Seminar: Kommunikation und Koordination in verteilten intelligenten Softwaresystemen – Formale Modellierung, Spezifikation und Verifikation

Prof. Wilfried Brauer

Michael Rovatsos
Matthias Nickles
Gerhard Weiß

Das π-Kalkül

Stefan Richter

31.01.2002

Zusammenfassung. In dieser Arbeit wird das π-Kalkül als eine klassisch formale Betrachtungsweise von kommunizierenden und mobilen Prozessen eingeführt. Nach einer kurzen begrifflichen und geschichtlichen Einordnung solcher Beschreibungsformen definieren wir die wesentlichen formalen Mittel des π-Kalküls und zeigen einige Eigenschaften auf. Abschließend erfolgen Hinweise auf Erweiterungen und Anwendungen des Kalküls.

Inhaltsverzeichnis

1 Warum gibt es das π-Kalkül?

In den letzten Jahrzehnten beschäftigt sich die Informatik zunehmend mit kommunizierenden Systemen, nachdem sich bis jetzt das Verständnis sequentieller Programme wesentlich tiefer entwickelt hat. Dies ist vor allem in der Entwicklung der Informatikpraxis zu begründen, in der diese Systeme eine zunehmende Rolle spielen. Bis heute ist hingegen der theoretische Hintergrund nur unzureichend erforscht.

Nachdem man bereits viele Modelle für sequentielle Berechnung gefunden hatte, wie etwa Automaten oder Turingmaschinen, kamen etwa um 1980 zwei wichtige Beschreibungsweisen für die Kommunikation sequentieller Systeme auf: Tony Hoares CSP (Communicating Sequential Processes) [2] und Robin Milners CCS (Calculus of Communicating Systems) [3]. Beide Modelle sind sich recht ähnlich und CCP ist der direkte Vorgänger des π-Kalküls.

Mit Milners π-Kalkül kann nun zusätzlich eine Komponente von Systemen beschrieben werden, die man Mobilität nennen könnte: Die Dynamisierung der Verbindungen zwischen den einzelnen Prozessen. Das bedeutet, daß Prozesse dynamisch Kanäle zu anderen Prozessen aufbauen und abbauen können. Dieses Konzept ist sehr mächtig, und es führt hier zu weit, seine Tragweite darzulegen. Ich verweise daher auf Robin Milners Buch *Communicating and mobile systems: the π-calculus*[1]. Diese Arbeit basiert im Wesentlichen auf [1]. Wenn nicht anders angegeben oder bewiesen, so findet sich der Beleg für eine Aussage stets hier.

Begrifflich sind sowohl das π-Kalkül als auch seine genannten Vorgänger als sogenannte Prozeßalgebren oder auch Prozeßkalküle einzuordnen (Für die folgenden Begriffsklärungen vergleiche auch [4]). Eine Prozeßalgebra ist eine Menge von Prozessen, die bezüglich gewisser Operationen abgeschlossen ist. Man kann dabei etwa die rein syntaktischen Terme der Prozeßausdrücke als Prozesse betrachten, und als Operationen zunächst nur Regeln auffassen, nach denen Prozeßausdrücke gebildet werden. Auf einer höheren Ebene werden Kongruenzen hinzugefügt, die Terme unifizieren, die sich in einer gewissen Weise gleich verhalten. Diese Kongruenzen kann man zunächst einfach als Operationen dem Kalkül hinzufügen. Andererseits ist es nun möglich, mit ihrer Hilfe von den Termen zu abstrahieren, und als Prozesse die jeweiligen Kongruenzklassen zu betrachten. Auf diesen Klassen oder auch auf den Termen selbst lassen sich nun übergangsrelationen definieren, die das Verhalten der Prozesse festlegen, und selbst wieder als die Operationen einer Algebra aufgefaßt werden können.

Der Begriff des Kalküls wird selten im Zusammenhang mit Prozessen gebraucht. Er stammt aus der Logik und weist lediglich darauf hin, daß sich eine Menge induktiv aus endlich vielen Regeln aufbauen läßt. Im Beispiel des π-Kalküls sind dies etwa die Prozeßausdrücke, die Kongruenzrelationen und auch die Transitionsrelationen. Solche Prozeßalgebren sind also in vielerlei Hinsicht Kalküle.

2 Das π-Kalkül

In diesem Kapitel soll zunächst in 2.1 das π-Kalkül syntaktisch formal definiert und die Konstruktionen informell erläutert werden. Dabei werden wir eine Äquivalenzrelation entwickeln, die es uns erlaubt, syntaktisch unterschiedliche Terme zu unifizieren, soweit sie den gleichen Prozeß beschreiben.

Um diese Gleichwertigkeit auch formal zu begründen, sollen im Weiteren (2.2) zwei Zustandsübergangsrelationen eingeführt werden, die uns eine operationale Semantik an die Hand geben. Wir werden jeweils zeigen, daß diese Transitionen bezüglich der bereits definierten Kongruenz invariant sind.

Im dritten Teil (2.3) finden wir zwei weitere Relationen auf π-Kalkül-Termen – beziehungsweise mit Hilfe der Kongruenzklassen auf Prozessen – die eine strenge Verhaltensgleichheit und eine schwächere Beobachtungsäquivalenz darstellen.

