Seminar
Parallelisierung und verteilte Ausführung funktionaler Programme
Thema
Ein Kalkül der mobilen Prozesse

von:
Nadali Walizadeh
Email: nadali@informatik.uni-frankfurt.de

nach einer Arbeit von:

Robin Milner, University of Edinburg, Scotland
Joachim Parrow, Swedish Institute of Computer Science, Kista, Sweden
David Walker, University of Technology,Sydney, Australia

June 1989 (Revised September 1990)

6. Feb 1997

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Es wird im folgenden ein Kalkül des Kommunikationssystems für die Beschreibung der Prozesse vorgestellt. Er ist ein mathematisches Modell, das die Mobilität der Prozesse bei der Kommunikation und die gegenseitige Wechselwirkung der nebeneinander laufenden Prozesse beschriebt. Das Kommunikationssystem wird nicht nur beim Datenaustausch zwischen benachbarten Komponenten des Systems beschrieben, sondern auch, wenn diese Komponenten mögliche Kommunikationskanäle unter sich austauschen. Der π-Kalkül ist also ein Kalkül, der insbesondere die Prozesse formalisiert, die ihre Struktur während der Kommunikation ändern.
Bis zum Zeitpunkt der Einfürung des π-Kalküls gab es andere Formalismen, die die Mobilität der Prozesse indirekt zum Ausdruck gebracht haben, wobei diese Strukturänderung der Prozesszustände nicht im Vordergrund stand; wie z.B. CCS (calculus of communicating systems) oder Petri Netz [13]. Es gab auch welche, die sie direkt - ohne mathematische Grundlagen ihrer Konzepte - beschrieben haben, wie das Aktormodell von Hewit [5]. In diesem Modell wurden die Prozesse als Argument über die Schnittstellen übergegeben, was beim π-Kalkül durch die Übergabe einer Verbindung realisiert wird. Der Unterschied zwischen den beiden Methoden wird hier näher behandelt. Eine wichtige Kritik an solchen Modellen ist, daß ihnen die mathematischen Grundlagen der mobilen Prozesse fehlen, was die zugrundliegende Idee der Einführung des π-Kalküls ist.

1.2 Historie

Der π-Kalkül ist eine Erweiterung der Prozessalgebra CCS (caculus of communicating systems). CCS wurde von R.Milner [9] eingeführt, ohne daß die Mobilität der Prozesse berücksichtigt wurde. U.Engburg und M.Nielsen haben CCS erweitert, indem sie die Mobilität der kommunizierenden Systeme hinzugefügt haben, sie haben aber ihre Arbeit nicht veröffentlicht. Viele Eigenschaften des π-Kalküls ist wie der Formalismus von Engburg und Nielsen; wie z.B Erweiterung der Bindungsbreich, starke Ähnlichkeit usw. Die Namensbindung wurde aber gesondert behandelt. Noch weitere Merkmale des π-Kalkül sind:

- Es wird im π-Kalkül zwischen Konstanten, Variablen und Kommunikationskanälen nicht unterschieden. Damit aber die Unterschiedliche Wirkung der Substituierung der Konstante und Variablen bei den Prozessen berücksichtigt wird, unterscheidet man zwischen Grund- und nicht-Grund-Äuivalenz der Prozesse.

- Die algebraischen Regeln wurden um die Expansionsregel erweitert. Damit sind diese Regeln als algebraische Theorie vollständig. Diese Regeln wurden herausgefiltert und bewiesen.

- Match-Konstruktur wurde eingeführt, der die Formalisierung der Berechnug von Werten vereinfacht.

- Es wurde eine einfache Methode für die Übersetzung vom λ-Kalkül eingefürt.

2 Konstruktion des π-Kalküls

2.1 Begriffe

Neben dem Flussdiagramm, das eine anschauliche Beschreibung der kommunizierenden Prozesse bietet, sind „Name" und „Agent" wichtige Begriffe, die uns ständig begleiten werden.

2.1.1 Name

Im π-Kalkül wird zwischen den Variablen, Konstanten und Kommunikationskanälen[1] nicht unterschieden. Alle werden als Namen bezeichnet. Dadurch wird der Kalkül vereinfacht, was aber einige Nachteile mit sich bringt. Wir werden z.B. sehen, daß bei der Substitution von freien Namen die starke Bisimulationseigenschaft der Agenten verloren geht. Die Substitution aller Namen führt uns zur starken Äquivalenz. Darauf werden wir später zurückkommen.

2.1.2 Agenten

Kommunikation und Nebenläufigkeit[2] sind zwei komplementäre Begriffe, die die Gesamtheit eines komplexen dynamischen Systems definieren. Einerseits Besteht so ein System aus einzelnen Teilen, die nebeneinander laufen und ihre eigene Identität besitzen, andererseits kommunizieren diese einzelne Bestandteile miteinander und machen somit ein zusammenarbeitenden Gesamtsystem aus.
Im π-Kalkül werden die Bestandteile eines System, die unabhängig von den anderen Teilen ihre eigene Identität haben, Agenten genannt. Was in einem Bereich als Teil eines System bezeichnet wird, wird in einem anderen Sinne als ein System mit seinen eigenen Bestandteilen bezeichnet. Um diese Doppeldeutigkeit zu umgehen, wird der Begriff Agent auf das Verhalten eines Systems bzw. der Bestandteile des Systems bezogen, d.h.: wenn ein System eine diskrete Aktion durchführt, dann handelt es sich um einen Agenten. Synonym zum Agenten werden hier auch die Begriffe „Prozeß" und „Prozessausdruck" verwendet.

Im folgenden wird angenommen, daß $\mathcal{N}$ eine endliche Menge von Namen sei. Weiter seien $x, y, z, w, v, u \in \mathcal{N}$ Variablen über Namen, und $\mathcal{K}$ eine Menge von Agentenbezeichnern, jeder mit eine Stelligkeit≥ 0; es seien $A, B, C, ... \in \mathcal{K}$; es seien P, Q, R Variablen über Agenten bzw. Prozessausdrücke. Sie kommen im π-Kalkül in sechs Variationen vor:

1. Summationsagent $\sum_{i \in I} P_i$, wobei I eine endliche Indexmenge ist.
   Dieser Agent verhält sich wie einer von P_i. $\mathbf{0}$ bezeichenet einen Agenten, der nichts tun kann. Deshalb wird er untätiger[3] Agent genannt. Es werden hier die Agenten $\mathbf{0}$ und $P_1 + P_2$ behandelt.

2. Eine Präfixform $\bar{y}x.P$, $y(x).P$ oder $\tau.P$.

[1] Englisch: communication links
[2] Englisch: concurrency
[3] Englisch: inaction

- $\bar{y}x$ wird negatives Präfix genannt. $\bar{y}$ kann als eine Ausgabeschnittstelle eines Agenten betrachtet werden. Der Agent $\bar{y}x.P$ gibt den Namen x auf der
  $\bar{y}$-Schnittstelle aus und verhält sich dann wie P. Das positive Präfix bindet keinen Namen. Man nennt y das Subjekt des Präfixes und x das Objekt des Präfixes.
- $y(x)$ wird positives Präfix genannt. Hier kann y als eine Eingabeschnittstelle eines Agenten betrachtet werden. Der Agent $y(x).P$ empfängt an der Schnittstelle x einen beliebigen Namen z und verhält sich wie $P\{z/x\}$. $P\{z/x\}$ bedeutet: Der Platzhalter x wird mit dem Namen z beim P ersetzt. Der Name x ist hier an des positive Präfix $y(x)$ gebunden.
- τ oder stilles Präfix. $\tau.P$ führt die interne Aktion τ aus und verhält sich wie P.

Die Präfixe $\bar{y}x, y(x)$ und τ sind Aktionen, die der Agent P durchführt.

3. Kompositionsagent $P_1 \mid P_2$.
   In diesem Agenten laufen P_1 und P_2 parallel. Ihre Aktionen können synchronisiert werden, d.h. eine Ausgabeaktion eines Agenten an der Schnittstelle $\bar{x}$ wird mit einer Eingabeaktion eines anderen an der Schnittstelle x synchronisiert. Der Kommpositionsagent $P_1 \mid P_2$ führt in dem Fall die interne Aktion τ durch.

4. Der Restriktionsagent $(x)P$.
   Dieser Agent ist ähnlich wie P, außer daß an der Schnittstelle x Aktionen verboten sind. Ausgenommen sind die Aktionen durch x, deren Ergebnis die interne Aktion τ ist, d.h. $P \equiv P_1 \mid P_2$. (siehe Kompositionsagent.)

5. Der Matchagent: $[x = y]P$.
   wenn x und y identisch sind, dann verhält sich dieser Agent wie P, sonst wie der Agent $\mathbf{0}$, d.h. er tut nichts.
   Wie vorhin erwähnt, die Verwendung dieses Agenten vereinfacht die Manipulation und die Berechnunng von Datenstrukturen. Es wird sich aber intessanterweise herausstellen, daß der π-Kalkül ohne ihn auch diese Aufgabe bewältigt kann.

6. Der definierte Agent $A(x_1, ..., x_n)$, wobei n die Stelligkeit von A ist, und alle x_i für $1 \leq i \leq n$ paarweise disjunk sind.
   Das bedeutet, daß es für den Agentbezeichner A eine eindeutige Definition
   $A(y_1, ..., y_n) \stackrel{\text{def}}{=} P$ geben muß, so daß nur $y_1, ..., y_n$ die eigentlichen unterschiedlich Namen sind, die in P frei vorkommen dürfen. Der Agent $A(x_1, ..., x_n)$ verhält sich wie $P\{x_1/y_1, ..., x_n/y_n\}$. Diese Definition bereitet eine Basis für die rekursive Definition vor, weil die rechte jeden Agenten ebenso wie A enthalten kann. Wir werden uns aber in diesem Zusammenhang nur mit endlichen Mengen von Agenten beschäftigen. Deshalb können die Agentbezeichner und damit ihre Definitionen weggelassen werden, d.h. es fehlt hier die ausführliche Behandlung der Rekursion. Wir werden sehen, daß im π-Kalkül die Rekursion durch die sogenannte Replikation beschrieben werden kann.

Man kann also die Syntax der Agenten im π-Kalküll folgendermaßen zusammenfassen:

$$\begin{array}{rcl} P & ::= & \mathbf{0} \\ & | & \bar{x}y.P \\ & | & x(y).P \\ & | & \tau.P \\ & | & (x)P \\ & | & [x=y]P \\ & | & P \mid Q \\ & | & P+Q \\ & | & A(y_1,...,y_n) \end{array}$$

wobei $\mathbf{0}$ ein Nulloperator ist, $\bar{x}y., x(y)., \tau., (x)$ und $[x=y]$ einstellige Operatoren sind, $|$ und $+$ binäre Operatoren sind, und n die Steligkeit von A ist.

