Fachbereich 12 Informatik und Mathematik
Institut für Informatik

Diplomarbeit

Ein neuer Ansatz zu einer kontextuellen Semantik im Pi-Kalkül: Untersuchung, Analyse und Vergleich

Matthias Pfaff
Studiengang: Informatik
Matrikelnummer: 2678061

Frankfurt am Main
27. Oktober 2010

eingereicht bei
Prof. Dr. Manfred Schmidt-Schauß
Künstliche Intelligenz / Softwaretechnologie

Gesetzt am 27. Oktober 2010 um 15:12 Uhr mit LaTeX.

Danksagung

Ich danke allen Freunden und Bekannten, die mich während der Entstehung dieser Arbeit begleitet und unterstützt haben.

Ganz besonders danke ich meinen Eltern, die es mir überhaupt erst ermöglichten zu studieren und mich dabei immer unterstützt haben.

Außerdem danke ich Dr. David Sabel und Prof. Dr. Manfred Schmidt-Schauß für Ihre hervorragende Betreuung und Ihre unzähligen hilfreichen Anregungen.

Matthias Pfaff

Erklärung gemäß DPO §11 Abs. 1

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internetquellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen und Abbildungen–, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Dietzenbach, den 27. Oktober 2010

Matthias Pfaff

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACM	Association for Computing Machinery
BPML	Business Process Modeling Language
bzw.	beziehungsweise
CCS	Calculus of Communicating Systems
CSP	Communicating Sequential Processes
d.h.	das heißt
engl.	englisch
evtl.	eventuell
ex.	existiert
g.d.w.	genau dann, wenn
geschr.	geschrieben
ggf.	gegebenenfalls
i.Allg.	im Allgemeinen
LCF	Logic of Computable Functions
LTS	Labelled Transitions System
ML	Meta Language
sog.	so genannt(er/e/es)
SOS	strukturell operationelle Semantik
vgl.	vergleiche
YAWL	Yet Another Workflow Language
z.B.	zum Beispiel

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

1 Einleitung

1.1 Motivation

Computer haben in relativ kurzer Zeit, angefangen bei Konrad Zuse und seinem als ersten Computer zu bezeichnenden Z3 [Roj00], eine rasante Entwicklung durchlaufen. Der anfänglich stationär, isoliert agierende Rechner, der sich auf fließbandartige Abarbeitung von Berechnungen beschränkte, findet sich nun in einer *vernetzten*, kontinuierlich ändernden Welt wieder, in der weder seine eigene Position im *Raum* fix sein muss noch er alleine in ihr agiert.

Aufgrund der immer neuen Einsatzgebiete heutiger Computer sind viele Aufgaben nicht mehr nur alleine mit einer einzelnen Recheneinheit (Entität) zu lösen, sondern verteilen sich auf eine Vielzahl. Vor diesem Hintergrund ist es mitunter nötig, dass einzelne Entitäten miteinander in *Kontakt* treten bzw. miteinander *kommunizieren* können. Beispielsweise gilt es in einem Mautsystem (vgl. Abbildung 1.1), Informationen über die einzelnen Standorte der zu erfassenden Objekte mitzuteilen [Mil99] und zu verarbeiten. Für Unternehmen wiederum ist es wichtig, einzelne Geschäftsprozesse zu koordinieren, um reibungslose Abläufe zu gewährleisten, was z.B. mithilfe passender Software realisierbar ist. In beiden Fällen müssen Informationen (Nachrichten) kommuniziert werden, um die jeweiligen Aufgaben (Berechnungen) zu lösen. Genau hier setzt der Pi-Kalkül an und bietet Methoden zur Modellierung, Analyse und Verifikation solcher Systeme.

Zur Beschreibung aufeinander folgender (sog. sequentieller) Berechnungen kennt die Informatik bereits viele Modelle wie etwa Turingmaschinen [Tur36] oder aber die Automatentheorie [Hop06], wie auch den Lambda-Kalkül [Chu32, Chu36, Chu41, Kle35]. Hinzu kam in den 1980er-Jahren die Möglichkeit, auch die Kommunikation zwischen mehreren sequentiellen Systemen (nebeneinander laufend) zu beschreiben: Tony Hoare entwickelte hierzu CSP (Communicating Sequential Processes) [Hoa85], Robin Milner CCS (Calculus of Communicatingsystems) [Mil82], ein ähnliches Modell. Beiden Modellen fehlte jedoch die Möglichkeit der Beschreibung sog. *mobiler* Prozesse, die sich durch dynamisches Auf- bzw. Abbauen der Verbindungen zwischen den einzelnen

Prozessen auszeichnet. Dabei kann als Nachrichteninhalt der Name des verwendeten Kommunikationskanals (Kanalname) versendet werden, womit sich die Topologie der Kommunikationsstruktur dynamisch verändern lässt. Genau diese Möglichkeit bietet der Pi-Kalkül nach Robin Milner, der als Nachfolger des von ihm ebenfalls erdachten CCS angesehen werden kann und so zu oben genannten Beispielen einen Lösungs- bzw. Modellierungsansatz bietet.

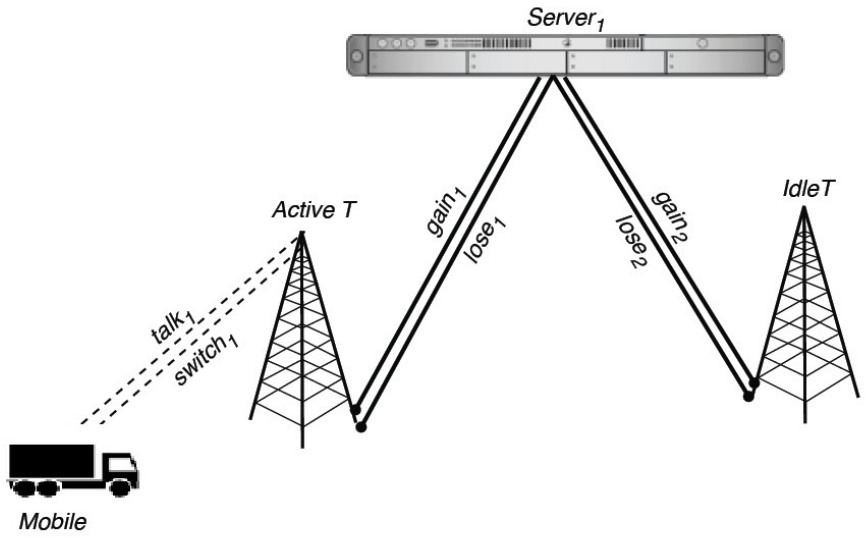


Abbildung 1.1: Mautsystem[1]

Der Pi-Kalkül erlaubt die Beschreibung von Prozessen, die sich in einer ständig verändernden Umgebung wiederfinden und darauf reagieren bzw. damit interagieren müssen (sog. reaktive Systeme) wie etwa heutige Unternehmen und deren Geschäftsprozesse (vgl. Abbildung 1.2) [Ove05]. Derlei dynamische Strukturen lassen sich mithilfe der Prozessalgebra des Pi-Kalküls abbilden und formal analysieren [Puh05, Puh07]. Die Aktualität dieses Themas lässt sich daran erkennen, dass der Pi-Kalkül als theoretisches Fundament der sog. Business Process Modeling Language (BPML) wie auch für XLANG von Microsoft dient [BPE02] und zur Beschreibung von Geschäftsprozessmodellen eingesetzt werden.
Frühere Ansätze zur formalen Beschreibung solcher Strukturen, wie beispielsweise mithilfe von Petrinetzen nach Carl Adam Petri [Pet62] als auch die hierauf aufbauenden Workflow-Netze [vdA98], sowie um Prozess-Pattern[2] erweitert die Yet Another Workflow Language (YAWL)[vdA02], boten nicht die Möglichkeit, die systeminhärente Kommunikation in allen Fassetten zu beschreiben. Sie bieten zwar die Möglichkeit, innerbetriebliche Business-Prozesse (also deren Kommunikation) zu beschreiben, eine

1 Darstellung angelehnt an Milner [Mil99].
2 Ein Prozess Pattern repräsentiert einen erprobten (Muster-)Prozess, der ein häufig auftretendes Problem löst [Cop96].

externe Kommunikation, wie sie etwa nötig ist, wenn zwei (oder mehr) Unternehmen miteinander kooperieren, lässt sich jedoch nicht formal unterscheiden. Petrinetze ermöglichen dies zwar, jedoch fehlt ihnen wiederum die Unterstützung für Process Pattern.