2.1 Definition und strukturelle Kongruenz

Die Essenz des π-Kalküls ist die Kommunikation. Daher werden mit π die sogenannten Aktionspräfixe bezeichnet. Ein Aktionspräfix stellt das Senden oder Empfangen einer Botschaft (genauer: eines Namens) oder einen stillen Übergang dar. Die Syntax ist:

Definition 1 (Aktionspräfix)

$$\begin{array}{lll} \pi ::= & x(y) & \textit{empfange } y \textit{ entlang } x \\ & \overline{x}\langle y\rangle & \textit{sende } y \textit{ entlang } x \\ & \tau & \textit{nichtbeobachtbare Aktion} \end{array}$$

Mit Hilfe dieser Präfixe definieren wir nun die eigentlichen Prozeßausdrücke:

Definition 2 (Das π-Kalkül) *Die Menge $\mathcal{P}^\pi$ der π-Kalkül* Prozeßausdrücke *ist durch die folgende Syntax definiert:*

$$\begin{array}{lll} P ::= & \sum_{i\in I}\pi_i.P_i & \textit{Summation} \\ & P_1 \mid P_2 & \textit{parallele Komposition} \\ & \text{new}\, a\ P & \textit{Restriktion} \\ & !P & \textit{Replikation} \end{array}$$

wo I eine endliche Indexmenge ist.

Diese Definition wird noch einmal etwas verallgemeinert werden, wenn wir Abstraktionen und Konkretisierungen einführen.

Die Prozeßausdrücke stehen für Prozesse. So steht etwa die Summation $x(a).P + z\langle b\rangle.Q$ für den Prozeß, der entweder den Parameter a entlang des Kanals x empfangen und

dann in den Prozeß P übergehen, oder den Namen b entlang des Kanals z senden und dann in den Prozeß Q wechseln kann. $R|S$ bezeichnet einfach die Nebeneinanderstellung der Prozesse R und S. Die Restriktion new a P bindet den Namen a in ihrem Geltungsbereich, hier in P, so daß er nur innerhalb P sichtbar ist. Die Replikation $!P$ schließlich steht für beliebig viele Instanzen von P, die parallel komponiert werden.

2.1.1 Das polyadische π-Kalkül

Das so eingeführte Kalkül ist monadisch in dem Sinne, daß wir in jedem Aktionspräfix π nur einen Namen versenden können. Oft ist es jedoch praktischer, Ausdrücke der Form $x(y_1 \cdots y_n).P$ oder $\overline{x}\langle z_1 \cdots z_n\rangle.Q$ zu verwenden. Diese können im monadischen π-Kalkül dargestellt werden, jedoch nicht in der offensichtlichen Form $x(y_1).\cdots.x(y_n).P$ bzw. $\overline{x}\langle z_1\rangle.\cdots.\langle z_n\rangle.Q$.

Man betrachte zum Beispiel

$$x(y_1y_2).P \mid \overline{x}\langle z_1z_2\rangle.Q \mid \overline{x}\langle z_1'z_2'\rangle.Q'.$$

Die offensichtliche Kodierung ergibt

$$x(y_1).x(y_2).P \mid \overline{x}\langle z_1\rangle.\overline{x}\langle z_2\rangle.Q \mid \overline{x}\langle z_1'\rangle.\overline{x}\langle z_2'\rangle.Q'$$

und dies ermöglicht Vermischungen: so kann y_1y_2 durch $z_1'z_1$ ersetzt werden, was wir in der Originalschreibweise nicht intendierten.

In einer korrekten Kodierung müssen wir daher sicherstellen, daß es auf dem Kanal, auf dem eine zusammengesetzte Botschaft gesandt wird, keine Interferenzen gibt. Also wird zunächst ein *neuer* Name geschickt, der dann als Kanal für die eigentliche Nachricht dient. Wir übersetzen demnach die polyadischen Aktionspräfixe wie folgt:

$$\begin{array}{lcl} x(y_1 \cdots y_n).P & \longmapsto & x(w).w(y_1).\cdots.w(y_n).P \\ \overline{x}\langle z_1 \cdots z_n\rangle.Q & \longmapsto & \text{new}\, w\ (\overline{x}\langle w\rangle.\overline{w}\langle z_1\rangle.\cdots\overline{w}\langle z_n\rangle.Q) \end{array}$$

Diese Kodierung zeigt, daß das monadische Kalkül als Grundlage ausreicht, da es das Polyadische ausdrücken kann. Wir werden von nun an daher das polyadische π-Kalkül uneingeschränkt benutzen, wo es angebracht scheint. Es erlaubt die Mehrfachaktionspräfixe $x(\vec{y})$ und $\overline{x}\langle\vec{z}\rangle$. Man beachte, daß es insbesondere die Fälle $x()$ und $\overline{x}\langle\rangle$ erlaubt, in denen $\vec{y}$ bzw. $\vec{z}$ leer sind; wir schreiben diese inhaltslosen Aktionen einfach als x und $\overline{x}$.

2.1.2 Kongruenzen

Um nun von Prozeßausdrücken abstrahieren zu können, und damit über Prozesse als solche zu sprechen, benötigen wir die strukturelle Kongruenz, die solche Ausdrücke als äquivalent ausweist, die den gleichen Prozeß bezeichnen. Dazu ist jedoch das Konzept der Prozeßkongruenz nötig, das wiederum vom der Idee des Prozeßkontexts abhängt:

Definition 3 (Prozeßkontext) Prozeßkontexte $\mathcal{C}$ *sind durch die folgende Syntax gegeben:*

$$\mathcal{C} ::= [\,] \quad | \quad \pi.\mathcal{C} + M \quad | \quad \text{new}\, a\ \mathcal{C} \quad | \quad \mathcal{C} \mid P \quad | \quad P \mid \mathcal{C} \quad | \quad !\mathcal{C}$$

wo M eine Summe und P ein Prozeß ist. $\mathcal{C}[Q]$ bezeichnet das Resultat der Ersetzung der Lücke $[\,]$ im Kontext $\mathcal{C}$ durch den Prozeßausdruck Q. Die elementaren Kontexte *sind* $\pi.[\,] + M$, $\text{new}\, a\ [\,]$, $[\,] \mid P$, $P \mid [\,]$ *und* $![\,]$.