Folgende Bemerkungen sind im Laufe dieses Artikels wichtig:

- Die Priorität der einzelnen Operatoren ist:

$$\text{Summation} < \text{Komposition} < \begin{cases} \text{Restriktion} \\ \text{Präfix} \\ \text{Match} \end{cases}$$

- In jedem Agenten der Form $x(y).P$ und $(y).P$ ist der Name y in P gebunden. Die Menge der gebundenen Namen von P wird mit $\mathrm{bn}(P)$ bezeichnet. Wenn $A(\tilde{x}) \stackrel{\mathrm{def}}{=} Q$, dann $\mathrm{bn}(A(\tilde{x})) = \mathrm{bn}(Q)$, wobei $\tilde{x} = x_1, ..., x_n$. Ein Namen kommt in einem Agenten frei, wenn er nicht im Bindungsbereich des Agenten liegt. Die Menge der freien Namen von P wird mit $\mathrm{fn}(P)$ und die Menge der Namen von P mit $\mathrm{n}(P)$ bezeichnet. Es gilt dann: $\mathrm{n}(P) = \mathrm{fn}(P) \cup \mathrm{bn}(P)$ bezeichnet. Es gilt weiter folgende Abkürzung: $\mathrm{fn}(P) \cup \mathrm{fn}(Q) \cup ... \cup \{x, y, , ...\}$ statt $\mathrm{fn}(P, Q, ..., x, y, ...)$

2.2 Eigenschaften des π-Kalküls

Die wichtigsten Eigenschaften vom π-Kalkül sind:

- Im Unterschied zum λ-Kalkül werden im π-Kalkül Namen durch Namen ersetzt, nicht durch Agenten (Prozeßausdrücke). Das gilt bis auf die parallele Komposition. Bei der parallelen Komposition wird ein negatives Präfix mit einem positiven Präfix synchronisiert. Wenn ein Agent Q den Zugriff auf den Agenten R über eine Verbindung zwischen dem P und Q vom P verlangt, kann man dies dadurch realisieren, daß man den Agenten R als Argument über die Schnittstelle vermittelt. Diese Wahl vermeidet man aus zwei Gründen:

1. Der formale Parameter bei Q, der für den Empfang vom R reserviert ist, kann mehrmals vorkommen, was zu Replikationen von R führt. Diesen Nebeneffekt will man im π-Kalkül vermeiden.
2. Wenn man R als Parameter in Q übergibt, so übergibt man auch das totale Zugriffrecht auf R. Uns interessiert aber meistens das partielle Zugriffsrecht auf R. Der Fall kommt vor, wenn R Verbindungen, die für Q unbekannt sind, zu siener Nachbaren hat.

– Die freien Namen eines Agenten teilen mit, mit welchen Agenten er in Verbindung ist. wenn z.B drei Agenten P, Q und R x als freien Namen haben, kann man das im Flußdiagramm wie in Abbildung (1) darstellen. Wenn die Verbindung x für drei Agenten restriktiert wird, wird im Flußdiagramm wie in Abbildung (2) dargestellt.

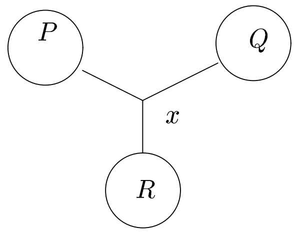


Abb.(1) $P \mid Q \mid R$

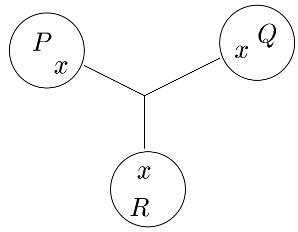


Abb.(2) $(x)(P \mid Q \mid R)$

– Da ein Agent einen freien Namen (z.B. eine Verbindung) über seine Schnittstellen erhalten kann, wird die Anzahl der freien Namen im Laufe jeder Sitzung variieren. In CCS ist das nicht der Fall, da sie eine feste Anzahl von Namen und Co-Namen (für Eingabe und Ausgabe) haben. Interessierende können in [9] mehr über CCS erfahren.

– Steht ein freier Name x in der Subjektposition eines Präfixes (z.B x in $\bar{x}y.P$ oder $x(y).P$), bedeutet das, daß der Agent über x mit anderen Agenten kommuniziert. Wenn x in der Objektposition steht (z.B x in $\bar{y}x.P$), dann gibt der Agent x über eine Schnittstelle aus.

– Der Unterschied zwischen zwei Formen der Namenbindungen wird im π-Kalkül ausdrüklich betont: $y(x).P$ und $(x)Q$.
In $y(x).P$ ist x ein Platzhalter für etwas, das über die y-Schnittstelle eingehen wird. Das kann jeder beliebige Name sein.
Im $(x)Q$ ist aber x ein privater Name für Q und darf nicht mit anderen Namen im Q verwechselt werden. Diese Vewechslung ist in CCS ausgeschlossen, da unterschiedliche Namen für Verbindungskanäle und zu übertragenden Daten verwenden werden.

2.3 Aktionen

Agenten werden mit ihren Zuständen identifiziert. Der Übergang von einem Zustand zu dem anderen geschieht durch eine Aktion. Man kann einen Agenten mit einem

endlichen Automaten vergleichen. In dem Sinne ist eine Aktion ein Zustandsübergang, der den Agenten von einem Zustand in den anderen überführt.
Ein Übergang im π-Kalkül wird formalisiert durch:

$$P \xrightarrow{\alpha} Q$$

Das bedeutet; P kann bei der Durchführung der Aktion α in Q wechsleln(übergehen.)
Die Aktionen, die hier vorkommen, sind von folgenden Arten:

- τ : Diese bezieht sich in erster Line auf gleichzeitige Zustandsänderung zweier Agenten, die miteinander kommunizieren. Wenn ein Agent eine Nachricht auf seiner Ausgabeschnittstellle ausgibt, die auf einer Eingabeschnittstelle eines anderen Agenten als Eingabe konsumiert wird, so ändern die beiden Agenten gleichzeitig ihren Zustand. Man kann diese Zusammengesetzung zweier Agenten, die parallel laufen, als einen Agenten bezeichnen, dessen Zustandsänderung mit τ geschieht. Da diese Aktion bezüglich des zusammengesetzten Agenten intern[4] geschieht, kann man sie auf allen komplementären Aktionen in einem zusammengesetzten Agenten verallgemeinen. Daraus ergibt sich, daß die Agenten, die dieselbe externe Aktionen verbieten, als äquivalent betrachtet werden. Darauf werden wir später genauer eingehen. Es sei angemerkt, daß eine interne Aktion τ auch von dem Agenten $\tau.P$ durchgeführt werden kann.

- Die freie Ausgabeaktion $\bar{x}y$. In dem Übergang $P \xrightarrow{\bar{x}y} Q$ wird P an seiner Ausgabeschnittstelle x den Namen y ausgeben und sich dann wie Q verhalten. Diese Aktion wird von dem Agenten $\bar{x}y.P$ durchgeführt.

- Die Eingabeaktion $x(y)$. $P \xrightarrow{x(y)} Q$ bedeutet das, daß P an seiner Eingabeschnittstelle irgendeinen Namen w erhält, und dabei in $Q\{w/y\}$ übergeht. (y) ist also eine Stelle, wo der empfangene Name stehen wird. Diese Aktion wird von dem Präfixagent $x(y).P$ durchgeführt.

- Die gebundene Ausgabeaktion $\bar{x}(y)$. $P \xrightarrow{\bar{x}(y)} Q$ impliziert, daß P einen gebundenen Namen an der Schnittstelle $\bar{x}$ ausgeben will. (y) ist ein Zeiger zu der Stelle, wo dieser Name steht. Diese Aktion wird vom $(y)\bar{x}y.P$ durchgeführt.

Die interne Aktion τ und freie Ausgabeaktion $\bar{x}y$ werden freie Aktionen genannt, während die Eingabeaktion $x(y)$ so wie die gebundenen Ausgabeaktion $\bar{x}(y)$ gebundene Aktionen genannt werden. Die Tabelle unten zeigt die Aktionen und ihre Eigenschaften zusammengefaßt .

α	Art	frei/gebunden	psitiv/negativ	fn(α)	bn(α)
τ	still	f	0	$\emptyset$	$\emptyset$
$\bar{x}y$	freie Ausgabe	f	$-$	$\{x, y\}$	$\emptyset$
$x(y)$	Eingabe	b	$+$	$\{x\}$	$\{y\}$
$\bar{x}(y)$	gebundene Ausgabe	b	$-$	$\{x\}$	$\{y\}$

Aktionstabelle.

[4] τ ist also bzgl. des zusammengesetzten Agenten eine Aktion ohne Datenaustausch. Deshalb wird Sie als stille (Englisch: silent) Aktion bezeichnet.

2.4 Übergänge : Operationale Semantik

Eine Übergangsrelation $\xrightarrow{\alpha}$ ist die kleinste Relation in Bezug auf die Agenten definiert, die eine von den Aktionsregeln erfüllt, die in der Tabelle unten stehen.

$$TAU-ACT: \frac{-}{\tau.P \xrightarrow{\tau} P} \qquad OUTPUT-ACT: \frac{-}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$$

$$INPUT-ACT: \frac{-}{x(z).P \xrightarrow{x(w)} P\{w/z\}} \qquad w \notin \mathrm{fn}((z)P)$$

$$SUM: \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad Match: \frac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}$$

$$IDE: \frac{P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha} P'}{A(\tilde{y}) \xrightarrow{\alpha} P'} \qquad A(\tilde{x}) \stackrel{\mathrm{def}}{=} P$$

$$PAR: \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$$

$$COM: \frac{P \xrightarrow{\bar{x}y} P' \quad Q \xrightarrow{x(z)} Q'}{P|Q \xrightarrow{\tau} P' \mid Q'\{y/z\}} \qquad CLOSE: \frac{P \xrightarrow{\bar{x}(w)} P' \quad Q \xrightarrow{x(w)} Q'}{P|Q \xrightarrow{\tau} (w)(P' \mid Q')}$$

$$RES: \frac{P \xrightarrow{\alpha} P'}{(y)P \xrightarrow{\alpha} (y)P'} \qquad y \notin \mathrm{n}(\alpha)$$

$$OPEN: \frac{P \xrightarrow{\bar{x}y} P'}{(y)P \xrightarrow{\bar{x}w} P'\{w/y\}} \qquad y \neq x \quad w \notin \mathrm{fn}((y)P')$$

Tabelle:Aktionsregeln. Die Regeln beinhalten binäre Operationen + und |. Sie haben eine symmetrische Form.

3 Basisanatomie des π-Kalküls

Wie vorhin erwähnt, ist die Bindung von Namen, die in dem positiven Präfix und der Restriktion vorkommt, das Herzstück des π-Kalküls. Deshalb ist es sehr wichtig, diesen Sachverhalt bei den kommunizierenden Agenten zu berücksichtigen. Hier werden wir beobachten, wie Mobilität und Bindungsbereich sich gegenseitig beeinflussen.