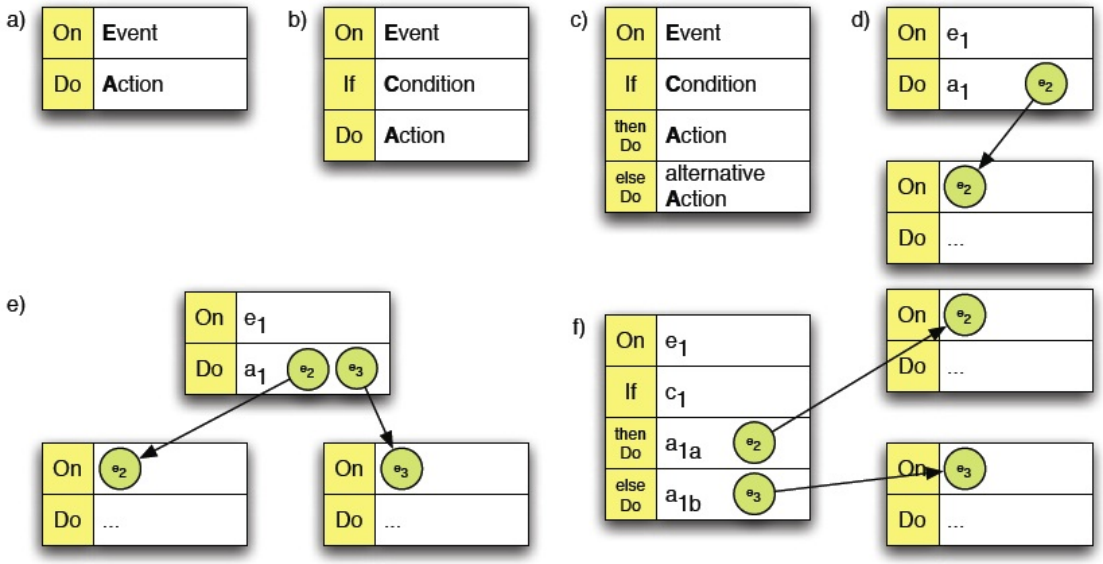


Abbildung 1.2: Eine EventAction (a), EventConditionAction (b) und EventConditionAlternativeAction (c) Darstellung von Business-Rules sowie sequientielle (d), parallele (e) und optionale (f) Kontrollabläufe [Puh05]

Selbst in der Biochemie kommt der Pi-Kalkül, wenn auch zumeist in einer stochastischen Variante, um biochemische Prozesse formal zu repräsentieren, zum Einsatz [Reg01]. Klassische Ansätze zur Modellierung von molekularbiologischen Systemen basieren größtenteils auf Differentialgleichungen, die die gemittelten Konzentrationen von auftretenden chemischen Elementen eines Modells kontinuierlich beschreiben [Lee03]. Da diese Art der Beschreibung eines biologischen Prozesses nicht in jedem Fall sinnvoll ist – wenn beispielsweise nur sehr wenige Moleküle an ablaufenden Prozessen beteiligt sind – so bedarf es zur Modellierung dieser Abläufe anderer Ansätze. Als Basis von weiterentwickelten Modellen hat sich hier der Pi-Kalkül nach Milner als sinnvoll erwiesen [Joh08].

Wie in den meisten Kalkülen ist auch im Pi-Kalkül eine Untersuchung des Programmverhaltens von Interesse. Die Semantik, also die formale Definition der Bedeutung von Befehlen, ist hier Ziel der näheren Untersuchung. Gemäß dieser (der Semantik) gilt es, nähere Schlüsse über das Verhalten von Programmfragmenten treffen zu können. Was sind korrekte Programmtransformationen besonders im Hinblick auf optimierte, abgeänderte Programmteile? Wann sind zwei Programme gleich? Um dies zu klären, wurden bereits die unterschiedlichsten semantischen Analysen in Bezug auf Verhaltensgleicheit durchgeführt. Genannt sei hier beispielsweise die Bisimulation, derzufolge zwei

Programme *bisimular* sind, wenn sie sich bei ihrer Auswertung stets gleich verhalten. Genauere Aussagen über das kontextuelle Verhalten von Programmen und Programmtransformationen wurden jedoch bisher nur unzureichend geklärt. Anders als bei der Bisimulation sind zwei Programme genau dann kontextuell gleich, wenn sie sich als Unterprogramm eines beliebigen anderen Programms (also in beliebigem Programmkontext) hinsichtlich ihres Verhaltens (bzgl. der Auswertung) nicht unterscheiden lassen. Genau an dieser Stelle setzt diese Arbeit mit einer genauere Untersuchung der kontextuellen Äquivalenz von Prozessen im Pi-Kalkül an.
Für weitergehendes Interesse an nebenläufiger Programmierung sei an dieser Stelle [Sab10a] als allgemeine Übersicht erwähnt.

1.2 Zielsetzung

Ziel der Arbeit ist es, einen neuen Ansatz zur kontextuellen Semantik im Pi-Kalkül zu untersuchen, zu analysieren und mit bereits bekannten Vorgehensweisen zu vergleichen. Ein erhofftes Ziel besteht darin mögliche Schlüsse über kontextuelles Verhalten, wie sie bereits im Lambda-Kalkül [SS07] gezogen wurden, auch auf den Pi-Kalkül anwenden zu können.

Als theoretische Grundlage dient hierbei der Pi-Kalkül nach [Mil99], der in beschränkterem Umfang Verwendung findet und, sofern sich Aussagen über das kontextuelle Verhalten von Programmtransformationen bzw. Programmen treffen lassen, bei Bedarf erweitert werden kann.
Im Rahmen dieser Diplomarbeit soll zunächst grundlegend überprüft werden, ob sinnvolle Aussagen in Bezug auf das kontextuelle Verhalten von Prozessen überhaupt möglich sind, d.h. die kontextuelle Gleichheit die erwünschten Eigenschaften zum Nachweis von Prozessgleichheiten beinhaltet und ob die Definition der Gleichheit von Prozessausdrücken im Pi-Kalkül über die kontextuelle Gleichheit grundsätzlich sinnvoll ist.

1.3 Überblick

Im Folgenden wird ein Überblick über die weiteren Kapitel dieser Diplomarbeit gegeben.

In *Kapitel 2* werden zunächst grundlegende Begriffe der Prozessalgebra erklärt sowie eine Aufstellung von unterschiedlichen Prozesstypen gegeben. Weiterführende Erläuterungen zu sequentiellen – also auch zu reaktiven Systeme und deren Eigenschaften – gehen der Beschreibung, *was* Kommunikation in solchen Systemen an sich bedeutet, voraus. Um die hieran anschließende Vorstellung des Vorläufers des Pi-Kalkül – den *Calculus of Communicating Systems* (kurz CCS) und dessen Semantik – besser zu verstehen, wird eine kleine Einführung in *logisches Schließen* gegeben. Anhand von Beispielen werden wichtige Eigenschaften von CCS näher erläutert, sodass Syntax und Semantik besser verstanden werden können. Eine erste allgemeine Einführung zu Prozessgleichheiten und

zwei Konzepte zu ihrem Nachweis – die Bisimulation und die kontextuelle Gleichheit – schließen Kapitel 2 ab.