Definition 4 (Prozeßkongruenz) *Sei $\cong$ eine Äquivalenzrelation über $\mathcal{P}^\pi$. Dann ist $\cong$ eine* Prozeßkongruenz*, falls es durch alle elementaren Kontexte bewahrt wird; das heißt, wenn $P \cong Q$, dann auch*

$$\begin{array}{rcl} \pi.P + M & \cong & \pi.Q + M \\ \text{new}\, x\ P & \cong & \text{new}\, x\ Q \\ P \mid R & \cong & Q \mid R \\ R \mid P & \cong & R \mid Q \\ !P & \cong & !Q. \end{array}$$

Der folgende Satz folgt leicht mittels struktureller Induktion:

Proposition 1 *Eine Äquivalenzrelation $\cong$ ist eine Prozeßkongruenz genau dann, wenn für alle Kontexte $\mathcal{C}$ aus $P \cong Q$ folgt $\mathcal{C}[P] \cong \mathcal{C}[Q]$.*

Wir sind jetzt bereit für strukturelle Kongruenz:

Definition 5 (Strukturelle Kongruenz) Strukturelle Kongruenz $\equiv$ *ist die kleinste Prozeßkongruenz, die folgende Äquivalenzen beinhaltet:*

1. *Änderung gebundener Namen (α-Konversion)*
2. *Kommutativität und Assoziativität der Summation und Komposition*
3. $P \mid 0 \equiv P$, $\text{new}\, x\ 0 \equiv 0$ *wobei 0 die leere Summe ist*
4. $\text{new}\, x\ (P \mid Q) \equiv P \mid \text{new}\, x\ Q$ *falls x nicht in Q vorkommt*
   $\text{new}\, xy\ P \equiv \text{new}\, yx\ P$
5. $!P \equiv P \mid !P$.

2.1.3 Abstraktionen und Konkretisierungen

Bis jetzt haben wir stets Prozesse betrachtet. Für die Verpflichtungen im nächsten Abschnitt müssen wir jedoch noch eine Verallgemeinerung dieses Begriffs zulassen: Agenten.

Agenten sind entweder Abstraktionen oder Konkretisierungen. Sie sind nützlich, um das Konzept der Bindung beziehungsweise Substitution von Namen zu fassen. Damit sind sie dem λ-Kalkül ähnlich. Wir schreiben A,B,... für beliebige Agenten.

Eine n-stellige Abstraktion ist ein parametrischer Prozeß in n Variablen, also etwa $(x_1 \cdots x_n).P$. Eine Abstraktion $(x).P$ repräsentiert also alle möglichen Pfade, die P nehmen kann, abhängig von der Botschaft z, die für x in P substituiert wird. Wir benennen Abstraktionen mit den Buchstaben F,G, ... und schreiben $F : n$, um die Stelligkeit des Agenten zu bezeichnen.

Eine Konkretisierung ist in gewisser Weise dual zu einer Abstraktion, weil sie Namen für deren Variablen bereitstellt, die darin ersetzt werden können. Eine solche Konkretisierung ist etwa $\text{new}\,\vec{x}\ \langle\vec{y}\rangle.Q$, wobei die Namen aus $\vec{x}$ eine Teilmenge der Namen in $\vec{y}$ sind (Abkürzung: $\vec{x} \subseteq \vec{y}$). Hier zeigt sich, daß Konkretisierungen jedoch im Gegensatz zu Abstraktionen eine Besonderheit haben: Es ist möglich, den Bindungsbereich von Namen auf die Botschaft $\vec{y}$ und den Fortsetzungsprozeß Q zu beschränken. Konkretisierungen werden im Folgenden mit C,D,... bezeichnet.

Definition 6 (Agenten) *Sei $n \geq 0$.*

Eine n-stellige Abstraktion *hat die Form $(\vec{x}).P$, wobei $|\vec{x}| = n$. Zwei Abstraktionen sind strukturell kongruent ($\equiv$), wenn ihre gebundenen Namen $(\vec{x})$ bis auf α-Konversion übereinstimmen und ihre Prozeßteile P strukturell kongruent sind.*

Eine n-stellige Konkretisierung *hat die Form* $\text{new}\,\vec{x}\ \langle\vec{y}\rangle.P$, *wobei $|\vec{y}| = n$ und $\vec{x} \subseteq \vec{y}$. Zwei Konkretisierungen sind strukturell kongruent, wenn ihre Präfixe* $\text{new}\,\vec{x}\ \langle\vec{y}\rangle$ *bis auf α-Konversion und Umordnung der restringierten Namen übereinstimmen und ihre Prozeßteile P strukturell kongruent sind.*

Ein Agent *ist entweder Abstraktion oder eine Konkretisierung. $\mathcal{A}^\pi$ bezeichne die Menge der Agenten.*

Man beachte, daß Prozesse Agenten sind; Ein Prozeß ist sowohl eine Abstraktion wie eine Konkretisierung der Stelligkeit 0.

Wir müssen nun die Summationsform $\sum \pi_i.P_i$ aus Definition 1 leicht verallgemeinern. Im polyadischen Kalkül erlauben wir bereits einen Summanden wie $\overline{x}\langle y_1 y_2\rangle.P$. Dies ist ein Spezialfall von $\overline{x}C$, wo C für eine Konkretisierung steht. Es erscheint daher passend, solche Summanden $\overline{x}C$ mit *beliebigen* Konkretisierungen C zuzulassen, sogar mit einer Restriktion; ein Beispiel ist $\overline{x}(\text{new}\,y_1\ \langle y_1 y_2\rangle.P)$. Dann haben Summen von nun an die Form $\sum \alpha_i A_i$, wobei jeder Summand von einer der Formen xF, $\overline{x}C$ oder τP ist.