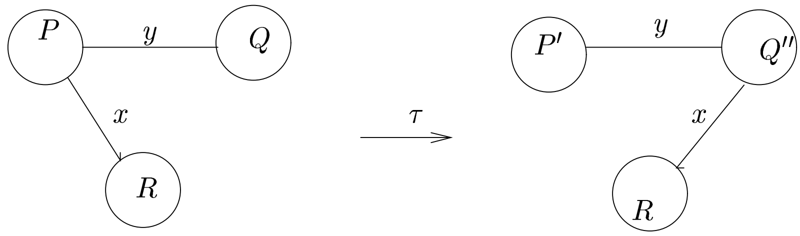
3.1 Namentransport

Wenn ein Agent P eine Verbindung x zu R hat, welche er über seine Verbindung y zu Q, der zum Empfang bereit ist, übermitteln will, dann unterscheidet man folgende Fälle:

1. **Fall**: $x \in \mathrm{fn}(P)$ und $x \notin \mathrm{fn}(Q)$. Also $P \equiv \bar{y}x.P'$ und $Q \equiv y(z).Q'$. Der Übergang ist dann:

$$\bar{y}x.P' \mid y(z).Q' \mid R \xrightarrow{\tau} P' \mid Q'\{x/z\} \mid R \tag{1}$$

Mit dem Flußdiagramm kann man das so darstellen:



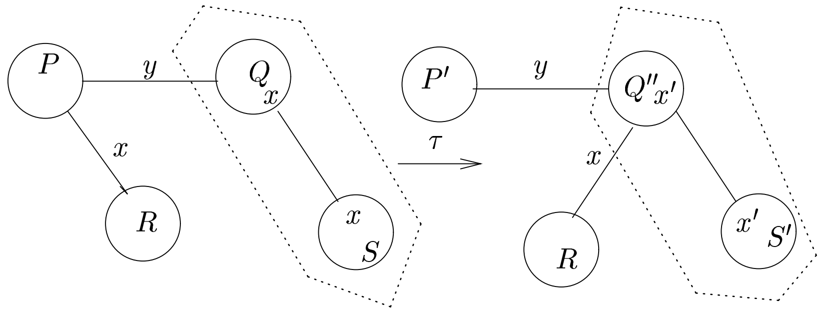
Q'' ist in dem Diagramm $Q'\{x/z\}$. In dieser Übertragung spielt es keine Rolle, ob der Name y für P und Q privat ist oder nicht. Wenn y für P und Q privat ist, dann gilt:

$$(y)(\bar{y}x.P' \mid y(z).Q') \mid R \xrightarrow{\tau} (y)(P' \mid Q'\{x/z\}) \mid R \tag{2}$$

2. **Fall**: $x \in \mathrm{fn}(P)$ und $x \in \mathrm{bn}(Q)$. D.h. x existiert bereits zwischen Q und einem anderem Agent S und ist privat für Q. Also $P \equiv \bar{y}x.P'$ und $Q \equiv y(z).Q'$. Der Übergang ist dann:

$$\bar{y}x.P' \mid R \mid (x)(y(z).Q' \mid S) \xrightarrow{\tau} P' \mid R \mid (x')(Q'\{x'/x\}\{x/z\}|S\{x'/x\}) \tag{3}$$

Das Flußdiagramm für diesen Vorgang ist:



Bindungsbereich von x Bindungsbereich von x'

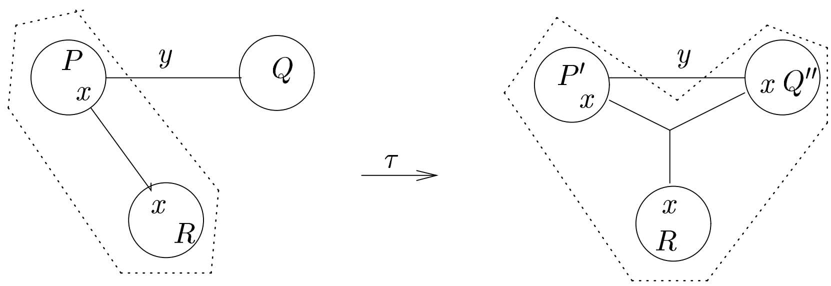
Erlüterung zum Diagramm: $Q'' \equiv Q'\{x'/x\}\{x/z\}$, $S' \equiv S\{x'/x\}$ und $x' \notin$ fn(Q) ist ein neuer Name.
Aus Gleichung (3) wie auch aus dem Digramm geht hervor, wenn x breits als gebundener Name in Q vorhanden ist, wird x in Q zu x' bei der Übertragung von x. Diese Art der Verbindungsübertragung beeinflußt den privaten Verbindungsbereich zwischen Q und S[5]. Die Umbenennung von gebundenen Variablen - *Alphakonversion* - ist nicht nur charakteristisch für den π-Kalkül, sondern läßt sich auch beim λ-Kalkül beobachten, wenn sich dies als notwendig erweist. Die Gleichung (3) gibt also ein Beispiel für die *Alphakonversion*:

$$(x)(Q \mid S) \;=\; (x')(Q\{x'/x\} \mid S\{x'/x\}) \qquad (4)$$

3. **Fall**: $x \in$ bn(P) und $x \notin$ n(Q). Man kann sich P als ein Programm, R als ein Register, welches eine lokale Variable von P speichert, x als ein Zeiger auf lokale Variable und Q als eine Prozedur vorstellen. P gibt über seine Verbindung y zu Q den Zeiger zu seiner lokalen Variable über.
Der π-Kalkül formalisiert diesen Sachverhalt folgendermaßen:

$$(x)(\bar{y}x.P' \mid R) \mid y(z).Q' \xrightarrow{\tau} (x)(P' \mid R \mid Q'\{x/z\}) \qquad (5)$$

Das dazugehörige Flußdiagramm ist:



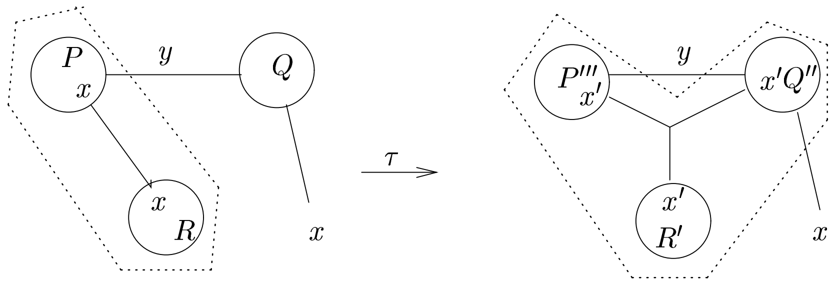
Bindungsbereich von x Bindungsbereich von x

Im Diagramm ist: $Q'' \equiv Q'\{x/z\}$. Aus dem Formalismus sowie dem Diagramm kann man ersehen, daß die private Verbindung x zwischen P und R zur privaten Schnittstelle zwischen Q und R gewachsen ist. Der Bindungsbereich von x ist also erweitert worden[6].

[5] Englisch: scope intrusion
[6] Englisch: scope extrusion

4. **Fall**: $x \in \text{bn}(P)$ und $x \in \text{fn}(Q)$. Das enspricht dem 3. Fall, aber zusätzlich ist eine *Alpha-Konversion* notwendig. d.h. die gebundenen Namen müsen umbenannt werden. Der Übergang ist im π-Kalkül:

$$(x)(\bar{y}x.P' \mid R) \mid y(z).Q' \xrightarrow{\tau} (x')(P'\{x'/x\} \mid R\{x'/x\} \mid Q'\{x'/z\}) \qquad (6)$$



Bindunsbereich von x Bindunsbereich von x'

Im Diagramm ist: $P'' \equiv P'\{x'/x\}$, $Q'' \equiv P'\{x'/z\}$ und $P' \equiv P\{x'/z\}$

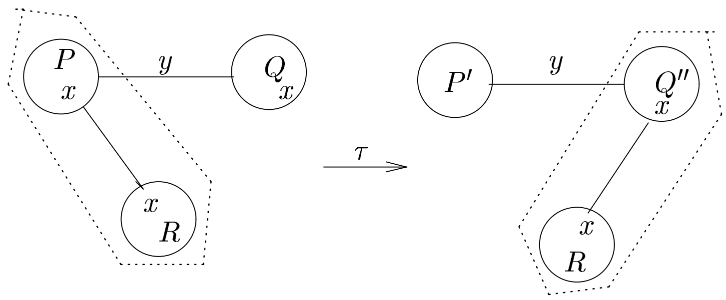
5. **Fall**: $x \notin \text{fn}(Q)$ und $x \notin \text{fn}(P')$. Dieser Fall hat zur Folge, daß der Bindungsbereich von x fortzieht[7].
Wir werden später sehen, daß es eine allgemeine Regel gibt:

$$(x)(P_1 \mid P_2) = P_1 \mid (x)P_2 \qquad \text{wenn } x \notin \text{fn}(P_1) \qquad (7)$$

Es wird also der 3.Fall durchgeführt, und auf (5) die Regel (7) angewendet:

$$(x)(P' \mid R) \mid Q'\{x/z\} = P' \mid (x)(R \mid Q'\{x/z\}) \qquad (8)$$

Man kann diesen Sachverhalt im Diagramm zeigen:



Bindungsbereich von x Bindungsbereich von x

3.2 Transport zweier Namen

Ein Agent P beabsichtigt, ein Paar $(u\ ,\ v)$ von Namen zu einem Agenten Q oder R über eine gemeinsame Verbindung zu senden. Es gelten folgende syntaktische Gleichungen:

$$P \equiv \bar{x}u.\bar{x}v.P' \quad , \quad Q \equiv x(y).x(z).Q' \quad , \quad R \equiv x(y).x(z).R'$$

[7] Englisch: migration of scope

Problem: Wie kann man sicherstellen, daß im Kompositionagenten $Q \mid P \mid R$ das gesendete Paar beim Zielagenten ankommt? Welche Maßnamen trifft der π-Kalkül, um zu verhindern, daß z.B Q nur u und R nur v erhält?
Lösung: Anstatt u und v gleichzeitig zu senden, sendet P den Namen eines kleinen privaten Prozesses w zu Q oder R, dessen Aufgabe nur darin besteht, erst u und dann v an R oder Q oder beide zu senden. Basierend auf dieser Idee kann man schreiben:

$$P \equiv (w)(\bar{x}w.P'|\bar{w}u.\bar{w}v.\mathbf{0}) \qquad Q \equiv x(w).w(y).w(z).Q'$$
$$R \equiv x(w).w(y).w(z).R' \quad \text{wobei} \quad w \notin \ \mathrm{fn}(P', Q', R').$$

Die Aktion $\bar{x}w$ nennt man atomare Aktion. Diese Aktion bildet zusammen mit den Aktionen des Agenten $\bar{w}u.\bar{w}v.\mathbf{0}$ eine molekulare Aktion. Mit Hilfe dieser Methode können, wie wir später sehen werden, die Datenstrukturen als Prozesse aufgefaßt und formalisiert werden.

Damit hat der Agent $Q \mid P \mid R$ eine von diese zwei Übertragungen zu erfüllen:

$$\begin{aligned} Q|P|R &\xrightarrow{\tau} (w)(w(y).w(z).Q'|P'|\bar{w}u.\bar{w}v.\mathbf{0})|R \\ &= \tau.\tau.(Q'\{u/y\}\{v/z\}|P'|R) \end{aligned}$$

oder

$$\begin{aligned} Q|P|R &\xrightarrow{\tau} Q|(w)(P'|\bar{w}u.\bar{w}v.\mathbf{0})|w(y).w(z).R' \\ &= \tau.\tau.(Q|P'|R'\{u/y\}\{v/z\}) \end{aligned}$$

4 Der π-Kalkül in der reellen Applikation

Wir werden in den nachfolgenden Abschnitten diese Abkürzungen nutzen:

alte Schreibweise	neue Schreibweise
$\bar{x}\epsilon.P$ wobei ϵ ein spezieller Name ist , der nie gebunden ist.	$\bar{x}.P$
$x(y).P$ *wobei* $y \notin \mathrm{fn}(P)$	$x.P$
$\bar{x}y.\mathbf{0}$	$\bar{x}y$
$x(v).([v = y_1]P_1 + [v = y_2]P_2, ...)$	$x : [y_1 \Rightarrow P_1\ ,\ y_2 \Rightarrow P_2\ ,\ ...]$ oder kurz $x : [y_i \Rightarrow P_i]_{i \in I}$ wobei $v \notin \mathrm{fn}(P_i)$ und y_i paarweise disjunk sind.