Kapitel 3 beginnt mit einer allgemeinen Vorstellung des Pi-Kalküls und seiner Unterschiede zu CCS. Daran anschließend erfolgt die Definition der Syntax, angelehnt an [Mil99]. Hierüber hinaus werden tiefere prozessalgebraische Eigenschaften sowie Besonderheiten im Pi-Kalkül zunächst eingeführt als auch anhand von Beispielen näher erklärt. In Vorbereitung der Untersuchung von kontextuell äquivalenten Prozessen werden die strukturelle Kongruenz sowie die operationale Semantik im Pi-Kalkül definiert. Bevor genauere Untersuchungen der Prozessgleichheit im Pi-Kalkül erfolgen, wird auf den Unterschied des *synchronen* vom *asynchronen* Pi-Kalkül eingegangen.
Es wird die Frage beantwortet: „Wann sind Prozesse als kontextuell äquivalent anzusehen?“ Hierfür werden die notwendigen Voraussetzungen zunächst definiert, bevor im Anschluss einige Programmtransformationen auf ihre Korrektheit bzgl. der Erhaltung der kontextuellen Gleichheit geprüft wie auch kontextuelle Ungleichheiten nachgewiesen werden. Die hierfür notwendigen formalen Grundlagen sind im Vorfeld hierzu definiert und genauer erklärt worden. Das Kapitel schließt mit einer Reihe an Beweisen der kontextuellen Äquivalenz von Prozessausdrücken.

In *Kapitel 4* werden eine Zusammenfassung der Arbeit sowie ein Ausblick auf mögliche weitere Untersuchungen bzgl. des in der Arbeit behandelten Themas gegeben.

Kapitel 2

2 Formale Grundlagen

Dieser Abschnitt gibt zunächst einen Einblick in das formale Handwerkszeug, das nötig ist, um sich in der Welt der Kalküle i.Allg. wie auch ganz speziell im Pi-Kalkül zurechtzufinden. Hierzu werden die nötigen Notationen und Vorgehensweisen erklärt, welche zum Ziehen von Schlüssen innerhalb von Kalkülen notwendig sind, als auch passende Beispiele dazu geliefert.

2.1 Prozessalgebra

Zur Beschreibung *reaktiver Systeme* entwickelte sich aus der bekannten Automatentheorie die Prozessalgebra, um mit ihrer Hilfe nebenläufige Prozesse, die in den letzten Jahren immer mehr, beispielsweise als sog. *eingebettete Systeme (embedded systems)*, an Bedeutung gewonnen haben, zu beschreiben. Die Prozessalgebra dient hierbei der formalen Beschreibung, Modellierung und Verifizierung solcher Prozesse. Konzeptuell wurden, wie bereits aus der Automatentheorie bekannt, Zustände für Aktionen oder Folgen von Aktionen festgelegt und deren Nachfolgezustand spezifiziert. Anders als bei der klassischen Automatentheorie werden dabei eine *algebraische* Betrachtung sowie der Aspekt der Beobachtbarkeit und Verhaltensäquivalenz reaktiver Systeme betont. Das Verhalten eines reaktiven Systems ergibt sich hierbei aus der Komposition einzelner strukturell einfacher Elemente und stellt einen weiteren Unterschied zur Automatentheorie dar, da diese Art der Verhaltensbeschreibung vom *Zustandsraum* abstrahiert.

Grundbegriffe
Zunächst wird eine kleine Übersicht über einen Auszug aus verschiedenen Prozesstypen und deren Bedeutung gegeben.

- **(Sequentieller) Prozess**: Eine Folge von „Berechnungsschritten“ (**Aktionen**).
- **Sequentielles Programm**: Beschreibung von „Rechenvorgängen“ (Prozessen).

Tabelle 2.1: Prozesstypen Teil 1

- **(Paralleler) Prozess**: Folge von „Berechnungsschritten“, die **parallel** („nebenläufig“) zueinander ablaufen **können**.
- **Paralleles Programm**: Beschreibung paralleler Prozesse.

- **Reaktives System**: Auf Eingabe wartender Prozess („Endlosschleife“) mit evtl. parallelen Prozessen.

Tabelle 2.2: Prozesstypen Teil 2

Die Begriffe *Nebenläufigkeit* und *Parallelität* wurden im vorherigen bereits verwendet. Da sich beide Begriffe unterscheiden, ergibt sich eine genauere Differenzierung wie folgt:

Parallelität: Zwei Ereignisse finden **parallel** statt, wenn diese *gleichzeitig* ausgeführt werden.

Nebenläufigkeit (engl. concurrency): Zwei Ereignisse sind **nebenläufig**, wenn sie parallel ausgeführt werden *können*, zwischen ihnen somit keine kausale Abhängigkeit besteht.

Somit stellt der Begriff der **Nebenläufigeit** den allgemeiner gefassten Begriff dar.

2.1.1 Sequentielle Systeme

Im weitesten Sinne sind rein *sequentielle Systeme* jene Systeme, die aufbauend auf einer *formalen Semantik*[1], zu einer gegebenen Eingabe in endlich vielen Transformationsschritten (Berechnung) eine Ausgabe erzeugen. Wichtig ist u.a. die *Terminierung* solch eines Systems, da im Allgemeinen bei einer Berechnung ein eindeutiges Ergebnis erwartet wird.
Da es jedoch nicht nur die in der Fußnote erwähnte denotationale Semantik zur formalen Beschreibung der Eigenschaften eines Programms gibt, sei hier kurz auf die weiteren „formalen Semantiken“ eingegangen:

- Eine *translationale Semantik* beschreibt die Bedeutung eines Programms P einer Sprache L, indem sie es in ein Programm P' einer Untersprache L' *übersetzt*, für welches bereits eine als bekannt vorausgesetzte Semantik existiert. Die Bedeutung eines Konstrukts der Sprache L wird dadurch definiert, indem angegeben wird, wie das Programm in der gewählten *Untersprache* aussieht. Somit genügt es, zur vollen Beschreibung der Sprache L die Translation (Übersetzung) in deren

1 Im Speziellen einer denotationalen Semantik: Ein Programm stellt sich als partielle Funktion von Zuständen in Zuständen dar.

Untersprache anzugeben und deren bereits definierte Semantik zu nutzen. Das Problem einer solchen Semantikdefinition liegt darin, dass lediglich das Problem der Semantikdefinition auf eine andere Sprache verschoben wird (vgl. Beispiel 2.1).

Beispiel 2.1 (translationale Semantik in natürlichen Sprachen)
Angenommen, man wolle die Bedeutung der gültigen Zeichenkette *„tabel“* in der zu beschreibenden Sprache *Englisch* durch Angabe der Zeichenkette *„Tisch“* in der bereits bekannten Sprache Deutsch angeben, so nützt es nichts, wenn nicht bereits Deutsch als bekannt vorausgesetzt wird.

Die Informatik unterscheidet i.Allg. hierüber hinaus noch weiter Semantikansätze:

- Eine *axiomatische Semantik* beschreibt die Bedeutung eines Programms, indem angegeben wird, welche Eigenschaften mithilfe logischer Schlüsse eines definierten Axiomensystems gefolgert werden können. Hierbei ist es i.Allg. nicht notwendig, alle Eigenschaften eines Programms axiomatisch zu erfassen, da mitunter einige aus dem zu Grunde liegenden Logikkalkül gefolgert werden können. Als Beispiel mag an dieser Stelle das Beweiskalkül von Hoare [Hoa69] dienen.

- Eine *denotationale Semanik* beschreibt jedes Konstrukt einer Sprache L als mathematisches Objekt. Es wird hierbei der Effekt der Ausführung eines Konstrukts als mathematisches Objekt, z.B. als Funktion, modelliert. Eine entsprechende *semantische Funktion* lässt sich als Abbildung eines Programms in den entsprechenden mathematischen Raum auffassen. Veränderungen der im Programm vorkommenden Variablen werden somit in funktionalen Zusammenhang gesetzt, vgl. hierzu Abbildung 2.1.