Wie oben angedeutet, ist es natürlich möglich, eine Konkretisierung auf eine Abstraktion anzuwenden. Dazu definieren wir einfach die folgende Abkürzung:

Definition 7 (Applikation) *Die* Applikation $F@C$ *einer gleichstelligen Abstraktion und Konkretisierung ist wie folgt definiert, vorausgesetzt* $\vec{z}$ *nicht frei in* $(\vec{x}).P$*:*

$$(\vec{x}).P \,@\, \text{new}\, \vec{z}\ \langle\vec{y}\rangle.Q \longmapsto \text{new}\, \vec{z}\ (\{\vec{y}/\vec{x}\}P \mid Q)$$

Um die Verpflichtungsregeln im nächsten Teil definieren zu können, müssen Restriktion und Komposition auf alle Agenten erweitert werden. Bei der Bildung von $\text{new}\, z\ A$ oder $A|Q$, die jeweils wieder Abstraktionen beziehungsweise Konkretisierungen der selben Stelligkeit wie A sein werden, wird im Wesentlichen nur die Konkretisierungs- bzw. Abstraktionseigenschaft nach außen geschoben, oder genauer:

Definition 8 *Sei* $z \notin \vec{x}$ *und* $\vec{x}$ *nicht frei in* Q*. Die Operationen* $\text{new}\, z\ A$ *und* $A|Q$ *(für beliebige Agenten* A *und Prozesse* Q*) sind wie folgt definiert, wobei* α*-Konversion zu benutzen ist, wo es nötig ist, um Namenskonflikte zu vermeiden. Für eine Abstraktion* $A = (\vec{x}).P$*:*

$$\begin{aligned} \text{new}\, z\ ((\vec{x}).P) &\longmapsto (\vec{x}).\text{new}\, z\ P \\ ((\vec{x}).P) \mid Q &\longmapsto (\vec{x}).(P \mid Q). \end{aligned}$$

Für eine Konkretisierung $A = \text{new}\, \vec{x}\ \langle\vec{y}\rangle.P$*:*

$$\begin{aligned} \text{new}\, z\ (\text{new}\, \vec{x}\ \langle\vec{y}\rangle.P) &\longmapsto \begin{cases} \text{new}\, z\vec{x}\ \langle\vec{y}\rangle.P & z \in \vec{y} \\ \text{new}\, \vec{x}\ \langle\vec{y}\rangle.\text{new}\, z\ P & \text{sonst} \end{cases} \\ (\text{new}\, \vec{x}\ \langle\vec{y}\rangle.P) \mid Q &\longmapsto \text{new}\, \vec{x}\ \langle\vec{y}\rangle.(P \mid Q). \end{aligned}$$

2.2 Reaktion und Verpflichtung

Nachdem nun das π-Kalkül für unsere Bedürfnisse syntaktisch vollständig definiert ist, wird die bisher nur informelle Semantik nun durch zwei Arten der operationalen Semantik zunächst ergänzt – das Konzept der Reaktion – und später ersetzt – das mächtigere Konzept der Verpflichtung.

2.2.1 Reaktion

Zum Begriff Reaktion wurde Robin Milner angeregt durch den entsprechenden Begriff in der Chemie, beziehungsweise durch die vorausgehende Arbeit *The chemical abstract machine* [5]. Er meint hier einen Zustands- oder genauer Prozeßübergang, der ohne Beeinflussung von außen, das heißt Einwirkung eines anderen Prozesses im Sinne von Senden oder Empfangen von Nachrichten, erfolgt. Man spricht bei einer solchen Einwirkung auch von Beobachtung, und in diesem Sinne erklärt Reaktion nur nicht beobachtbare Transitionen.

Solche Übergänge können im Wesentlichen auf zwei Arten geschehen, und diese beiden Möglichkeiten werden durch die beiden ersten Reaktionsregeln (vgl. Definition 9)

angegeben: Entweder findet die interne τ-Aktion statt, die spontan zu dem von ihr bewachten Prozeß übergeht (Für die Form $\pi.P$ sagen wir, der Prozeß P wird von dem Aktionspräfix π *bewacht*), oder eine Aktion und ihr Komplement (Für die Aktion α ist $\overline{\alpha}$ das Komplement und umgekehrt. Die Aktion τ besitzt und benötigt kein Komplement zur Reaktion.) in zwei parallel komponierten Prozessen reagieren miteinander.

Natürlich können die genannten Konstrukte, um zu einer Reaktion zu führen, nicht an beliebigen Stellen in einem Prozeßausdruck vorkommen. So kann in $\overline{a}.(b.Q + \overline{b}.P)$ die Interaktion zwischen b und $\overline{b}$ nicht stattfinden, ehe die Aktion $\overline{a}$ beobachtet wurde. Die beiden nächsten Regeln PAR und RES besagen, daß Reaktion innerhalb von Komposition und Restriktion passieren kann. Die letzte Regel ermöglicht die Verwendung struktureller Kongruenz bei der Herleitung von Reaktionen.

Definition 9 (Reaktion) *Die Reaktionsrelation $\rightarrow$ über $\mathcal{P}^\pi$ enthält genau die Übergänge, die mit den folgenden Regeln hergeleitet werden können:*

$$\text{TAU}: \quad \tau.P + M \rightarrow P$$

$$\text{REACT}: \quad (xF + M) \mid (\overline{x}C + N) \rightarrow F@C$$

$$\text{PAR}: \ \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q} \qquad \text{RES}: \ \frac{P \rightarrow P'}{\text{new}\, x\ P \rightarrow \text{new}\, x\ P'}$$

$$\text{STRUCT}: \ \frac{P \rightarrow P'}{Q \rightarrow Q'} \text{ falls } P \equiv Q \text{ und } P' \equiv Q'$$

Man beachte, daß für die Replikation keine besondere Regel nötig ist. Das gewünschte Verhalten wird allein mit Hilfe der strukturellen Kongruenz erreicht.