4.1 Ein Aktivator

Um einen Prozeß P zu aktivieren, stehen zwei Alternativen zur Verfügung:

1. P wird direkt aufgerufen.

2. Man bindet ein Präfix z an P und sendet z über eine Verbindung x zu einem Aktivator $Exec(x)$[8], wobei

$$Exec(x) \stackrel{\mathrm{def}}{=} x(y).\bar{y} \qquad (9)$$

D.h. der Akivator erhält über eine Schnittstelle x den Namen eines Triggers, den er y nennt, über den er den Prozeß aktiviert. Dabei muß gelten: $x, z \notin \mathrm{fn}(P)$. Im π-Kalkül folmalisiert man einen Agenten:

$$(z)(\bar{x}z \mid z.P) \qquad (10)$$

der sich bei der Präsenz von $Exec(x)$ wie P verhält: d.h. P ist equivalent zum Agenten:

$$(x)((z)(\bar{x}z \mid z.P) \mid Exec(x)) \quad (11)$$

Zum Beweis wende Gleichung (8) an:

$$\begin{aligned}(x)((z)(\bar{x}z \mid z.P) \mid Exec(x)) \quad &= \quad (x)(z)(\bar{x}z \mid z.P \mid x(y).\bar{y}) \\ & \quad\ \ \tau.(x)(z)(\mathbf{0} \mid z.P \mid \bar{z}) \\ & \quad\ \ \tau.\tau.(x)(z)(\mathbf{0} \mid P \mid \mathbf{0}) \\ & \quad\ \ \tau.\tau.P \qquad \text{da } x, z \notin \mathrm{fn}(P)\end{aligned}$$

[8] Englisch: executor

4.2 Versendung eines Prozesses als Parameter

Aus einigen Gründen, die vorhin erwähnt wurden (s. Kapitel 2.2: Eigenschaften des π-Kalküls), wird die Versendung der Agenten und Prozessen im π-Kalkül vermieden. Statt dessen wird die Versendung der Schnittstellen empfohlen und es wird gezeigt, daß man damit auch das erreichen kann, was man mit der Versendung der Prozesse erreichen will. Es wird deswegen diese Versendung nicht ausführlich vorgestellt. Damit aber der Zusammenhang der Mobilität und dem Bindungsbereich untersucht werden kann, sind einige Anmerkungen erforderlich.
Angenommen, ein Prozeß Q soll den Prozeß P zu einem Aktivator senden, und der Aktivator soll P parallel zum Prozeß R laufen lassen:

$$(x)(\bar{x}P.Q \mid x(p).(p \mid R)) \qquad (12)$$

dabei ist p eine Prozeßvariable und $x \notin \text{fn}(P,\ Q,\ R)$.
Es stellt sich nun die Frage: Was passiert, wenn zwischen P und Q eine private Verbindung w besteht? D.h., was ist der nächste Zustand dieses Agenten:

$$(x)((w)(\bar{x}P.Q) \mid x(p).(p \mid R)) \qquad (13)$$

Es gibt zwei Alternativen, den privaten Namen w zu behandeln. In beiden Alternativen wird Wert darauf gelegt, ob $w \in \text{fn}(R)$ oder $w \notin \text{fn}(R)$:

- Erste Alternative: die Verbindung w zwischen dem Prozeß P und dem Prozeß Q wird gebrochen. Wir untersuchen zwei Fälle, indem wir w beobachten:

  1. $w \notin \text{fn}(R)$. w ist also nicht als eine Verbindung (Schnittstelle) bei R reserviert. Wenn also P mit R parallel laufen soll, wird R nicht über w verfügen. w ist deshalb im $(w)(\bar{x}P.Q)$ nur eine private Schnittstelle für Q. Formal folgt aus (13):

  $$\begin{aligned}&(x)((w)(\bar{x}P.Q) \mid x(p).(p \mid R))\\ =\ &\tau((w)Q \mid (P \mid R)) \qquad (14)\end{aligned}$$

  2. $w \in \text{fn}(R)$. w ist dann eine Verbindung zwischen P und R. In diesem Fall wird bei der Versendung des Prozesses sein ganzer Kontext übertragen. Das ist wie Übergabe einer Funktion als Parameter im LISP. Diese Art der Bindung entspricht der sogenannten "dynamischen Bindung"[9]. Die freien Variablen des Prozesses werden dann in der neuen Umgebung interpretiert. Im π-Kalkül wird jedoch nur die statische Bindung behandelt.

- Zweite Alternative: die Verbindung w bleibt unverändert. Diese entspricht dem 3. Fall im vorigen Kapitel: d.h. eine Art von Bereichsausdehnung. Hier wird ein neuer Name w' mit $w' \notin \text{fn}(P, Q, R)$ ausgewählt, und w in w' umbenannt, um eine Verwechslung mit dem eventuell existierenden w-Verbindung im R auszuschließen.

$$\begin{aligned}&(x)((w)(\bar{x}P.Q) \mid x(p).(p \mid R))\\ =\ &\tau(w')(Q\{w'/w\} \mid (P\{w'/w\} \mid R)) \qquad (15)\end{aligned}$$

[9] Englisch: dynamic binding

Wir zeigen hier, daß im π-Kalkül die Versendung der Schnittstellen zum selben Ergebnis wie die Versendung der Prozesse führt: Deshalb wird die Versendung der Schnittstellen der Versendung der Prozesse, die einge Seiteneffekte verursacht, vorgezogen. Im oben genannten Beispiel transportiert man anstelle des Prozesses P zu R den Namen eines Links (d.h. einer Verbindung) z zu R. Nachdem R den Link empfangen hat, wird dieser Verbindung zwischen P und R aktiv. Statt (13) erhalten wir:

$$(x)((z)(w)(\bar{x}z.(z.P \mid Q)) \mid x(y).\bar{y}.R) \qquad (16)$$

wobei $y,\ z \notin \mathrm{fn}(R)$. Da $x \notin \mathrm{fn}(P, Q, R)$, also kann man x weglassen:

$$\tau(z)((w)(z.P \mid Q) \mid \bar{z}.R)$$

Bei der Anwendung der Regel (7) und Umbenennung vom w in w', mit $w' \notin \mathrm{fn}(R)$ folgt:

$$\tau.(z)(w')(z.P\{w'/w\} \mid Q\{w'/w\} \mid \bar{z}.R) \qquad (17)$$

Wir wissen, daß $z \notin \mathrm{fn}(R)$, und z ein Link ist, der zum Prozeß Q versendet wurde, um dadurch den Prozeß P zu aktivieren. Es wird danach der Link z nicht mehr gebraucht. Wir können also z weglassen:

$$= \quad \tau.\tau.(w')(P\{w'/w\} \mid Q\{w'/w\} \mid R) \quad (18)$$

Mit zwei zusätzlichen τ-Aktionen erhalten wir Gleichung (15). Deshalb ist diese Behauptung bei Präsenz von expliziter Rekursion (also nicht mit Hilfe von Y-Kombinator, was bei λ-Kalkül[10] üblich ist) wahr: die Schnittstellenversendung ist genauso ausdrucksfähig wie Prozessversendung. Der Grund liegt darin, daß wenn eine Kopie eines Prozesses von dem Intresse ist, erweist sich die Schnittstellenvesendung nachteilig, da diese nicht den Prozeß ersetzen kann. Deshalb wird die Rekursion im π-Kalkül explizit vorgegeben. In einer Hinsicht erweist sich die Linksübertragung noch ausdrucksfähiger als Prozzessübertragung, weil nicht nur der Trigger eines Prozesses sondern jede beliebige Verbindung des Prozesses versendet werden kann.

[10] Zur Erinnerung: $YH = H(YH)$, wobei $Y = (\lambda h.(\lambda x.h\ (x\ x))(\lambda x.h\ (x\ x)))$

4.3 Werte und Datenstrukturen

Es gelten im folgenden diese Abkürzungen:

$$\bar{x}y_1 \cdots y_n \quad statt \quad \bar{x}y_1. \cdots .\bar{x}y_n \tag{19}$$

$$x(y_1) \cdots (y_n) \quad statt \quad x(y_1). \cdots .x(y_n) \tag{20}$$

$$x : [..., v(y_1) \cdots (y_n) \Rightarrow P, ...] \quad statt \quad x : [..., v \Rightarrow x(y_1) \cdots (y_n)P, ...] \tag{21}$$

Im π-Kalkül markiert man n Konstanten $v_1, ..., v_n$, wenn die Menge der zu verarbeitenden Werte aus der endlichen Menge $V = \{v_1, ..., v_n\}$ ist. Z.B. sei $V = \{t, f\}$ die Menge der Wahrheitswerte. Dann setzt man $\underline{t} = T$ und $\underline{f} = F$, wobei T und F konstante Werte sind.

1. Beispiel: man kann einen Prozeß, der einen Wahrheitswert von der Schnittstelle y (sie kann z.B. ein Register sein) auf die Schnittstelle z kopiert, so definieren:

$$Copy(y,\ z) \stackrel{\text{def}}{=} y : [T \implies \bar{z}T,\ F \implies \bar{z}F] \tag{22}$$

2. Beispiel: ein Prozeß, der die Konjunktion zweier Wahrheitswerte, die von den Verbindungen x und y angekommen sind, auf der Schnittstelle z ausgibt, kann so definiert werden:

$$And(x,\ y,\ z) \stackrel{\text{def}}{=} x : [T \implies Copy(y,\ z),\ F \implies \bar{z}F] \tag{23}$$

Aus den zwei Beispielen können wir erkennen, daß eine n-stellige Funktion mit $n + 1$ Namen dargestellt werden kann. Für den Fall $n = 0$ definiert man (z.B.) $True(x) \stackrel{\text{def}}{=} \bar{x}T$ und $False(x) \stackrel{\text{def}}{=} \bar{x}F$, wobei x ein Pointer ist, der auf den zugehörigen Wert zeigt. Wir wollen uns die 0-stellige Funktionen, $True(x)$ und $False(x)$, in einem Beispiel anschauen:

$$\begin{aligned}(x)(True(x) \mid Copy(x,\ y)) &= \tau.True(y) \\ (x)(y)(True(x) \mid False(y) \mid And(x,\ y,\ z)) &= \tau.\tau.False(z)\end{aligned}$$

Wenn die Menge der zu manipulierenden Werte unendlich ist, wird z.B die folgende Definition für die Nachfolgefunktion ungültig, weil n eine unendliche Summe umfaßt. Die Funktion hat zwei Schnittstellen x und y. Wenn man eine natürliche Zahl n an der Schnittstelle x an die Funktion übergeben würde, würde sie die nächste natürliche Zahl an der Schnittstelle y ausgeben:

$$Succ(x,\ y) \stackrel{\text{def}}{=} x : [\underline{n} \implies \bar{y}\underline{n+1}]_{n \in N}$$

Für die Lösung dieses Problemes stellt der π-Kalkül eine Listenstruktur zur Verfügung, auf der zwei Operatoren 'nil' (nullstellig) und 'cons' (binär) operieren. Der Agent $[[L]](x)$ wird dann die Elemente der Liste L stückweise über die Schnittstelle x liefern. Jede Liste ist durch Konstanten NIL und CONS definiert. Die übersetzung der Listen ist formalisiert durch:

$$[[\mathrm{nil}]](\mathrm{x}) \stackrel{\text{def}}{=} \bar{x}\mathrm{NIL} \tag{24}$$

$$[[\mathrm{cons}(\mathrm{L}_1,\mathrm{L}_2)]](\mathrm{x}) \stackrel{\text{def}}{=} (y)(z)(\bar{x}\mathrm{CONS}.\bar{x}y.\bar{x}z \mid [[L_1]](y) \mid [[L_2]](z)) \tag{25}$$

In einem Agenten der Form $(x)(P \mid [[L_1]](x))$ holt der Agent P die Liste L elementweise durch den privaten Namen x. Dabei kann jedes Element selbst eine Liste sein.