  Beispielsweise lässt sich das Programm `quadrat n` (quadrierung eines Wertes `n`) mit dem Ergebnis `n`, in Prädikatenlogik, als Formel: $\boxed{\forall n : quadrat(n) = n^2}$ darstellen.

  An dieser Stelle sei angemerkt, dass für syntaktisch reichere Kalküle die denotationale Semantik im Hinblick auf die Modellierung nebenläufiger Systeme zu schwierig zu formulieren ist.

- Eine *operationale Semantik* beschreibt die Bedeutung eines Programms, indem abstrakt definiert wird, wie eine Berechnung zu ihrem Ergebnis kommt bzw. wie das Ergebnis eines Programms „berechnet“ wird. Es wird hier die Frage beantwortet „Wie wird das Programm ausgeführt?“ Ähnlich einem Automaten wird hier angegeben, wie sich dieser verhält. Überführungsfunktionen beschreiben die Zustandsänderungen eines Automaten von dessen Startzustand zu einem Endzustand. Bei Programmen wird so die Reihenfolge der Zustände, i.Allg. die aktuelle Speicherbelegung, von Anfang bis Ende definiert, wobei sich der Wert

eines Programms mittels Termersetzung über ein Ersetzungssystem berechnen lässt.

Eine Turingmaschine (TM) kann ebenfalls zur Beschreibung einer operationalen Semantik verwendet werden, da diese genau einen abstrakten operationalen Automaten darstellt (vgl. Abbildung 2.2).

Zusätzlich kann die operationale Semantik noch in eine *big-step* oder *small-step*-Semantik unterschieden werden. Die small-step-Semantik legt die Auswertung eines Programms in vielen kleinen Einzellschritten fest. Die Semantik eines solchen Programms ergibt sich so aus einer Folge von Einzelschritten, während die die big-step-Semantik die Auswertung in größeren Schritten, zumeist in einem, festlegt.

- Eine *kontextuelle Semantik* ist eine operationale Semantik, die mithilfe einer kontextuellen Präordnungsrelation (vgl. Definition 3.23) erweitert wurde, sodass dies zu einer kontextuellen Äquivalenz führt. Zwei Prozessterme P und Q sind kontextuell gleich bzw. zeigen gleiches Verhalten, wenn sie in allen Programmen (Programmkontexten) gegeneinander ausgetauscht werden können, ohne dass sich die Semantik des Programms verändert [Pit97]. Die kontextuelle Semantik bietet somit die Möglichkeit, Programmtransformationen einer Sprache auf ihre Korrektheit zu prüfen.

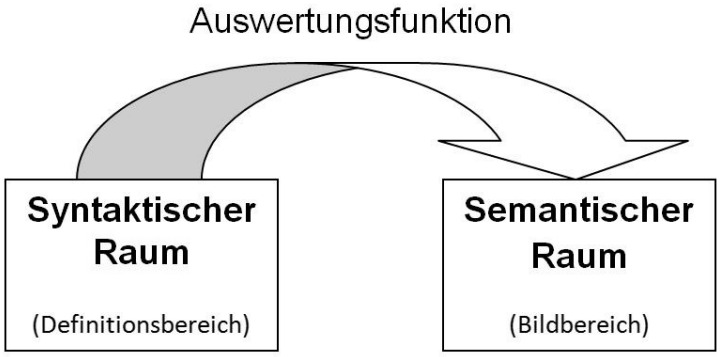


Abbildung 2.1: Konstrukten einer Sprache (syntaktischer Raum) werden mittels Auswertungsfunktion Werte (semantischer Raum) zugeordnet. Zustandsänderungen des Programms werden mittels math. Funktionen beschrieben

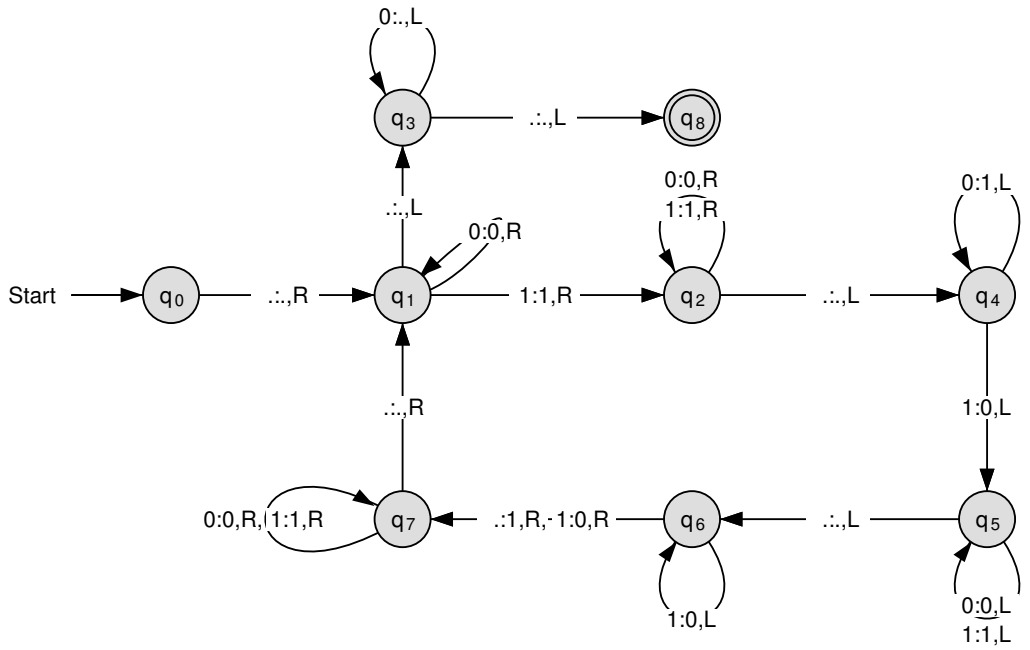


Abbildung 2.2: Zwei Dualzahlen werden mithilfe einer TM addiert. Dabei ist zu sehen, wie für jeden Zustand dessen Übergangsfunktion angegeben ist und somit die Bedeutung des Programms der Addition sich durch den operationalen Verlauf der TM definiert.[2]

Wie bereits zu Anfang des Kapitels erwähnt, entwickelte sich die Beschreibung *reaktiver Systeme* aus der Automatentheorie, sodass nach der Übersicht der einzelnen Semantikdefinitionen – in Definition 2.1 die rein formale Beschreibung eines einfachen Automaten – aufgezeigt und anhand eines Beispiels näher erklärt wird. Hierbei ergibt sich das Verhalten eines solchen Systems mit endlich vielen Zuständen sowie Zustandsübergängen (Transitionen) durch eine Folge von Aktionen, die von *Anfangszuständen* in *Endzustände* führen.
Transitionssysteme repräsentieren somit Zustände und Zustandsübergänge von Systemen.

Definition 2.1 (Transitionssystem)
Ein *Transitionssystem* hat die Form $A = (Q, \Sigma, I, \delta, F)$, und es gilt:

- Q : Zustandsmenge
- Σ : endliches Eingabealphabet
- $I, F \subseteq Q$: Menge von *Anfangs-* und *Endzuständen*

2 Darstellung angelehnt an W.P. Kowalk [Kow96].

- $\delta : Q \times \Sigma \to Q$: Übergangsrelation

A heißt *endlich*, falls Q endlich ist.

Solch ein Transitionssystem heißt *nichtdeterministischer endlicher Automat*, falls $I = q_o$ für ein $q_0 \in Q$ und $\delta : Q \times \Sigma \to 2^Q$ eine Abbildung in die Potenzmenge von Q ist. Dies bietet die Möglichkeit, das Verhalten eines sequentiellen Prozesses mittels *Transitionen* zu beschreiben.