Die letzte Regel erübrigt auch den Beweis, daß die Reaktion bezüglich struktureller Kongruenz invariant ist.

2.2.2 Verpflichtungen

Die Reaktionsregeln erklären zwar internes Verhalten, erlauben jedoch keine Schlüsse über Beobachtungen, das heißt Kommunikation zwischen verschiedenen Prozessen. Um diese zu beschreiben, definieren wir Relationen $\xrightarrow{\alpha}$. In dem π-Vorgänger CCS [3] heißen diese Relationen einfach Übergänge (transitions). In π [1] bezeichnet Milner sie nun als Verpflichtungen (commitments), da man ein Relationselement $P \xrightarrow{\alpha} P'$ als *Verpflichtung* des Prozesses P interpretieren kann, an einer Reaktion mit der Aktion α teilzunehmen.

Definition 10 (Verpflichtung) *Die Verpflichtungen eines Prozesses sind genau diejenigen, die aus den folgenden Regeln zusammen mit α-Konversion abzuleiten sind:*

$$\mathrm{SUM_C}: \quad M + \alpha A + N \xrightarrow{\alpha} A$$

$$\mathrm{L-REACT_C}: \frac{P \xrightarrow{x} F \quad Q \xrightarrow{\overline{x}} C}{P \mid Q \xrightarrow{\tau} F@C} \qquad \mathrm{R-REACT_C}: \frac{P \xrightarrow{\overline{x}} C \quad Q \xrightarrow{x} F}{P \mid Q \xrightarrow{\tau} F@C}$$

$$\mathrm{L-PAR_C}: \frac{P \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \mid Q} \qquad \mathrm{R-PAR_C}: \frac{Q \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \mid P}$$

$$\mathrm{RES_C}: \frac{P \xrightarrow{\alpha} A}{\mathrm{new}\, x\ P \xrightarrow{\alpha} \mathrm{new}\, x\ A} \text{ falls } \alpha \notin x, \overline{x}$$

$$\mathrm{REP_C}: \frac{P \mid !P \xrightarrow{\alpha} A}{!P \xrightarrow{\alpha} A} \text{ falls } \alpha \notin x, \overline{x}$$

Setzt man für α τ, so erkennt man, daß in diesem Fall die Verpflichtungsregeln im Wesentlichen nur eine Umformulierung der Reaktionsregeln darstellen. Die Verdoppelung der REACT- und PAR- Regel, sowie die Einführung von REP ist notwendig, weil hier auf eine explizite Regel zur Einführung der strukturellen Kongruenz verzichtet wird. Diese Äquivalenz (bis auf strukturelle Kongruenz) von Reaktion und τ-Verpflichtung kann leicht gezeigt werden, wir verzichten darauf allerdings ebenso wie Milner [1].

Wie jedoch oben angekündigt, zeigen wir eine gewisse Invarianz der Verpflichtungen bezüglich struktureller Äquivalenz. Im Gegensatz zum Reaktionsfall, wo dies mit der STRUCT-Regel trivial war, bleibt uns hier zu beweisen:

Theorem 1 (Strukturelle Kongruenz respektiert Verpflichtung) *Falls $P \xrightarrow{\alpha} A$ und $P \equiv Q$, dann gibt es einen Agenten B, so daß $Q \xrightarrow{\alpha} B$ und $A \equiv B$.*

Beweis. Wir beginnen mit Induktion über die Anzahl n der Anwendungen von Regeln der strukturellen Konsequenz (Definition 5) für die Herleitung von $P \equiv Q$.

Für null Anwendungen ist $P = Q$ und $P' = Q'$ und daher $P' \equiv Q'$, denn $\equiv$ ist eine Äquivalenzrelation und reflexiv.

Gelte also die Behauptung für Inferenzen der Länge n. Dann ist der letzte Schritt in einer Herleitung der Länge $n + 1$ eine Anwendung genau einer Regel aus Definition 5. Wir führen nun Induktion über die Tiefe der Ableitung von $P \xrightarrow{\alpha} P'$ durch. Es sind daher alle Verpflichtungsregeln (Definition 10) als möglicher letzter Schritt der Ableitung zu betrachten.

Ab diesem Punkt entsteht eine Fallunterscheidung mit mehr als $10 * 7$ Regelkombinationen (Die Reaktionsregeln sind zwar in fünf Kategorien unterteilt, genauer betrachtet sind es jedoch etwa zehn, und außerdem können sie in verschiedenen Kontexten auftreten). Aus Platzgründen werden wir uns auf einige interessante Fälle beschränken:

Nehmen wir etwa an, daß die letzte angewandte Verpflichtungsregel $\mathrm{L}-\mathrm{PAR_C}$ war. Dann hat P die Form $P_1 \mid P_2$ und P' ist $P_1' \mid P_2$, und $P_1 \xrightarrow{\alpha} P_1'$ geht aus einer kürzeren Inferenz hervor. Jetzt gibt es viele Arten, auf die $P_1 \mid P_2 \equiv Q$ durch Anwendung genau einer Kongruenzregel gelten könnte. Wir beschränken uns hier genau wie Milner [1] auf zwei Fälle:

- Nehmen wir an, daß die Kommutativität genutzt wurde. Dann gilt $Q = P_2 \mid P_1$. Nun können wir $\mathrm{R}-\mathrm{PAR_C}$ benutzen, um $Q \xrightarrow{\alpha} P_1' \mid P_2 =: Q'$ herzuleiten. Dann gilt offensichtlich $P' \equiv Q'$ wie verlangt.

