $\simeq$ äquivalent sind. Falls wir nun diese Prozesse mit $\overline{\underline{b}}.\mathbf{0}$ parallel komponieren, so gilt $(a.\underline{b}.\mathbf{0} + \underline{b}.a.\mathbf{0}) \,|\, \overline{\underline{b}}.\mathbf{0} \not\simeq a.\mathbf{0} \,|\, \underline{b}.\mathbf{0} \,|\, \overline{\underline{b}}.\mathbf{0}$. Im linken Prozeß wird aufgrund lokaler Präemption die a-Transition abgeschnitten, da die wichtigere Kommunikation auf Port $\underline{b}$ in einer vergleichbaren Lokalität stattfindet, während diese Kommunikation im rechten Prozeß in unvergleichbaren Lokalitäten ausgeführt wird. Dieses Beispiel zeigt auch, daß das Interleavingprinzip bei Anwesenheit lokaler Präemption nicht allgemeingültig ist. Zur Definition einer kompositionellen Bisimulation muß lokale Präemption berücksichtigt werden. Dies führt auf die Äquivalenz $\simeq^{\mathrm{l}}$, die sich nur im Falle unwichtiger Aktionen von $\simeq$ unterscheidet. Für $\simeq^{\mathrm{l}}$ gilt: $P \simeq^{\mathrm{l}} Q \;\wedge\; P \xrightarrow{m,\alpha} P'$ impliziert $Q \xrightarrow{n,\alpha} Q' \;\wedge\; P' \simeq^{\mathrm{l}} Q'$ für einen Prozeß Q' und eine Lokalität n mit $\underline{\mathit{II}}_{[n]}(Q) \subseteq \underline{\mathit{II}}_{[m]}(P)$. Hierbei bezeichnet $\underline{\mathit{II}}_{[m]}(P)$ die Menge aller wichtigen, beobachtbaren Aktionen, die als Label einer initialen Transition von P in einer zu m vergleichbaren Lokalität auftreten. Als zentrales Resultat erhalten wir, daß $\simeq^{\mathrm{l}}$ die *größte in* $\simeq$ *enthaltene Kongruenz* ist. Diese Kongruenz läßt sich auch axiomatisch und mit Hilfe einer Modallogik charakterisieren [CLN98].

5 Zusammenfassung

Das Konzept der Präemption ist zur adäquaten Modellierung nebenläufiger und verteilter Systeme unerläßlich. Es findet sich in Prioritäten- und Realzeitkonzepten vieler Spezifikations- und Programmiersprachen wieder. Ein adäquater formaler Rahmen zur Untersuchung des Präemptionskonzeptes sind Prozeßalgebren. Sie bilden nicht selten das Fundament für praxisrelevante Spezifikationssprachen, z.B. für die in UML enthaltenen *Statecharts* [Har87].

Leider konnte hier aus Platzgründen nur ein kurzer Einblick in die Welt der Prozeßalgebren und das Konzept der Präemption gegeben werden. Der Leser sei auf die diesem Artikel zugrunde liegende Dissertation verwiesen [Lüt98]. Dort ist auch eine Anwendung des lokalen Präemptionskonzeptes auf *temporale Prozeßalgebren* zu finden [CLM97] und ebenso eine Prozeßalgebra mit *dynamischen Prioritäten* zur kompakten Kodierung von Realzeit [BCL99]. Das Präemptionskonzept wird in [Lüt98] anhand der Verhaltensäquivalenzen *starke Bisimulation* und *Beobachtungsäquivalenz* untersucht. Wesentliche technische Resultate liegen dabei im Nachweis von Kongruenzeigenschaften und in *axiomatischen*, *logischen*, und *algorithmischen Charakterisierungen* von Kongruenzen. Neben theoretischen Betrachtungen zu Präemptionskonzepten enthält die Dissertation auch zwei Fallstudien: die Modellierung eines *Eisenbahnsignalsystems* [CNSL96] und des *SCSI-2 Busprotokolls* [BCL99].

Literatur

[BCL99] G. Bhat, R. Cleaveland und G. Lüttgen. A practical approach to implementing real-time semantics. *Annals of Software Engineering*, 7, 1999.

[CH90] R. Cleaveland und M. Hennessy. Priorities in process algebra. *Information and Computation*, 87(1/2):58–77, 1990.

[CLM97] R. Cleaveland, G. Lüttgen und M. Mendler. An algebraic theory of multiple clocks. In *CONCUR '97*, Band 1243 in *LNCS*, Seiten 166–180, 1997. Springer Verlag.

[CLN98] R. Cleaveland, G. Lüttgen und V. Natarajan. A process algebra with distributed priorities. *Theoretical Computer Science*, 195(2):227–258, 1998.

[CLN99] R. Cleaveland, G. Lüttgen und V. Natarajan. Priority in process algebra. In *Handbook of Process Algebra*. Elsevier, 1999.

[CNSL96] R. Cleaveland, V. Natarajan, S. Sims und G. Lüttgen. Modeling and verifying distributed systems using priorities: A case study. *Software—Concepts and Tools*, 17(2):50–62, 1996.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Lüt98] G. Lüttgen. *Pre-emptive Modeling of Concurrent and Distributed Systems*. Dissertation, Universität Passau, 1998. Shaker Verlag.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

Gerald Lüttgen studierte von 1989 bis 1994 Informatik an der Rheinisch-Westfälischen Technischen Hochschule Aachen und promovierte im Mai 1998 bei Prof. Bernhard Steffen an der Universität Passau. Seine Dissertation wurde ebenfalls von Prof. Rance Cleaveland von der North Carolina State University betreut, an der G. Lüttgen 1995 einen einjährigen Gastaufenthalt – gefördert durch ein Doktorandenstipendium des DAAD – verbrachte. Seit Juli 1998 ist G. Lüttgen wissenschaftlicher Mitarbeiter des *Institute for Computer Applications in Science and Engineering* am NASA Langley Research Center, an dem er Spezifikations- und Verifikationstechniken für nebenläufige und verteilte Systeme entwickelt und diese im Luft- und Raumfahrtbereich anwendet.