Der in Abbildung 2.3 gezeigte Automat erkennt und akzeptiert Eingabefolgen der Form a^*ba^*. Er besteht aus zwei Zuständen, der Startzustand q_0 ist hier mit einem Pfeil – q_1 als akzeptierender Zustand – doppelt umkreist dargestellt. Erlaubte Zustandswechsel sind hier durch die Übergangspfeile illustriert.

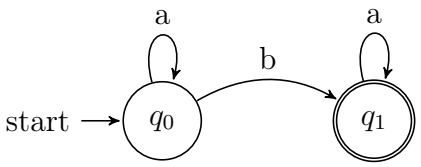


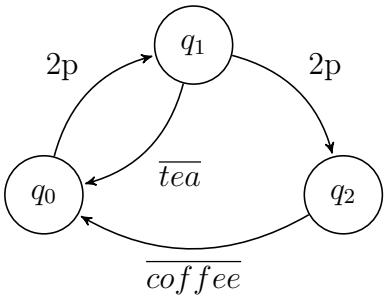
Abbildung 2.3: Ein endlicher Automat zur Erkennung der durch den regulären Ausdruck a^*ba^* definierten Sprache

Der Übergang eines *Zustandes* in einen *Nachfolgezustand*, hier (q_0, b, q_1): Geschrieben als $q_0 \xrightarrow{b} q_1$, wobei q_0 der Startzustand und q_1 der akzeptierende Endzustand ist sowie $\xrightarrow{b}$ den Übergang unter Eingabe von b darstellt.

Ein Transitionssystem, dass ohne Start- sowie Endzustand auskommt, also einen Automaten ohne Start- und Endzustand beschreibt, wird *Labelled Transitions System (LTS)* genannt. So kann ein *endlicher Automat* als Spezialfall eines *LTS* angesehen werden.

Definition 2.2 (Labeled Transition System)
Ein Tripel $[S, E, A]$ heißt LTS, wobei S eine endliche Menge von Zuständen, A eine endliche Menge von Aktionen und E mit $E \subseteq S \times A \times S$ eine Menge von Ereignissen ist.

Beispiel 2.2 (LTS eines Getränkeautomaten)
Auf Basis von Definition 2.2 lässt sich folgender Getränkeautomat als *LTS* beschreiben [Mil99].



$S = \{p_0, p_1, p_2\}$ *und* $E = \{(p_0, 2p, p_1), (p_1, \overline{tea}, p_0), (p_1, 2p, p_2), (p_2, \overline{coffee}, p_0)\}$

2.1.2 Reaktive Systeme

Bei *reaktiven Systemen* handelt es sich i.Allg. um Systeme mit unbeschränkter Laufzeit, bei denen meist mehrere Entitäten miteinander parallel kommunizieren bzw. miteinander interagieren. Ein reaktives System berechnet nicht einfach ein Ergebnis $f(x)$ für eine Eingabe x, sondern *reagiert* auf Ereignisse aus seiner Umgebung. Charakteristisch für solche Systeme ist ihre Interaktion mit evtl. vielen nebenläufigen Prozessen sowie ihren nicht unbedingt eindeutig bestimmten Ergebnissen (aufgrund von Nichtdeterminiusmus), also auch der Möglichkeit, ohne erkennbares Resultat zu operieren (reiner Nachrichtenaustausch zur Koordination einzelner Prozesse). Anders als bei rein sequentiell arbeitenden Systemen sind in diesem Fall gegebenenfalls mehrere nebenläufige Prozesse mit deren Ausgaben bei der kompletten Systembeschreibung zu berücksichtigen.
Abstrakt kann das Verhalten reaktiver Systeme anhand des nachfolgenden Pseudo-Codes beschrieben werden.

loop

Überwache jeden INPUT auf mögliche Eingaben,
je nach Eingabewert führe passende Aktion aus.

forever

Geprägt haben David Harel und Amir Pnueli den Ausdruck *reaktive Systeme* in [Har85] als Systeme, die gerade ihre Berechnungen als Reaktion auf Stimuli ihrer Umgebung durchführen.

2.1.3 Kommunikation

Wie bereits teilweise in den vorigen Abschnitten angesprochen, kann *Kommunikation* auf unterschiedliche Art und Weise stattfinden. Zwei Konzepte zum Nachrichtenaustausch zwischen Prozessen werden nachfolgend kurz beschrieben, wobei ausschließlich das Konzept der *synchronen Kommunikation* Bestandteil dieser Ausarbeitung ist.

- **Asynchrone Kommunikation**:
  Gesendete Nachrichten werden bei dem jeweiligen Empfänger in einem *Nachrichtenspeicher* abgelegt. Zum passenden Zeitpunkt wird die entsprechende Nachricht dann vom Empfänger aus diesem Puffer geholt. Senden und Empfangen einer Nachricht geschehen somit zu verschiedenen Zeitpunkten (vgl. Abb. 2.4).

- **Synchrone Kommunikation**:
  Eine Nachricht kann nur zum Zeitpunkt des Sendens empfangen werden. Es findet also *eine* gleichzeitige Aktion statt (vgl. Abb. 2.5).

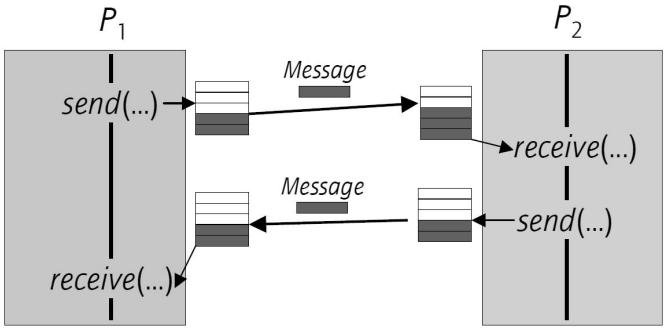


Abbildung 2.4: Asynchrone Kommunikation unter Zuhilfenahme von Nachrichtenspeichern[3]

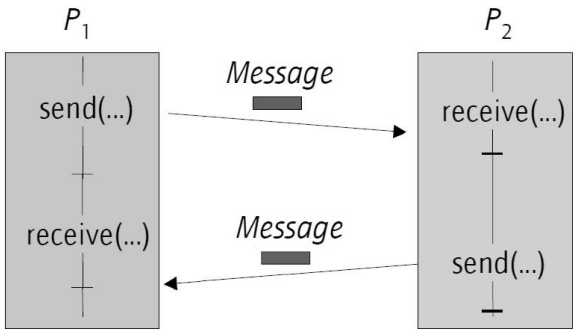


Abbildung 2.5: Synchrone Kommunikation, bei der nur gesendet wird, wenn auch der Empfänger bereit ist[3]

3 Darstellung angelehnt an G.Mühl [Müh04].

Wie sich eine solche Kommunikation, also die Interaktion kommunizierender Prozesse, rein formal mittels Prozessalgebra ausdrücken lässt und ganz im Speziellen mithilfe des *synchronen Pi-Kalküls*, wird zu einem späteren Zeitpunkt in Kapitel 3 betrachtet.

Da im Umfang dieser Arbeit weiterführende Verhaltensuntersuchungen des CCS sowie auch später des Pi-Kalküls durchgeführt werden, erfolgt im nächsten Abschnitt eine kleine Einführung in ein Konzept, das hierbei Verwendung finden wird – das logische Schließen aus der Philosophie, ganz im Speziellen aus einem Teilgebiet der Philosophie, der Logik.

2.2 Logisches Schließen

Aussagen über eine Menge von Sätzen zu treffen, um genau diese näher zu beleuchten, ist Aufgabe der deduktiven Logik (zweiwertiger Logik). Es geht in ihr nicht um eine Erweiterung des *Wahren*, sondern um eine genauere Untersuchung dessen, was in den einzelnen Aussagen (Prämissen) *verborgen* ist [WKE01].