- Nehmen wir an, daß eine einzelne Kongruenzregel innerhalb von P_1 benutzt wird, so daß $P_1 \equiv Q_1$ und $Q = Q_1 \mid Q_2$. Weil $P_1 \xrightarrow{\alpha} P_1'$ eine kürzere Ableitung zugrunde liegt, haben wir als Induktionshypothese $Q_1 \xrightarrow{\alpha} Q_1'$ und $P_1' \equiv Q_1'$. Sei Q' nun $Q_1' \mid P_2$; mit Hilfe von $\mathrm{L}-\mathrm{PAR_C}$ erhalten wir $Q \xrightarrow{\alpha} Q'$ und $P' \equiv Q'$ wie gefordert.

...

■

2.3 Starke und schwache Bisimulation

Strukturelle Äquivalenz hilft uns, von Prozeßausdrücken zu abstrahieren, um über Prozesse sprechen zu können. Sie reicht aber nicht aus, um Verhaltensgleichheit darzustellen. Es kann durchaus vorkommen, daß zwei Prozesse nach außen hin nicht unterscheidbar sind, aber trotzdem nicht strukturell gleich sind. Daher benötigen wir die Konzepte der Bisimulation.

2.3.1 Starke Bisimulation

Informell ist eine starke Bisimulation eine Relation zwischen Prozessen, so daß sich diese Prozesse gleich verhalten. Durch die Verpflichtungen, die hier möglich sind, können natürlich auch Agenten auftreten. Daher müssen wir jede binäre Relation zwischen Prozessen auf Agenten wie folgt verallgemeinern:

Definition 11 *Sei $\mathcal{S}$ eine binäre Relation über $\mathcal{P}^\pi$. Dann werde $\mathcal{S}$ wie folgt auf $\mathcal{A}^\pi$ erweitert:*

$$F\mathcal{S}G \iff \forall \vec{y}.F\langle\vec{y}\rangle\mathcal{S}G\langle\vec{y}\rangle \qquad (\textit{wobei } F, G : n \textit{ und } |\vec{y}| = n)$$

$$C\mathcal{S}D \iff \exists \vec{z}, \vec{y}, P, Q.\ C \equiv \text{new } \vec{z}\ \langle\vec{y}\rangle.P \wedge D \equiv \text{new } \vec{z}\ \langle\vec{y}\rangle.Q \wedge P\mathcal{S}Q$$

Jetzt können wir Simulation und damit Bisimulation definieren:

Definition 12 (Starke (Bi-)Simulation) *Eine binäre Relation $\mathcal{S}$ über $\mathcal{P}^\pi$ ist eine* starke Simulation*, falls aus $P\mathcal{S}Q$ folgt:*

$$P \xrightarrow{\alpha} A \Longrightarrow \exists B.\ Q \xrightarrow{\alpha} B \wedge A\mathcal{S}B$$

Falls sowohl $\mathcal{S}$ als auch seine Inverse starke Simulationen sind, dann heißt $\mathcal{S}$ starke Bisimulation*. Zwei Agenten A und B sind* stark äquivalent*, geschrieben $A \sim B$, wenn das Paar (A, B) in einer starken Bisimulation ist.*

Man beachte, daß es für $(\vec{x}).P \sim (\vec{x}).Q$ nicht ausreicht, daß $P \sim Q$; es muß vielmehr gelten $\forall \vec{y}.\ \{\vec{y}/\vec{x}\}P\ \{\vec{y}/\vec{x}\}Q$!
Ein Beispiel: Falls $x \neq y$, dann gilt

$$\overline{x} \mid y \sim \overline{x}.y + y.\overline{x}$$

Beweis. Wir zeigen, daß

$$\mathcal{S} = \{(\overline{x} \mid y,\ \overline{x}.y + y.\overline{x}), (y, y), (\overline{x}, \overline{x}), (0, 0)\}$$

eine starke Bisimulation ist. Die einzige Schwierigkeit könnte beim ersten Paar auftreten. Die einzigen Verpflichtungen von $\overline{x} \mid y$ sind aber $\overline{x} \mid y \xrightarrow{y} \overline{x}$ und $\overline{x} \mid y \xrightarrow{\overline{x}} y$, und ihnen entsprechen auf der anderen Seite die einzigen beiden Verpflichtungen $\overline{x}.y + y.\overline{x} \xrightarrow{y} \overline{x}$ und $\overline{x}.y + y.\overline{x} \xrightarrow{\overline{x}} y$. ∎

Andererseits zerstören wir die Äquivalenz, wenn wir x für y substituieren, das heißt

$$\overline{x} \mid x \not\sim \overline{x}.x + x.\overline{x}$$

Beweis. $\overline{x} \mid x \xrightarrow{\tau} 0$ aber $\overline{x}.x + x.\overline{x}$ besitzt keine τ-Verpflichtung, denn es ist eine Summe, also wäre die einzige anwendbare Verpflichtungsregel $\mathrm{SUM_C}$, aber es kommt kein τ-Präfix vor. ∎

Daher wären diese ersten beiden Agenten nicht stark äquivalent, falls sie parametrisch in y wären, obwohl es für die entsprechenden Prozesse eine Bisimulation gibt.

2.3.2 Beobachtungsäquivalenz

Damit zwei Prozesse eine starke Bisimulation haben (genauer: Elemente einer starken Bisimulation sind), müssen sie sich genau gleich verhalten. Das heißt, es müssen alle Transitionen in beiden Systemen genau gleich erfolgen, insbesondere auch τ-Übergänge. Mit der nun einzuführenden schwachen (Bi-)Simulation sollen alle Prozesse gleichgesetzt werden, die sich in einem sogenannten *Experiment* gleich verhalten können. Dies bedeutet, daß unter einer Fairnessannahme, die sicherstellt, daß kein Prozeß unendlich lange in

einem Zustand „hängenbleibt“, wenn eine Reaktion möglich ist, die Beobachtung einer Sequenz von Aktionen bei einem Prozeß auch beim Anderen möglich ist.