1. Beispiel: $L = $ cons(cons(nil,nil),nil) wird so übersetzt:

$$[[L]](x) = (y)(z)(\bar{x}\mathrm{CONS}yz \mid (v)(w)(\bar{y}\mathrm{CONS}vw \mid \bar{v}\mathrm{NIL} \mid \bar{w}\mathrm{NIL}) \mid \bar{z}\mathrm{NIL})$$

2. Beispiel: $Equal(x,y,b)$ soll die Gleichheit der Listen x und y prüfen und das Ergebnis auf der Ausgabeschnittstelle b ausgeben.

$$\begin{aligned}
Equal(x,y,b) &\stackrel{\text{def}}{=} x : [\mathrm{NIL} \Rightarrow Null(y,b), \mathrm{CONS}(x_1)(x_2) \Rightarrow \\
&\qquad Consequal(x_1,x_2,y,b)] \\
Null(y,b) &\stackrel{\text{def}}{=} y : [\mathrm{NIL} \Rightarrow True(b), \mathrm{CONS}(y_1)(y_2) \Rightarrow False(b)] \\
Consequal(x_1,x_2,y,b) &\stackrel{\text{def}}{=} y : [\mathrm{NIL} \Rightarrow False(b), \mathrm{CONS}(y_1)(y_2) \Rightarrow \\
&\qquad (b_1)(b_2)(Equal(x_1,y_1,b_1) \\
&\qquad\qquad \mid Equal(x_2,y_2,b_2) \\
&\qquad\qquad \mid And(b_1,b_2,b)
\end{aligned}$$

Eine Beobachtung ergibt, daß für die Übersetzung der Datenstrukturen nur eine endliche Menge von Konstantennamen notwendig ist (T,F,CONS und NIL). Es gibt aber Übersetzungen, die ohne Konstantennamen auskommen. Die Idee dabei ist, durch Matching die Namen, die bei den Agenten privat vorkommen, zu unterscheiden. Z.B.man kann die Wahrheitswerte folgendermaßen definieren:

$$True(x) \stackrel{\text{def}}{=} (u)(v)(\bar{x}uvu)$$

$$False(x) \stackrel{\text{def}}{=} (u)(v)(\bar{x}uvv)$$

Wie vorhin behauptet, kann man sogar die Matchform weglassen, indem man eine andere Methode für die Matchform findet. Betrachte folgenden Agenten:

$$P \equiv x(u)(v)(w).([w=u]P_1 \ + \ [w=v]P_2)$$

Definiert man die Wahrheitswerte folgendermaßen:

$$True(x) \stackrel{\text{def}}{=} (u)(v)(\bar{x}uvu.\bar{u})$$

$$False(x) \stackrel{\text{def}}{=} (u)(v)(\bar{x}uvv.\bar{v})$$

so kann man P ohne die Matchform definieren:

$$P \equiv x(u)(v).(u.P_1 \ + \ v.P_2)$$

4.4 Reduktion des Kombinatorgraphen

Damit die Programme in den funktionalen Sprachen zur Kompilierung vorbereitet werden, übersetzt man sie erst in λ-Terme. Die λ-Terme werden danach so transformiert, daß die Terme nur von (Super)Kombinatoren gebildet sind(*lambda-lifting*). Ein solcher Term kann durch einen Kombinatorgraph repräsentiert werden. Eine Applikation ist ein binärer Operator. In so einem Graphen steht in jedem inneren Knoten eine Applikation, die einen rechten und einen linken Sohn hat, in den Blättern stehen die Kombinatoren und ihre Argumente. Die hier vorgestellte Reduktion des Kombinatorgraphen simuliert die Termreduktion. Im Graphen sehen wir, daß zwei Applikationsknoten auf den selben Knoten angewendet werden. Diese Eigenschaft bezeichnet man als „Sharing". Die Benutzung dieser Methode dient zur effizienten Implementierung der funktionalen Programmiersprachen. Zusätzlich werden die drei Kombinaroren $\mathbf{S}$, $\mathbf{K}$ und $\mathbf{I}$ verwendet, die Implementierung der funktionalen Programmiersprachen besser als Superkombinatoren unterstützen, indem sie die Implementierung vereinfachen.
Wir nehmen an, daß in folgenden Abschnitten M, N und P transformierte Programmausdrücke sind, die mit $\mathbf{S}$, $\mathbf{K}$ und $\mathbf{I}$ Kombinatoren aufgebaut sind. Man kann die Syntax dieser Terme definieren durch:

$$M \quad ::= \quad \mathbf{S} \mid \mathbf{K} \mid \mathbf{I} \mid (M\ N)$$

wobei die Kombinatoren wie folgt definiert sind:

$$\begin{aligned} \mathbf{S}\,M\ N\ P &\longrightarrow M\ P\ (N\ P) \\ \mathbf{K}\,M\ N &\longrightarrow M \\ \mathbf{I}\,M &\longrightarrow M \end{aligned}$$

Um die Reduktion des Kombinatorgraphen im π-Kalkül zu übersetzen, werden 4 weitere Kombinatoren $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{K}_1$ und $\mathbf{I}_1$ zur Hilfe genommen und $\mathbf{S}$, $\mathbf{K}$, $\mathbf{I}$ in $\mathbf{S}_0$, $\mathbf{K}_0$, $\mathbf{S}_0$ umbenannt werrden. Es gibt für jeden Kombinator eine Reduktionsregel, die angewendet wird, wenn der zugehörige Kombinator beim linkesten Sohn steht. Die Abbildung auf der nächsten Seite zeigt diese Regel.

Aus dem Diagramm erkennt man, daß jeder Kombinator mit dem Index i mit $i+1$ Parametern beschrieben werden kann. Die ersten i-Parameter sind die Kinder des Kombinators, und der letzte Parameter repräsentiert den Vater des Kombinators. Es wird die Reduktion des Kombinatorgraphen im π-Kalkül durch die Kompositionsagenten dargestellt.

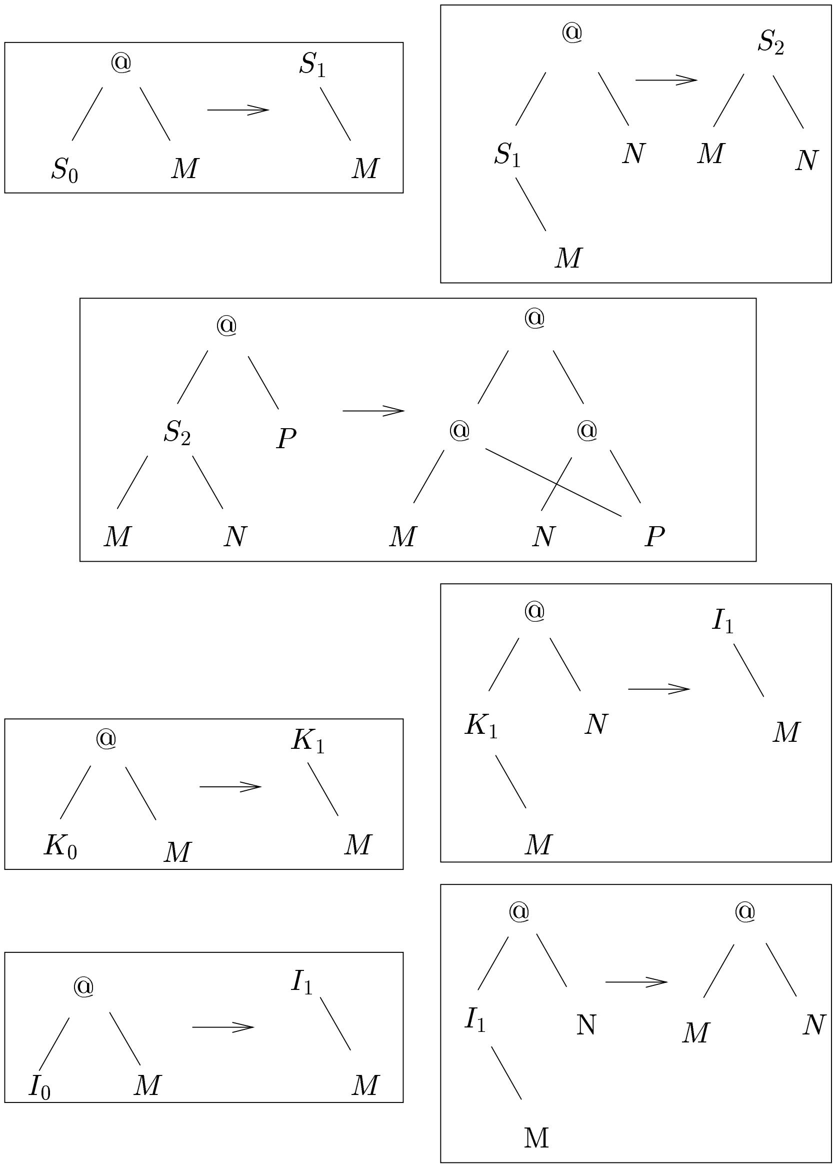
Dies sind die Definitionen der Kombinatoren:

$$
\begin{aligned}
\mathbf{S}_0(p) &\stackrel{\text{def}}{=} (w)(\bar{p}w(\bar{w}s_0 \mid \mathbf{S}_0(p))) &(1)\\
\mathbf{S}_1(x,\ p) &\stackrel{\text{def}}{=} (w)(\bar{p}w(\bar{w}s_1x \mid \mathbf{S}_1(x,p))) &(2)\\
\mathbf{S}_2(x,\ y,\ p) &\stackrel{\text{def}}{=} (w)(\bar{p}w(\bar{w}s_2xy \mid \mathbf{S}_2(x,\ y,\ p))) &(3)\\
\mathbf{K}_0(p) &\stackrel{\text{def}}{=} (w)(\bar{p}w(\bar{w}k_0x \mid \mathbf{K}_0(p))) &(4)\\
\mathbf{K}_1(x,\ p) &\stackrel{\text{def}}{=} (w)(\bar{p}w(\bar{w}k_1x \mid \mathbf{K}_1(x,p))) &(5)\\
\mathbf{I}_0(p) &\stackrel{\text{def}}{=} (w)(\bar{p}w(\bar{w}i_0x \mid \mathbf{I}_0(p))) &(6)\\
\mathbf{I}_1(x,\ p) &\stackrel{\text{def}}{=} (w)(\bar{p}w(\bar{w}i_1x \mid \mathbf{I}_1(x,p))) &(7)
\end{aligned}
$$