> „Die (*deduktive*) Logik ist somit ein Instrument, welches dem Benutzer ermöglicht, die Urteile eines verborgenen Bereichs - ob diese nun in der Sprache des Alltags oder in einer Wissenschaftssprache formuliert sind - in folgender Weise zu ordnen: Man unterteilt die akzeptierten Sätze in Grundsätze (oder Axiome) und Lehrsätze (oder Theoreme). Die Lehrsätze werden mit Hilfe logischer Schlüsse entweder direkt aus den Grundsätzen oder aber mittelbar aus ihnen unter Verwendung von bereits daraus abgeleiteten Lehrsätzen gewonnen. *Falls* die Wahrheit der Grundsätze bereits *anderweitig* gesichert ist, sichert die Logik die Wahrheit der *Lehrsätze* somit durch deren Deduktion aus den *Grundsätzen*.
> Die Logik hilft darüber hinaus, festzustellen, ob ein solches System von akzeptierten Sätzen widerspruchsfrei ist.
> [...]
> In manchen Fällen hilft die Logik darüber hinaus auch, zu ermitteln, welche - stillschweigenden - Voraussetzungen beim wahrheitserhaltenden Argumentieren gemacht werden, um zu bestimmten Konklusionen zu gelangen; [...]“
>
> [WKE01, S.28]

Axiom
↓
Einzelaussage

Ein wichtiger Umstand deduktiver Schlüsse besteht darin, dass ein deduktiv gültiges Argument nichts über den Wahrheitsgehalt der Prämissen aussagt, sondern lediglich wahr sein kann, wenn die Prämissen wahr sind. Die Information der Konklusion (die Aussage) steckt bereits implizit in den Prämissen [Kah09]. Im Besonderen lassen sich aus Axiomen, d.h. aus allgemein gültigen Aussagen die ihrerseits nicht bewiesen werden können und müssen, immer wahre Einzelaussagen folgern.

2.2.1 Inferenzregeln

Um aus bereits bekannten Aussagen weitere Aussagen ableiten zu können, werden Inferenzregeln verwendet. Mit ihrer Hilfe ist es möglich, aus bereits bekannten Prämissen implizit darin enthaltene Informationen abzuleiten. Dabei ist zu beachten, dass Inferenzregeln lediglich die Form eines Argumentes bestimmen und durch dortiges Einsetzen eines konkreten Inhalts ein deduktiv gültiges Argument erzeugt wird. Somit kann ein solches Argument als Instantiierung von Inferenzregeln angesehen werden. Hierzu einige Beispiele:

Beispiel 2.3 (Deduktiver Schluss)
Wahrheitskonservierende Schlüsse

$$\frac{\text{Alle Eisbären sind weiß.} \quad \text{Das Tier hier ist ein Eisbär.}}{\text{Das Tier ist weiß.}} \qquad \frac{\text{Wenn es regnet, wird die Erde nass.} \quad \text{Es regnet.}}{\text{Die Erde ist nass.}}$$

Beispiel 2.4 (Hypothetischer Syllogismus [WKE01])

$$\frac{A \longrightarrow B \quad B \longrightarrow C}{A \longrightarrow C} \qquad \frac{\mathit{Hypothese}}{\mathit{Konklusion}}$$

Bedingt durch beide Prämissen (Hypothesen) $A \longrightarrow B \wedge B \longrightarrow C$, folgt als Schluss die Konklusion $A \longrightarrow C$.

Über dem *Bruchstrich* befinden sich die Voraussetzungen unter deren Annahme sich der Ausdruck darunter ergibt. Diese Inferenzsysteme sind hier stellvertretend für viele weitere anzusehen. An dieser Stelle wird jedoch darauf verzichtet, weitere Beispiele anzugeben, da lediglich die Funktionsweise eines solchen erkannt werden sollte, um in zukünftigen Betrachtungen deren Aussage verstehen zu können. Um genauere Aussagen über Prozessausdrücke treffen zu können bzw. Reduktionen zu definieren, werden solche Inferenzsysteme bei der Betrachtung des CCS sowie des Pi-Kalküls als Grundgerüst immer wieder Verwendung finden.

2.3 Der Vorläufer des Pi-Kalküls: CCS

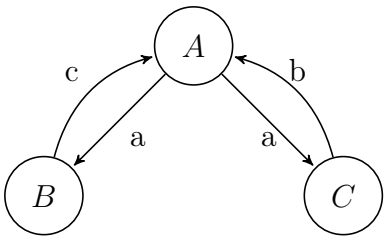
Wie bereits in Abschnitt 1.1 der Einleitung erwähnt, kann der von Robin Milner begründete *Calculus of Communicating Systems (CCS)* als Vorläufer des ebenfalls von ihm entwickelten *Pi-Kalküls* betrachtet werden. Aus diesem Grund wird im Nachfolgenden eine kleine Einführung in CCS nach [Mil82, Mil99, Ace07, Kön10] erfolgen, bevor in Kapitel 3 auf den hierauf aufbauenden *Pi-Kalkül* eingegangen wird.

Robin Milner wurde unter anderem für seine herausragende Arbeit, bei der Weiterentwicklung der Theorie der endlichen Automaten um die Möglichkeit der Kommunikation und hieraus entstandenen Calculus of Communicating Systems, 1991 mit dem Turing Award der Association for Computing Machinery (ACM) ausgezeichnet.

> „For the three distinct complete achievements: 1) LCF, the mechanization of Scott's Logic of Computable Functions , [...], 2) ML, the first language to include polymorphic type inference [...], 3) CCS, a general theory of concurrency. [...]"
>
> [gemäß Turing Award Zitierung]

Die Idee von *CCS* besteht nun darin, Transitionssysteme in textueller Form zu notieren.

Beispiel 2.5 (Transistionssystem in CCS)
Nachfolgender *Automat* lässt sich rein formal mit CCS darstellen.



$$A := a.B + a.C$$
$$B := c.A$$
$$C := b.A$$

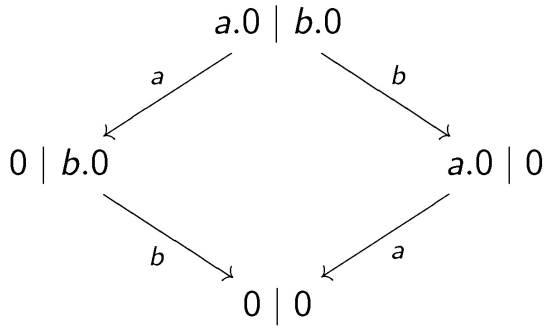
$$\text{Mit } A := a.c.A + a.b.A$$

Wie sind also die Symbole im obigen Beispiel zu deuten? Hierzu soll zunächst eine *informelle* Erklärung genügen, um später in diesem Kapitel diese auch rein formal zu festigen.

- $a.P$ ist ein Prozess, der zunächst die *Aktion* a ausführt und sich anschließend wie Prozess P verhält, dargestellt in Präfix-Schreibweise.
- $P_1 + P_2$ beschreibt die *nicht deterministische Auswahl* des auszuführenden Prozesses (Entweder P_1 oder P_2).

Auch die parallele Komposition von Prozessen ist im CCS möglich – dargestellt als $P_1 \mid P_2$. Hierzu ebenfalls zunächst ein Beispiel.

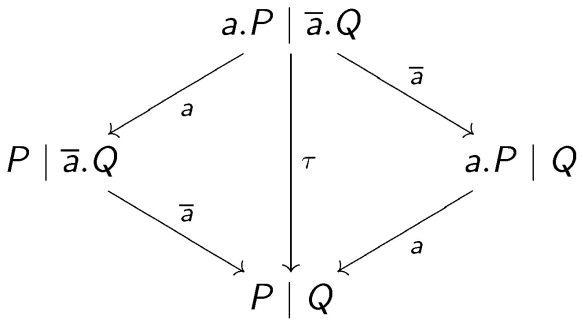
Beispiel 2.6 (CCS und parallele Prozesse)
Mögliche Aktionen der unabhängigen Prozesse $a.0 \mid b.0$.



0 wird als inaktiver *Null-Prozess* angesehen, der keine Aktion mehr ausführt.