Dazu brauchen wir zunächst das Konzept des (atomaren) Experiments. Hier zeigt sich eine weitere Asymmetrie zwischen positiven und negativen Aktionen.

Definition 13 (Atomares Experiment) *Die Übergangsrelation $\xrightarrow{x\langle\vec{y}\rangle}$ über $\mathcal{P}^\pi$ ist wie folgt definiert:*

$$P \xrightarrow{x\langle\vec{y}\rangle} \iff \exists F.\ P \xrightarrow{x} F \ \wedge\ F\langle\vec{y}\rangle \equiv P'$$

Das stille Experiment $P \Rightarrow Q$ steht für eine Folge aus einer oder mehrerer Reaktionen $P \to \ldots \to Q$. Formal gilt also $\Rightarrow \stackrel{def}{=} \to^*$.

Ein atomares Eingabeexperiment *ist ein Element der Relation $\overset{x\langle\vec{y}\rangle}{\Rightarrow}$, die wie folgt definiert ist:*

$$P \overset{x\langle\vec{y}\rangle}{\Rightarrow} P' \text{ genau dann, wenn } P \Rightarrow \xrightarrow{x\langle\vec{y}\rangle} \Rightarrow P'.$$

Ein atomares Ausgabeexperiment *ist ein Element der Relation $\overset{\overline{x}}{\Rightarrow}$, die wie folgt definiert ist:*

$$P \overset{\overline{x}}{\Rightarrow} \text{new } \vec{z}\ \langle\vec{y}\rangle.P' \iff \exists P''.\ P \Rightarrow \xrightarrow{\overline{x}} \text{new } \vec{z}\ \vec{y}.P'' \ \wedge\ P'' \Rightarrow P'.$$

Ein Experiment ist dann also eine Folge von atomaren (Eingabe- oder Ausgabe-) Experimenten. Für die Definition der schwachen Simulation benötigen wir aber nur atomare Experimente (und stille Übergänge):

Definition 14 (Schwache Simulation) *Eine binäre Relation $\mathcal{S}$ über $\mathcal{P}^\pi$ ist eine* schwache Simulation*, falls aus $P\mathcal{S}Q$ folgt*

$$\begin{array}{c} P \Rightarrow P' \implies \exists Q'.\ Q \Rightarrow Q' \wedge P'\mathcal{S}Q'; \\ P \overset{x\langle\vec{y}\rangle}{\Rightarrow} P' \implies \exists Q'.\ QP \overset{x\langle\vec{y}\rangle}{\Rightarrow} Q' \wedge P'\mathcal{S}Q'; \\ P \overset{\overline{x}}{\Rightarrow} C \implies \exists D.\ QP \overset{\overline{x}}{\Rightarrow} D \wedge C\mathcal{S}D. \end{array}$$

Wenn sowohl $\mathcal{S}$ als auch die konverse Relation schwache Simulationen sind, dann ist $\mathcal{S}$ eine schwache Bisimulation*. Zwei Agenten A und B sind* schwach äquivalent *oder* beobachtungsäquivalent*, geschrieben $A \approx B$, wenn das Paar (A, B) in einer schwachen Bisimulation ist.*

Genau wie bei starker Äquivalenz bedeutet $F \approx G$ von zwei Abstraktionen, daß $F\langle\vec{y}\rangle \approx G\langle\vec{y}\rangle$ für alle $\vec{y}$.

Man kann leicht zeigen, daß es für den Beweis von schwacher Simulation schon genügt, den Abschluß bezüglich *einfacher* Übergänge zu zeigen, also aus $P\mathcal{S}Q$ folgt

$$P \to P' \implies \exists Q'.\ Q \Rightarrow Q' \wedge P' \mathcal{S} Q';$$
$$P \xrightarrow{x\langle\vec{y}\rangle} P' \implies \exists Q'.\ QP \stackrel{x\langle\vec{y}\rangle}{\Rightarrow} Q' \wedge P' \mathcal{S} Q';$$
$$P \xrightarrow{\overline{x}} C \implies \exists D.\ QP \stackrel{\overline{x}}{\Rightarrow} D \wedge C \mathcal{S} D.$$

Wir verzichten auf den Beweis.

Damit ist das π-Kalkül im Wesentlichen formal eingeführt.

3 Ausblick

In diesem Teil sollen Erweiterungen und Anwendungen des π-Kalküls diskutiert werden, auf die wir jedoch aus naheliegenden Gründen weder vollständig noch im Detail eingehen können.

3.1 Erweiterungen

Bereits in [1] finden wir die erste Erweiterung des Kalküls, die hier lediglich aus Platz- und Zeitknappheit nicht im Detail behandelt wird: Typen beziehungsweise Sorten. Das π-Kalkül ist bis jetzt untypisiert, es ist jedoch recht leicht, Sorten und sogenannte *Sortierungen* einzuführen. Sortierungen sind Abbildungen, die den Sorten von Kanälen die Sorten der Daten zuordnen, die durch sie transportiert werden. Mit ihrer Hilfe kann man etwa zeigen, daß Objekte aus der objektorientierten Programmierung oder Funktionen aus der funktionalen Programmierung im π-Kalkül als Botschaften übertragen werden können, und damit insbesondere, daß die Übertragung beliebiger Prozesse selbst durch Prozesse möglich ist.