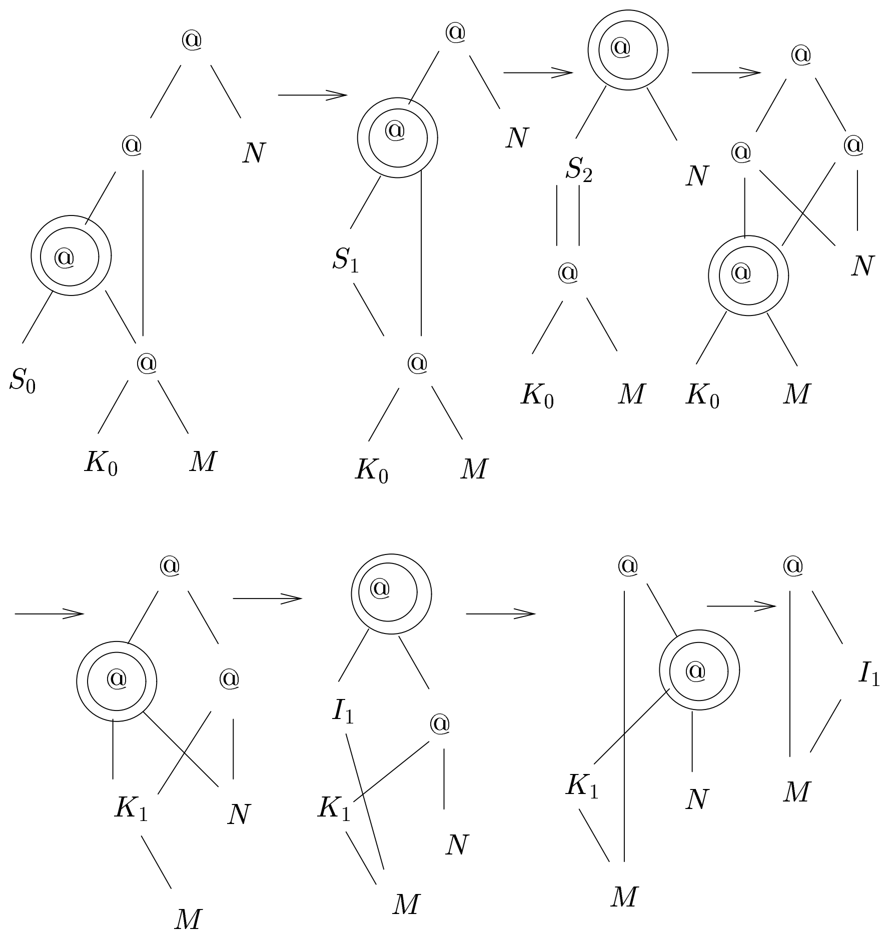
Aus den Definitionen geht hervor, daß jeder Agent erst seine Identität und dann die Namen seiner Kinder ausgibt. In der Synchronisierung mit den anderen Agenten, d.h. in einem Kompositionsagenten, stellt sich der Agent wiederher, wenn er die Information(en) ausgegeben hat. w ist ein Prozeß, der die atomaren Aktionen zu einer molekularen Aktion umbildet. Damit stellt er sicher, daß die Informationen eines Kombinators nicht verloren gehen.

Die Definition des Applikationsknotens lautet:

$$
\begin{aligned}
@(x,y,p) \stackrel{\text{def}}{=} x(w).w : [&s_0 \Rightarrow \mathbf{S}_1(y,\ p),\\
&s_1(x_1) \Rightarrow \mathbf{S}_2(x_1,\ y,\ p),\\
&s_2(x_1)(x_2) \Rightarrow (p_1)(p_2)(@(x_1,\ y,\ p_1)\\
&\qquad\qquad \mid (@(x_2,\ y,\ p_2)\\
&\qquad\qquad \mid (@(p_1,\ p_2,\ p)),\\
&k_0 \Rightarrow \mathbf{K}_1(y,\ p),\\
&k_1(x_1) \Rightarrow \mathbf{I}_1(x_1,\ p),\\
&I_0 \Rightarrow \mathbf{I}_1(y,\ p),\\
&I_1(x_1) \Rightarrow @(x_1,\ y,\ p)]
\end{aligned}
$$



Regel für die Reduktion des Kombinatorgraphen



Graphenreduktion für den Ausdruck **S** (**K** M)(**K** M) N

Beispiel: Reduktion von $\mathbf{S}_2$

$$
\begin{aligned}
& (p')(\mathbf{S}_2(x,y,p') \mid @(p',z,p)) \mid M(x) \mid N(y) \mid P(z)) \\
\equiv\ & (p')((w)(\bar{p'}w.(\bar{w}s_2xy \mid \mathbf{S}_2(x,\ y,\ p'))) \mid p'(w).w : [...]) \mid M(x) \mid N(y) \mid P(z) \qquad (1) \\
=\ & \tau.(p')(w)((\bar{w}s_2xy \mid \mathbf{S}_2(x,\ y,\ p')) \mid w : [...]) \mid M(x) \mid N(y) \mid P(z) \qquad (2) \\
=\ & \tau.\tau.\tau.\tau.(p')(w)((\mathbf{0} \mid \mathbf{S}_2(x,\ y,\ p')) \mid (p_1)(p_2)(@(x,\ z,\ p_1) \mid @(y,\ z,\ p_2) \mid @(p_1,\ p_2,\ p)) \qquad (3) \\
& \mid M(x) \mid N(y) \mid P(z)
\end{aligned}
$$

Im Schritt (1) werden die Defintionen der Agenten $\mathbf{S}_2(x,y,p')$ und $@(p',z,p)$ genutzt. Der Schritt (2) ergibt sich aus der gleichzeitigen Durchführung der Aktionen $\bar{p'}w$ und $p'(w)$ im Schritt (1). Hier wird w auf der einer Seite ausgegeben und von der anderen Seite empfangen. Das führt dazu, daß w als die Verbindung zwischen beiden Agenten besteht. Im Schritt (2) werden 3 Paare von Aktionen durchgeführt: Das erste Paar besteht aus der Ausgabe von s_2 auf einer Seite und Empfang von s_2 auf der anderen Seite durch den Kanal w. Dasgleiche geschieht mit dem zweiten Paar; also Ausgabe von x auf einer Seite und Empfang von x auf der anderen Seite durch w. Letztes Paar von Aktionen sendet y durch den Kanal w.

4.5 Übersetzung des λ-Kalküls

Durch die Behandlung von der Reduktion des Kombinatorgraphen konnte man diese Problematik umgehen, da es eine Übersetzung von λ-Kalkül in die Reduktion des kombinatorgraphen gibt. Es gibt aber eine direkte Übersetzung von λ-Kalkül in π-Kalkül. Als Motivation für diese Übersetzung kann man die Potentiale des π-Kalküls für die Formalisierung des Parallelismus nennen. Das kann man damit begründen, daß im π-Kalkül die Namen - dazu gehören also die Kanäle zwischen den Prozessen - im Vordergrund stehen, während im λ-Kalkül die Funktionen diese Rolle übernehemen[11].
Seien im folgenden $M, N, ... \in \Lambda$. (Λ ist eine Menge von λ-Ausdrücken.), $\mathcal{V}$ eine unendliche Menge, $x, y, z \in \mathcal{V}$, $\mathcal{V} \subseteq \mathcal{N}$, und $u, v, w \in \mathcal{N} - \mathcal{V}$ Wir nehmen an, daß die Termreduktion des hier zu übersetzenden λ-Kalküls die sognannte Lazy-Reduktion[12] ist.

Sie ist die kleinste Relation $\longrightarrow$ auf Λ, die folgendes erfüllt:

$$(\lambda x M)N \longrightarrow M\{N/x\}$$

$$\text{Wenn } M \ \longrightarrow\ M' \text{ dann } MN \ \longrightarrow\ M'N$$

Sie ist die Normalorders-Reduktion und ist deterministisch, denn jeder Term M, der noch weiter reduziert werden kann, ist von der Form: $M_0M_1 \cdots M_m (m \geq 0)$, wobei M_0 ein λ-Term ist. $M \longrightarrow M'$; kann genau dann gelten, wenn M_0 von der Form $\lambda x N$ ist. M' ist dann $N\{M_1/x\}M_2 \cdots M_m$. Daraus folgt, daß es für jeden Term M einen irreduziblen Term M' gibt, der von der Form $\lambda x N$ oder $xN_1 \cdots N_n (n \geq 0)$ ist.

Die Syntax vom λ-Kalkül ist gegeben durch:

$$M \ ::=\ x \mid (\lambda x M) \mid (MN)$$

[11] Für mehr Information siehe [13]

[12] Für mehr Informationen s.[1].

wobei bei fehlender Klammerung die Linksassoziativität bei Applikationen vorausgesetzt ist.
Wir nehmen an, daß in jedem Unterterm jede freie Variable x nur einmal vorkommen darf. Es handelt sich dabei um einen *linearen* λ-Kalkül. Da in diesem Kalkül jede Reduktion terminiert, wird der Y-Kombinator ausgenommen.
Übersetzung: Sei $[[M]] : Name \rightarrow Agent$ eine Abbildung. $[[M]]u$ ist ein Agent, bei dem u ein Zeiger ist, der auf die Argumente des Termes M zeigt.

$$\begin{aligned} [[\lambda x M]]u &\stackrel{\text{def}}{=} u(x)(v).[[M]]v && (25)\\ [[x]]u &\stackrel{\text{def}}{=} \bar{x}u && (26)\\ [[MN]]u &\stackrel{\text{def}}{=} (v)([[M]]v \mid (x)\bar{v}xu.x(w).[[N]](w)) && (27)\\ & \qquad\qquad x \text{ kommt nicht frei in } N \text{ vor} \end{aligned}$$

Die Lazy-Reduktion-Eigenschaft kann man in (27) sehen. Das Präfix $\bar{v}xu$ wird aktiviert, wenn M in die Form $\lambda x M'$ gebracht wird. Das Präfix $x(w)$ wird dann aktiviert, wenn M' den Term N durch x aufruft.

Beispiel: Es ist leicht zu erkennen, daß die folgende syntaktische Gleichheit stimmt:

$$[[\lambda xx]]v \equiv v(x)(w).\bar{x}w$$

Wir übersetzen den Term $(\lambda xx)N$:

$$\begin{aligned} [[(\lambda xx)N]]u &\equiv (v)([[\lambda xx]]v \mid (x)\bar{v}xu.x(w).[[N]]w) && (1)\\ &\equiv (v)(v(x)(w).\bar{x}w \mid (x)\bar{v}xu.x(w).[[N]w) && (2)\\ &= \tau.(v)(x)(v(w).\bar{x}w \mid \bar{v}u.x(w).[[N]]w) && (3)\\ &= \tau.\tau.(v)(x)(\bar{x}u \mid x(w).[[N]]w) && (4)\\ &= \tau.\tau.\tau.(v)(x) \mid \mathbf{0} \mid [[N]]u) && (5)\\ &= \tau.\tau.\tau.[[N]]u && (6) \end{aligned}$$

In den Schritten (1) und (2) sind die Definition der Agenten genutzt. Im Schritt (3) wird der Name x von einer Seite über $\bar{v}$ ausgegeben und von der anderen Seite empfangen. Das führt dazu, daß x zu einer privaten Verbindung zwischen den Agenten wird. Im 4. Schritt gibt der gebundene Name $\bar{v}$ den freien Namen u aus, und auf der anderen Seite wird er Empfangen. Der Zeiger w zeigt jetzt auf u; bzw. die Stelle w wird mit u überschrieben. Im 5. Schritt passiert folgendes: $\bar{x}$ liefert den Namen u, und auf der anderen Seite wird die Stelle w mit u überschrieben. Läßt man den **0**-Agent weg, erhält man die Gleichung (6).