Ähnlich wie in Beispiel 2.6 ist es möglich, die *Kommunikation* von Prozessen untereinander darzustellen. In Beispiel 2.7 sind zwei Prozesse mit zueinanderpassenden Ein- und Ausgabeaktionen beschrieben. Dies ermöglicht ihnen eine interne Aktion τ als eine Art von Synchronisation, wobei $\bar{a}$ mitunter auch *Koaktion* zu a genannt wird.

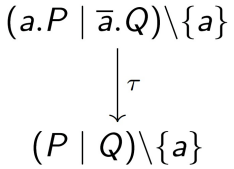
Beispiel 2.7 (Synchronisation der Ein-/Ausgabe)
Prozesse $a.P \,|\, \overline{a}.Q$ lassen folgende Aktionen zu.



τ stell hierbei die *interne*, nach außen *unsichtbare* Aktion dar.

Wird nicht gewünscht, dass bestimmte Aktionen von Prozessen nach *außen* sichtbar sind, so ist es möglich, diese so zu *verschatten* bzw. zu *restringieren*, dass zunächst nur eine *interne* Kommunikation stattfinden kann, bevor evtl. weitere Aktionen ausgeführt werden. Wie so etwas formal dargestellt wird, wird im folgenden Beispiel gezeigt.

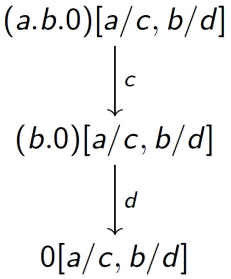
Beispiel 2.8 (Restringierte Kommunikation)
Prozesse der Form $(a.P \,|\, \bar{a}.Q) \setminus \{a\}$ lassen nur folgende parallele Kommunikation zu.



Hierbei stellt ... $\setminus \{a\}$ die Restriktion des Prozesses $(a.P \,|\, \bar{a}.Q)$ dar.

Wie bereits aus anderen Kalkülen bekannt, ist es auch in CCS möglich, Umbenennungen vorzunehmen. Umbenannt werden hier Aktionen von Prozessen, bevor diese nach außen kommuniziert werden. Dabei können Umbenennungen sowohl *gleichzeitig* als auch *hintereinander* durchgeführt werden. Aufgrund dieser beiden Ausführungsvarianten kann man unterschiedliche Ergebnisse erhalten (vgl. Beispiel 2.9).

Beispiel 2.9 (Umbenennung)
Umbenennung von $(a.b.0)\ [a/c,b/d]$ bevor Aktionen nach außen gegeben werden.



Bei gleichzeitiger Umbenennung von $[a/c,c/d]$ werden alle a in c umbenannt **und** alle c in d. Eine Umbenennung der a in d findet hier nicht statt, im Gegensatz zur Ausführung von zwei Umbenennungen $[a/c]\,,[c/d]$ hintereinander, bei welcher tatsächlich alle a in d umbenannt werden.

2.3.1 CCS: Syntax und Semantik

Nachdem nun eine eher allgemeine Einführung in CCS erfolgte, wird nachfolgend eine genauere syntaktische sowie semantische Beschreibung des CCS gegeben.

Definition 2.3 (Syntax von CCS)
Ein CCS-Prozess ist entweder

- ein inaktiver Prozess **0** (bzw. nil),
- ein Prozess der Form $\alpha.P$, mit $\alpha \in Act$ (Präfixoperator),
- eine nichtdeterministische Auswahl $P_1 + P_2$ (Auswahloperator),
- eine parallele Komposition $P_1 \,|\, P_2$,
- eine Restriktion $P \setminus N$, mit $N \subseteq \mathcal{N}$,
- eine Umbenennung $P\,[f]$, mit $f : \mathcal{N} \to \mathcal{N}$ als Umbenennungsfunktion, oder
- eine Konstante A, wobei A der Form $A := P$ vordefiniert ist.

P, P_1, P_2 sind selbst wieder Prozesse aus der Menge aller Prozess $\mathcal{K}$, $\alpha \in Act$, mit $Act = \{\tau\} \cup \mathcal{N} \cup \{\overline{\alpha} \,|\, \alpha \in \mathcal{N}\}$ ist die Menge aller Aktionen und $\mathcal{N}$ die Menge aller Nachrichten.

Somit lassen sich alle Prozessausdrücke mit folgender Grammatik P darstellen:

$$P ::= \quad nil \quad | \quad \alpha.P \quad | \quad P + P \quad | \quad P \,|\, P \quad | \quad P \setminus N \quad | \quad P\,[f] \quad | \quad A$$

Da es mitunter verschiedene Notationen für ein und denselben Prozessausdruck gibt, sei hier erwähnt, dass für zuvor genannte *Restriktion* aus Definition 2.3 die Notation

(alternative Notaion der Restriktion) $(\nu a).P$ mit $a \in N$

als äquivalent anzusehen ist und so auch später Verwendung findet, da so bei größeren Prozessausdrücken die Übersicht gewahrt bleibt. Hierbei stellt der ν-Operator ebenfalls sicher, dass der Kommunikationskanal a lediglich für Prozess P sichtbar ist.

Nachdem bisher beschrieben wurde, *wie* sich Prozessausdrücke zusammensetzen *(Stichwort: Syntax)*, folgt nun die Beschreibung der *Bedeutung* syntaktisch korrekter Programme mithilfe folgender Übergangsrelation - der *strukturellen operationellen Semantik (SOS)* . Allgemein lassen sich Transitionen eines CCS-Prozesses mithilfe von Ableitungsregeln der Form:

$$\frac{X_1,...,X_n}{Y}$$

darstellen.

Wie bereits aus Abschnitt 2.2 bekannt, handelt es sich hierbei um ein Inferenzsystem mit den Vorbedingungen $X_1,...,X_n$ sowie der Folgerung Y. Es ist nicht zwingend notwendig, dass Prämissen vorhanden sein müssen. Der Fall $n = 0$ beschreibt lediglich eine immer geltende Folgerung Y, d.h. ein Axiom.

Definition 2.4 (Reduktionsregeln)
Die in Definition 2.5 eingeführten Ableitungsregeln, stellen die *Reduktionsregeln*[4] des CCS dar. Hierbei ist eine Regel

$$\text{(name)} \quad a \longrightarrow b$$

wie folgt zu lesen: Ein Ausdruck der Form a kann durch einen Ausdruck der Form b unter Verwendung der Regel mit der Bezeichnung (name) ersetzt werden.

Mitunter wird die verwendete Reduktionsregel auch über dem Reduktionspfeil notiert.

4 Analog zu [Kön10].

Definition 2.5 (SOS von CCS)
Sei $P, Q \in \mathcal{K}_{\mathcal{CCS}}$ und "$\stackrel{\alpha}{\rightarrow}$"definiert durch:

$$\text{(PRE)}\ \frac{}{\alpha.P \stackrel{\alpha}{\rightarrow} P} \qquad \text{(COM)}\ \frac{P \stackrel{x}{\rightarrow} P' \quad Q \stackrel{\overline{x}}{\rightarrow} Q'}{P|Q \stackrel{\tau}{\rightarrow} P'|Q'}$$

$$\text{(PAR1)}\ \frac{P \stackrel{\alpha}{\rightarrow} P'}{P|Q \stackrel{\alpha}{\rightarrow} P'|Q} \qquad \text{(PAR2)}\ \frac{Q \stackrel{\alpha}{\rightarrow} Q'}{P|Q \stackrel{\alpha}{\rightarrow} P|Q'}$$

$$\text{(NDC1)}\ \frac{P \stackrel{\alpha}{\rightarrow} P'}{P + Q \stackrel{\alpha}{\rightarrow} Q'} \qquad \text{(NDC2)}\ \frac{Q \stackrel{\alpha}{\rightarrow} Q'}{P + Q \stackrel{\alpha}{\rightarrow} P'}$$

$$\text{(RES)}\ \frac{P \stackrel{\alpha}{\rightarrow} P' \text{ mit } \alpha \neq x,\overline{x} \in \mathcal{N}}{(\nu x).P \stackrel{\alpha}{\rightarrow} (\nu x).P'}$$

Nicht immer existiert nur eine mögliche Ableitungsfolge, was anhand des nachfolgenden Beispiels näher gezeigt wird.