Desweiteren gibt es viele Versuche, das π-Kalkül selbst zu verändern. Beachtenswert sind vor allem jene, die die bereits genannten Asymmetrien beseitigen wollen. Am wichtigsten erscheint mir davon das sogenannte *Fusionskalkül*, entwickelt von Joachim Parrow und Björn Viktor [6]. Sie entfernen die beiden Konstruktionen zur Namensbindung (Restriktion $\mathrm{new}\, x\ P$ und Eingabe $y(x).P$ und ersetzen sie durch eine Einzige, den Reichweitenoperator $(x)P$, der den Namen x für den P lokal macht. Die Übertragung von Botschaften bei einer Aktion wird emuliert durch die *Fusion*, die einfach die jeweils die Namen gleichsetzt, die als Argumente der positiven beziehungsweise negativen Aktion auftreten, also etwa

$$\overline{u}\, vw.P \mid u\, xy.Q \xrightarrow{\{v=x,w=y\}} P \mid Q.$$

Dadurch verhalten sich die positive und negative Aktion exakt symmetrisch. Das entstehende Kalkül ist einfacher und sogar ausdrucksstärker als das π-Kalkül, und diese Erweiterungen vereinfachen viele Beweise über Eigenschaften von Prozessen.

Das π-Kalkül ist seit seiner Entstehung vor mehr als zehn Jahren sehr stark in der wissenschaftlichen Diskussion, was die große Anzahl von Publikationen zu diesem Thema beweist.

3.2 Anwendungen

Wozu kann das π-Kalkül dienen? Offensichtlich ist es ein eher theoretisches Instrument, denn es ist sehr einfach, sehr ausdrucksstark und vor allem sehr präzise. Damit ist es natürlich auch recht aufwendig, praktische Anwendungen im π-Kalkül darzustellen. So kann zwar relativ leicht objektorientierte Programmierung oder das λ-Kalkül eingebettet werden, die nötige Darstellung ist jedoch zu länglich für diese Arbeit.

Die Stärke des Kalküls liegt in der Darstellung von Kommunikation und Koordination und insbesondere der Mobilität von Prozessen, die es anders als seine Vorgänger als zentralen Punkt behandelt. Es kann also recht gut für die Spezifikation, aber wie wir sehen werden, sogar die Implementierung von mobilen Systemen eingesetzt werden. Dabei ist nicht zu vergessen, daß es sich um Darstellungssystem auf einer sehr niedrigen Ebene handelt. In vielen Fällen wird es daher sinnvoller sein, die Grundkonstrukte einer höheren Sprache in Pi darzustellen und ihre Eigenschaften so präzise festzulegen, und dann mit diesen höheren Konstrukten weiterzuarbeiten.

Die wohl prominenteste Anwendung ist die Sprache Pict, eingeführt von Benjamin Pierce und David Turner [7]. Dies ist der Versuch, auf der semantischen Basis des π-Kalkül eine Sprache aufzubauen, ähnlich wie dies funktionale Sprachen wie Haskell, ML ... mit dem λ-Kalkül tun. Pict übersetzt große Teile von Pi mit einigen Änderungen direkt in eine Kernsprache, auf der dann höhere Sprachkonstrukte aufgebaut werden.

Der folgende Prozeß soll beispielsweise einen „Cons-Zellen-Server“ implementieren, der eine Cons-Zelle (Mit Kopf hd und Rest tl) aufbaut und die Adresse entlang des Kanals r zurückgibt, wenn man ihm ein Tripel

```
[hd tl r]
```

übergibt:

```
cons?*[hd tl r] = (new l (r!l | l?*[n c] = c![hd tl]))
```

Dabei steht $x?$ für eine Eingabe, $y!$ für eine Ausgabe, und $z?*$ für eine sogenannte *replizierte Eingabe*.

Bei der Entwicklung von Pict stellten sich erhebliche Fragen unter Anderem in Bereich der Compilierung, da für eine akzeptable Performanz Dinge wie Prozeßerzeugung, Kontextumschaltung und Kommunikation über Kanäle sehr effektiv implementiert werden müssen.

Pict ist im Übrigen stark typisiert und es wird großer Wert auf ein praktisches Typsystem und Polymorphismus gelegt. Da wir diese Aspekte schon im π-Kalkül nicht behandelt haben, werden wir auch hier nicht näher auf sie eingehen.

3.3 Schlußbemerkung

Wie wir gesehen haben, ist das π-Kalkül zwar auf recht einfachen Konzepten aufgebaut, allein seine formale Einführung benötigt jedoch schon einigen Aufwand. Es ist ebenso

sehr ausdrucksstark, die Darstellung komplexer Systeme ist aber erwartungsgemäß aufwendig. Daher scheinen Vereinfachungen, wie sie etwa in [6] vorgeschlagen werden, sehr wünschenswert.

In jedem Fall ist die historische Bedeutung immens: Das π-Kalkül hat eine Reihe wissenschaftlicher Diskussionen und Arbeiten ausgelöst, die sich für die Betrachtung kommunizierender und mobiler Systeme als äußerst fruchtbar erweisen dürften. Ähnliche und weiterentwickelte Kalküle finden zunehmend Anwendung in der (wissenschaftlichen) Praxis ([7],[6]) und werden unser Verständnis für diesen lange vernachlässigten Zweig der Informatik nachhaltig beeinflussen. Für die präzise Darstellung von Kommunikation und Kooperation sind solche Systeme wohl zumindest eine der führenden und adäquatesten Beschreibungsweisen, die uns zur Verfügung stehen.

Literatur

[1] Milner, Robin. *Communicating and mobile systems: the π-calculus*. Cambridge University Press 1999

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

[3] Milner, Robin. *A Calculus of Communicating Systems*, Lecture Notes in Computer Science, Vol 92. Springer-Verlag 1980

[4] Schneider, Hans-Jochen (Hrsg.). *Lexikon der Informatik und Datenverarbeitung*[3]. Oldenbourg 1991

[5] Berry, G. und Boudol, G. The chemical abstract machine, in Proc. 17th Annual Symposium on Principles of Programming Languages. 1990

[6] Parrow, Joachim und Viktor, Björn. The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes, in Proc. 13th Annual IEEE Symposium on Logics in Computer Science. 1998

[7] Pierce, Benjamin und Turner, David. Pict: A programming language based on the Pi-calculus, in *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press 1998