Um den vollen λ-Kalkül - also λ-Kalkül mit Rekursion - zu übersetzen, benötigt man Replikationen des rekursiv aufgerufenen Agenten. Dazu definiert man bei einer Präfixaktion α den Fixpunktoperator:

$$\alpha * P \stackrel{\text{def}}{=} \mathbf{fix}X(\alpha.(P \mid X))$$

wobei $\mathbf{fix}XE$ für verschiedene Lösungen der Agentengleichung $X = E$, wobei X in E wieder vorkommt, steht. Mit diese Definition kann man jetzt einen Replikator folgendermaßen definieren:

$$\alpha * P = \alpha.(P \mid \alpha * P)$$

Für die Übersetzung des vollen λ-Kalküls gelten die Definitionen (25) und (26), während die Definition (27) mit dem Replikator definiert werden muß:

$$[[MN]]u \stackrel{\text{def}}{=} (v)([[M]]v \mid (x)\bar{v}xu.x(w) * [[N]](w) \qquad (28)$$
$$x \text{ kommt nicht frei in } N \text{ vor}$$

N kann jetzt mehr als ein Mal durch M aufgerufen werden. Jeder Aufruf erzeugt eine neue Replikation von N mit einer neuen Argumentliste.

5 Starke Ähnlichkeit und starke Äquivalenz

5.1 Motivation

Das Ziel ist, festzustellen, unter welchen Umständen ein Agent einen anderen simulieren kann, und welche Kriterien für die Äquivalenz der Agenten zugrunde gelegt werden. Beispielsweise führen zwei Agenten eine gleiche externe Aktion durch, wobei der eine einen Zustand erreicht, welchen der andere mit einer zusätzlichen internen Aktion erreichen kann. Die Frage lautet, ob diese beiden Agenten als äquivalent bezeichnet werden können, obwohl sie sich bezüglich der externen Aktionen unterseiden. Man kann sich hier zwei endliche Automaten vorstellen, wobei der eine deterministisch und der andere nichtdeterministisch rechnet, und beide die gleiche Sprache akzeptieren. Daß die beiden endlichen Automaten äquivalent sind, ist bereits bekannt.
Ist die Simulation eines Agenten durch den anderen von Intresse, erfordern die gebundenen Namen und gebundenen Aktionen der Agenten besondere Aufmerksamkeit. Wir betrachten zwei ähnliche Agenten:

$$P \equiv x(y).R \qquad \text{und} \qquad Q \equiv (z)x(y).R$$

wobei $z \notin \text{fn}(R, x)$. Es ist offensichtlich, daß P den Eingabeübergang $\xrightarrow{x(z)}$ hat, den Q nicht genau simulieren kann. Der Agent Q kann aber ihn mit $\xrightarrow{x(w)}$ simulieren. Die Unwichtigkeit dieses Unterschiedes für die Simulation kann man dadurch erklären daß für einen Beobachter von außen die Identität der gebunden Namen in Q und P unsichtbar bleibt. Ein gebundener Name ist eine Referenz, die auf irgendeine Adresse bei den Agenten zeigt. Die unterschiedliche Namenauswahl für den gebunden Namen bei den Agenten ist also nicht ein Hindernis dafür, daß zwei Agenten sich ähnlich verhalten. D.h. zwei Agenten simulieren sich gegenseitig, obwohl ihre gebundenen Namen nicht identisch sind.
Anderes ist es jedoch, wenn die freien Namen eines Agenten bei einer Aktion substituiert weden müssen. Dieser Fall führt uns zur starken Äquivalenz.

5.2 Starke Ähnlichkeit

Eine binäre Relation $\mathcal{S}$ in Bezug auf die Menge der Agenten ist (starke) Simulation, wenn sie die Forderungen in der Tabelle unten erfüllt:

> $\mathcal{S}$ ist eine Simulation, wenn aus $P\mathcal{S}Q$ folgt, daß
>
> 1. wenn $P \xrightarrow{\alpha} P'$ und α eine freie Aktion ist,
>   dann gilt es für einige Q', daß $Q \xrightarrow{\alpha} Q'$ und $P'\mathcal{S}Q'$
>
> 2. wenn $P \xrightarrow{x(y)} P'$ und $y \notin \mathrm{n}(P, Q)$,
>   dann gilt es für einige Q', daß $Q \xrightarrow{x(y)} Q'$ und für alle w,
> $P'\{w/y\}\mathcal{S}Q'\{w/y\}$
>
> 3. wenn $P \xrightarrow{\bar{x}(y)} P'$ und $y \notin \mathrm{n}(P, Q)$,
>   dann gilt es für einige Q', daß $Q \xrightarrow{\bar{x}(y)} Q'$ und $P'\mathcal{S}Q'$

Definition der (starken) Simulation.

Die Relation $\mathcal{S}$ heißt (starke) Bisimulation, wenn ihere Umkehrung $\mathcal{S}^{-1}$ eine Simulation ist. Die Relation der starken Ähnlichkeit ($\dot\sim$) zwischen den Agenten P und Q entsteht - geschrieben: $P \dot\sim Q$ - genau dann, wenn eine Bisimulation $\mathcal{S}$ existiert, so daß $P\mathcal{S}Q$.

Beispiele: Es gelten folgende Abkürzungen:

$$\begin{array}{lll} \bar{x} & \textit{statt} & \bar{x}v.\mathbf{0} \\ y & \textit{statt} & y(u).\mathbf{0} \end{array}$$

1. Unter der gegebenen Bedigungen besteht zwischen den folgenden beiden Agenten die Relation der starken Ähnlichkeit:

   $$\bar{x} \mid y \;\dot\sim\; \bar{x}.y + y.\bar{x} \quad \text{wenn } x \neq y \text{ und } u \neq v \qquad (1)$$

2. Wenn aber $x = y$ gewählt wird, dann gilt:

   $$\bar{x} \mid x \;\dot{\not\sim}\; \bar{x}.x + x.\bar{x} \qquad (2)$$

   Man sieht, daß auf der rechten Seite ein τ-Übergang fehlt.

3. Assoziert man eine gebundene Ausgabeaktion mit den beiden Agenten, bleibt die Bisimulationseigenschaft erhalten:

   $$(y)\bar{z}y.(\bar{x}|y) \;\dot\sim\; (y)\bar{z}y.(\bar{x}.y + y.\bar{x})$$

   Zur Erläuterung muß man sagen, daß eine Verwechslung der freien Namen x, der auf den beiden Seiten erscheint, mit dem gebundenen Namen y ausgeschlossen ist.

4. Im Gegensatz bleibt die Relation der starken Ähnlichkeit bei der Assoziation einer gebundenen Eingabeaktion mit den ähnlichen Agenten nicht erhalten:

$$z(y).(\bar{x}|y) \ \dot{\not\sim}\ z(y).(\bar{x}.y + y.\bar{x})$$

Der Grund liegt darin, daß y mit x instatiert werden kann, wodurch die ähnlichkeit der Agenten verloren geht. Es sollte aber nach dem Definitionteil 2 in der Tabelle die starke Ähnlichkeit bei der beliebigen Instantierung des gebundenen Name y gelten.

5.3 Starke Äquivalenz und Unterschiede

Es wurde bereits gezeigt, daß die starke ähnlichkeit bei der Substituierung der freien Namen nicht mehr gilt. Namen verhalten sich bei dieser Eigenschaft wie Konstanten, die nicht variiert werden können. Deshalb nennt man die starke Ähnlichkeit manchmal die (starke) Grundäquivalenz. Diesen Abschnitt beschreiben wir in Form einige wichtige Definitionen:

- Eine Substitution ist eine Funktion σ mit: $\sigma : N \to N$, die fast immer Identität ($x\sigma = x$) ist. Eine andere Schreibweise für $x_i\sigma = y_i$ für alle i mit $1 \leq i \leq n$ ist: $\{y_1/x_1, ..., y_n/x_n\}$ oder $\{\tilde{y}/\tilde{x}\}$.

- $P\sigma$ bezeichnet einen Agenten, der beim gleichzeitigen Ersetzen $z\sigma$ für jedes freie Vorkommen von z in P für jedes z aus P entsteht. Hierbei werden die gebundenen Namen umbenannt, um ihre Gefangenschaft zu verhindern. Interpretiert man $\equiv$ als syntaktische Gleichheit, gilt dann :

$$(x(y).P)\sigma \equiv x\sigma(y').P\{y'/y\}\sigma \quad \text{wobei } y' \notin \mathrm{fn}((y)P, P\sigma) \ \text{ und } \ y'\sigma = y'$$

$$((y).P)\sigma \equiv (y')P\{y'/y\}\sigma \quad \text{wobei } y' \notin \mathrm{fn}((y)P, P\sigma) \ \text{ und } y'\sigma = y'$$

- Das Symbol $\equiv_\alpha$ ist wie *Alpha-Konversion* im λ-Kalkül. D.h. Wenn $P \equiv_\alpha Q$, dann sind P und Q *Alpha-Konvertibel*, sie sind also bis auf die Umbenennung der gebundenen Namen gleich.

- P und Q sind (stark) äquivalent ($P \sim Q$), wenn $P\sigma \dot{\sim} Q\sigma$ für alle Substitutionen σ gilt. D.h.: statt (1) kann man schreiben:

$$\bar{x} \mid y \ \sim \ \bar{x}.y \ + \ y.\bar{x} \ + \ [x = y]\tau \qquad (4)$$

Im Gegenteil zu starken Äquivalenz verhalten sich die Namen hier wie Variablen. Die Substitution müs also für alle Namen gelten.

- Ein *Unterschied* ist eine irreflexive symetrische Relation zwischen Namen. Sei D die Menge der Unterschiede. Man sagt: eine Substitution respektiert einen Unterschied, wenn gilt: $\forall(x, y) \in D,\ x\sigma \neq y\sigma$

- P und Q heißen (starke) D-äquivalent ($P{\sim_D}Q$), wenn für alle Substitutionen, die D respektieren, gilt: $P\sigma \dot{\sim} Q\sigma$.
Daraus folgt: wenn $D \subseteq D' \implies P{\sim_{D'}}Q$. Man kann (4) schreiben:

$$\bar{x} \mid y \ \sim_{\{x,y\}} \ \bar{x}.y \ + \ y.\bar{x}$$

Auf Unterschiede basiert kann man die starke Ähnlichkeit und starke Äquivalenz so definieren:

$$\dot{\sim} = \sim_{\mathbf{N}} \quad und \quad \sim = \sim_{\emptyset}$$

6 Algebraische Theorie

In diesem Abschnitt fassen wir die Axiome der starken Ähnlichkeitrelation, starken Äquivalenzrelation zusammen. Diese Axiome sind von den Eigenschaften der bereits genannten Relationen herausgefiltert und bewiesen worden. Den Beweis dieser Axiome findet man in [15]. Diese Axiome sind für eine endliche Menge von Agenten definiert. Da die Relation $\dot{\sim}$ nicht rekursiv aufzählbar ist, ist sie ungültig, wenn die Menge der Agenten unendlich ist.
Man axiomatisiert zuerst die Relation der Ähnlichkeit, da viele diese Axiome in der Äquivalenzrelation gelten.

6.1 Die Axiome der starken Ähnlichkeit

Damit es vereinfacht wird, schreiben wir $=$ statt $\dot{\sim}$. $\equiv$ stellt die syntaktische Gleichheit dar.
Die **SEG**(strong ground equivalence)-Theorie umfaßt folgende Axiome und Schlußfolgerungen:

Alpha- Konversion: **A** $vonP \equiv Q$ folgt $P = Q$
Kongruenz:
C0 von $P = Q$ folgt

$$\tau.P = \tau.Q \qquad \bar{x}y.P = \bar{x}y.Q$$
$$P + R = Q + R \qquad P \mid R = Q + R$$
$$(x)P = (x)Q \qquad [x = y]P = [x = y]Q$$

C1 von $P\{z/y\} = Q\{z/y\}$, für alle Namen $z \in \text{fn}(P, Q, y)$, folgt

$$x(y).P = x(y).Q$$

Summation:

S0 $P + \mathbf{0} = P$
S1 $P + P = P$
S2 $P + Q = Q + P$
S3 $P + (Q + R) = (P + Q) + R$

Restriktion:

R0 $(x)P = P$ wenn $x \notin \text{fn}(P)$
R1 $(x)(y)P = (y)(x)P$
R2 $(x)(P + Q) = (x)P + (x)Q$
R3 $(x)\alpha.P = \alpha.(x)P$ wenn $x \notin \text{n}(\alpha)$
R4 $(x)\alpha.P = \mathbf{0}$ wenn x das Subjekt von α ist

Match:

M0 $[x = y]P = \mathbf{0}$ wenn $x \neq y$
M1 $[x = x]P = P$

Expansion:

E Seien $P \equiv \sum_i \alpha_i.P_i$ und $Q \equiv \sum_j \beta_j.Q_j$, wobei α_i(bzw. β_j) bindet keinen freien Namen in Q (bzw. P), dann folgt

$$P \mid Q = \sum_i \alpha_i.(P_i|Q) + \sum_j \beta_j.(P|Q_j) + \sum_{\alpha_i \; comp \; \beta_j} \tau.R_{ij}$$

wobei die Relation α_i *comp* β_j(α_i *complements* β_j) in folgenden 4 Fällen erfüllt ist:
1.α_i ist $\bar{x}u$ und β_j ist $x(v)$; dann R_{ij} ist $P_i \mid Q_j\{u/v\}$.
2. α_i ist $\bar{x}(u)$ und β_j ist $x(v)$; dann R_{ij} ist $(w)(P_i\{w/u\} \mid Q_j\{w/v\})$, wobei w ist in $(u)P_i$ oder in $(v)Q_j$ nicht frei.
3. α_i ist $x(v)$ und β_j ist $\bar{x}u$; dann R_{ij} ist $P_i\{u/v\} \mid Q_j$.
4. α_i ist $x(v)$ und β_j ist $\bar{x}(u)$; dann R_{ij} ist $(w)(P_i\{w/v\} \mid Q_j\{w/u\})$, wobei w ist in $(v)P_i$ oder in $(u)Q_j$ nicht frei.

Bezeichner[13] **I** aus $A(\tilde{x}) \stackrel{\text{def}}{=} P$ folgt $A(\tilde{x}) = P\{\tilde{y}/\tilde{x}\}$

Wenn wir diese Axiome genauer anschauen, erregen einige Punkte unsere Aufmerksamkeit:

1. **M0** und **M1** überschatten die letzte Klausel von **C0**, nämlich

   $$\text{Aus } P \equiv Q \text{ folgt } [x=y]P = [x=y]Q$$

   Diese Klausel brauchen wir für die Axiome der Äquivalezrelation, bei der die Regel **M0** nicht gilt.

2. Die Regel **C1** kann nicht für alle Substitutierung von y durch z gelten; also wir können nicht schreiben:

   $$\text{Aus } P \equiv Q \text{ folgt } x(y).P = x(y).Q$$

   Als Gegenbeispiel kann man das Beispiel 4 im Abschnitt 5.2 nennen. y ist ein Platzhalter für einen ankommenden Namen, der auch x sein kann.

3. Im Bezug auf die Regeln **C0**, **M0** und **M1** kann man alle Machoperatoren eines Agenten weglassen, wenn der Agent keine Eingabepräfix enthält. Bei der Eigabepräfix $x(y)$ kann die Machform $[y=z]$ nicht vom Agent $x(y).[y=z]P$ weggelassen werden. Das kann man wie der letzte Punkt begründen, nähmlich, wenn der ankommende Name z ist, ist dieser Agent wie **0** Aggent.

4. In der Regel **R3** kann das Präfix α auch das gebundene Ausgabepräfix $\bar{z}(y)$ sein.

5. In der Regel **R3** sind die Teilregeln 2 und 4 kritisch, da ein private Name - also eine Verbindug - zwischen dem Sender und dem Empfanger Übertragen wird.

[13] Englisch: Identifier

6.2 Die Axiome der starken Äquivalenz

Interpretiert man = als die Äquivalenzrelation, gelten alle Regeln der **SGE** bis auf die Regeln **M0** und **E**. **M0** gilt nicht, weil wenn wir eine Eingabepräfix $x(y)$ an den Agenten $[x = y]P$ hängen und dabei y mit x substituieren, haben wir die Regel **M1**. Währed muß es für starken Äquivalez gelten, daß bei der Substitutierung aller Namen beiden Agenten äquivalent sind, ist in dem Fall der Agent $x(y)[x = y]P$ äquivalet zu dem P und nicht dem **0** Agent.
Man kann aus den Gleichungen (1) und (2) aus dem Abschnitt 5.2 erlesen, warum die Regel **E** nicht gilt. Wenn wir die Regel **E** mit der neuen Regel **E'** ersetzen, wird diese Regel für die Äquivalenzrelation gelten:

E' Seien $P \equiv \sum_i \alpha_i.P_i$ und $Q \equiv \sum_j \beta_j.Q_j$, wobei α_i(bzw. β_j) bindet keinen freien Namen in Q (bzw. P), dann folgt

$$P \mid Q = \sum_i \alpha_i.(P_i|Q) + \sum_j \beta_j.(P|Q_j) + \sum_{\alpha_i \; opp \; \beta_j} [x_i = y_j]\tau.R_{ij}$$

wobei die Relation α_i *opp* β_j(α_i *lehnt* β_j *ab*)
in folgenden 4 Fällen erfüllt ist:
1.α_i ist $\bar{x_i}u$ und β_j ist $y_j(v)$; dann R_{ij} ist $P_i \mid Q_j\{u/v\}$.
2. α_i ist $\bar{x_i}(u)$ und β_j ist $y_j(v)$; dann R_{ij} ist $(w)(P_i\{w/u\} \mid Q_j\{w/v\})$, wobei w ist in $(u)P_i$ oder in $(v)Q_j$ nicht frei.
3. α_i ist $x_i(v)$ und β_j ist $\bar{y_j}u$; dann R_{ij} ist $P_i\{u/v\} \mid Q_j$.
4. α_i ist $x - i(v)$ und β_j ist $\bar{y_j}(u)$; dann R_{ij} ist $(w)(P_i\{w/v\} \mid Q_j\{w/u\})$, wobei w ist in $(v)P_i$ oder in $(u)Q_j$ nicht frei.

Die Regel **C1** von **SGE** wird in etwas stärekeren Form gelten, weil die Substituierung aller Namen erlaubt ist:
C1$'$ aus $P = Q$ folgt $x(y).P = x(y).Q$
Zusammengefaßt kann man sagen, daß die Axiome

SGE - {C1, M0, E} $\cup$ **{C1', E'}**

für die Äquivalenzrelation gelten, wenn man die = als die starke Äquivalenz ($\equiv$) interpretiert. Es gelten zusätzlich die folgenden Regeln:
1. $P \mid \mathbf{0} \equiv P$
2. $P \mid Q \equiv P \mid Q$
3. $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
4. $(x)(P \mid Q) \equiv P \mid (x)Q$ wenn $x \notin \mathrm{fn}(P)$

7 Zusammenfassung

Der π-Kalkül ist die Prozessalgebra, die die dynamischen Prozesse beschreibt, die mit einander kommunizieren und dabei ihren Zustand ändern. Wir haben die Syntax des Kalküls vorgestellt und die Eigenschaften des Kalküs herauskristalisiert. Dabei wurde in Form der geeigneten Beispiele gezeigt, wenn die Konstanten, Variablen und

Kommunikationskanäle im Kalkül nicht unterschieden werden, auf welche Variationen der Bindungsbereich man Acht geben muß. Wir haben den π-kalkül, während er die Datenstruktoren, die Reduktion der Kombinatorgraphen und den λ-Kalkül übersetzt hat, beobachtet. Anschliessend haben wir die Agenten im Vergleich gezogen, wobei die starke Ähnlichkeit und die stake Äquivalez der Agenten erläutert wurde. Die algebraische Regel zeigen die Ausdruckfähigkeit des π-Kalküls. Wir betonen, daß der π-Kalkül ein vollständiges mathematisches Modell ist. Zur Begründung genügt es zu sagen, das der π-Kalkül den λ-Kalkül, desen vollständigkeit bereits bekannt ist, simuliert.

8 Literaturverzeichnis

Literatur

[1] S. Abramsky. *The lazy Lambda Calculus,*. D. Turner, Addison-Wesley, 1988. to appear in **Declarative Programming**.

[2] E. Astesiano and E. Zucca. Parametric channels via label experessions in ccs. *Journal of Theor. Comp. Science*, Vol 33:45–64, 1984.

[3] J.A. Bergstra and J-W. Klop. Algebra of communicating prosses with abstraction. *Journal of Theor. Comp. Science*, Vol 33:77–121, 1985.

[4] G. Boudol. Towards a lambda-calculus for concurrent and communicating systems. *INRIA Sophia-Antipolis,private communication*, 1988.

[5] W.D. Clinger. Foundations of actor semantics. *AI-TR-633, MIT Artifical Inteligence Laboratory*, 1981.

[6] U. Engberg and M. Nielsen. A calculus of cammunicating systems with label-passing. *Report DAIMI PB-208, Computer Science Department, University of Aarhus*, 1986.

[7] C.A.R. Hoare. ***Cammunicating Sequential Processes***. Prantice Hall, 1985.

[8] S.Leinwand J.A.Goguen and T.Winkler. Cell and ensemble architecture for the rewrite rule machine. *Proc. International Conference on Fifth Generation Computing Systems, ICOT*, pages 869–878, 1988.

[9] R. Milner. ***Communication and Concurrency***. Prantice Hall, 1989.

[10] R. Miner. Funktions as processes. *Research Report 1154, INRIA*, 1990. Also in abbreviated form in Proc. ICALP '90.

[11] F. Nielson. The typed λ - calculus with first-class processes. *Report ID-TR:1988-43, Inst. for Datateknik, Teknishe Hojskole, Lyngby, Denmark*, 1988.

[12] C-H.L. Ong. Fully abstract models of the lazy lambda calculus. *Proc 29 th Symposium on Foundations of Computer Science*, pages 68–376, 1989.

[13] Gerald K. Ostheimer and Antony J.T.Davie. Modelling parallel graph reduktion in the pi-calculus. *University of St Andrews Department of Mathematical and Computational Sciences.*

[14] W. Reisig. ***Petri Netzs***. Springer Verlag, 1983. EATCS Monographs on Theoretical Computer Science, ed. W.Brrauer, G.Rozenberg, A.Salomaa.

[15] J.G.Parrow R.Miner and D.J.Walker. A calculus of mobile procvesses part i. *Report ECS-LFCS-89-86, Laboratory for Foundations of Computer Science, Computer Science Depatment, Edinburgh University*, 1989.

[16] J.G.Parrow R.Miner and D.J.Walker. A calculus of mobile procvesses part ii. *Report ECS-LFCS-89-86 Laboratory for Foundations of Computer Science, Computer Science Depatment, Edinburgh University*, 1989.

[17] M.R. Sleep and C-H.L. Kennaway, J.R.Ong. The zero assignment parallel processor (zapp) project. *in The Distributed Computing Systems Programme, ed. D.A.Duce, Peter Peregrinus Ltd.*, pages 250–269, 1984.

[18] B. Thomsen. A calculus of higher-order communicating systems. *Proc POPL Conference*, 1989.