Beispiel 2.10 (Ableitung)
Für den CCS-Ausdruck $(x.P \,|\, Q) \,|\, \bar{x}.R$ ist folgende Transition möglich:

$$\dfrac{\dfrac{\text{(PRE)} \quad x.P \stackrel{x}{\rightarrow} P}{\text{(PAR)} \quad x.P \,|\, Q \stackrel{x}{\rightarrow} P \,|\, Q} \qquad \dfrac{}{\text{(PRE)} \quad \bar{x}.R \stackrel{\bar{x}}{\rightarrow} R}}{\text{(PAR2)} \quad (x.P \,|\, Q) \,|\, \bar{x}.R \stackrel{\tau}{\rightarrow} (P \,|\, Q) \,|R}$$

Die Ableitung $PAR(PAR(PRE),PRE)$ ist aufgrund der nichtdeterministischen Wahl der Transitionsschritte nicht die einzig mögliche. $PAR(PAR(PRE))$ stellt eine weitere mögliche Ableitungsfolge dar und führt zu:

$$(x.P \,|\, Q) \,|\, \bar{x}.R \stackrel{x}{\longrightarrow} (P \,|\, Q) \,|\, \bar{x}.R$$

Es sei nochmals darauf hingewiesen, dass in dieser Arbeit lediglich das Kommunikationskonzept der synchronen Kommunikation Anwendung findet. Demzufolge findet nur dann eine Kommunikation zwischen Prozessen statt, wenn sich diese – wie unten zu sehen ist – im passenden *Sende-* sowie *Empfangsmodus* befinden.

$$\bar{x}.P \,|\, x.Q \stackrel{\tau}{\rightarrow} P \,|\, Q$$

D.h., lediglich Transitionen der Form $\stackrel{\tau}{\rightarrow}$ sind von Interesse.

2.4 Prozessgleichheit

Nachdem sowohl die Syntax als auch die Semantik für entsprechende Kalküle definiert wurden und es somit möglich ist, Programme bzw. Prozesse im jeweiligen Kalkül zu formulieren und auch auszuführen, fehlt bisher ein wichtiger Begriff, der Begriff der Gleichheit von Programmen. Warum dieser so wichtig ist, liegt darin begründet, dass Programmtransformationen oder die Optimierung der Programme durch einen Compiler nicht dazu führen dürfen, dass das ursprüngliche Programm und das optimierte nicht mehr gleich sind.
Es gibt hierbei jedoch nicht nur *einen* Gleichheitsbegriff, der beschreibt, wann zwei Ausdrücke gleich sind. Konkret seien hier sowohl die *kontextuelle Gleichheit* als auch die *Bisimulation* als Möglichkeiten, Prozesse zu vergleichen, genannt.

In dieser Arbeit wird der Pi-Kalkül im Hinblick auf seine kontextuelle Semantik untersucht. Zunächst werden jedoch beide Konzepte kurz vorgestellt.

2.4.1 Bisimulation

Die Bisimulation ist als Konzept von Milner und Park [Mil82, Par81] zur Nachprüfung der semantischen Äquivalenz von Programmen eingeführt worden. Die Idee hierbei besteht darin, das beobachtbare Verhalten von Programmen bzw. deren Zustände – z.B. bei einem Automaten – zu vergleichen. Erfolgt hierbei eine Unterscheidung zwischen beobachtbaren und nicht beobachtbaren Übergängen, so wird dies als *schwache Bisimulation* [Mil89] bezeichnet.

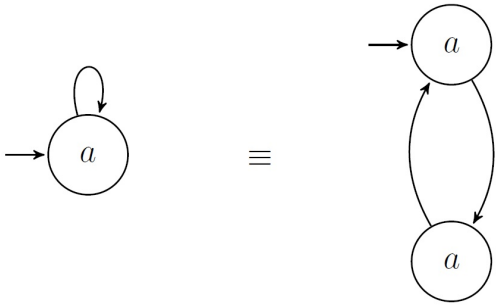
Definition 2.6 (Bisimulation)
Es seien P Prozesse und $\xrightarrow{\alpha}$ gegeben.
Eine binäre Relation R über Prozessen ($R \subseteq P \times P$) ist eine Bisimulation, wenn für alle (P,Q) $\in R$ gilt;

- Falls $P \xrightarrow{\alpha} P'$, dann existiert ein Prozess Q', sodass $Q \xrightarrow{\alpha} Q'$ und $(P', Q') \in R$
- Falls $Q \xrightarrow{\alpha} Q'$, dann existiert ein Prozess P', sodass $P \xrightarrow{\alpha} P'$ und $(P', Q') \in R$.

Gegeben zwei Prozesse P und Q, so ist P bisimular zu Q (geschrieben $P \sim Q$) falls eine Bisimulation R mit (P,Q) $\in R$ existiert.

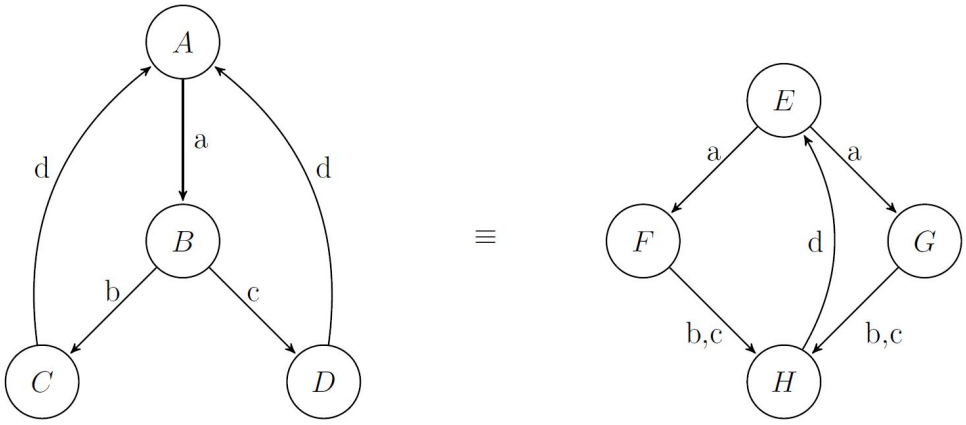
So können, wie in Beispiel 2.11 und 2.12 zu sehen ist, zunächst unterschiedlich aussehende Prozesse als äquivalent identifiziert werden.

Beispiel 2.11
Trotz unterschiedlicher Struktur sind diese Systeme bisimular.



Beispiel 2.12
Die Zustände A und E sind mit folgender Bisimulationsrelation, bisimular:

$$R = \{(A,E),(B,F),(B,G),(C,H),(D,H)\}$$



Aussage:
Die Bisimularität $\sim$ ist selbst eine Bisimulation, was sich leicht beweisen lässt.

Beweis.

- Für zwei Zustände Z_1, Z_2 gelte $Z_1 \sim Z_2$, wobei eine Bisimulation R mit $(Z_1, Z_2) \in R$ existiert. So ist zu zeigen dass es für jede Transition und deren Umkehrung $Z_1 \xrightarrow{\alpha} Z_2'$ eine Transition der Form $Z_2 \xrightarrow{\alpha} Z_2'$ gibt, sodass $Z_1' \sim Z_2'$ gilt.

  Sei $Z_1 \xrightarrow{\alpha} Z_1'$ eine beliebige Transition, da $(Z_1, Z_2) \in R$ und R eine Bisimulation ist, gibt es eine Transition $Z_2 \xrightarrow{\alpha} Z_2'$ mit $(Z_1', Z_2') \in R$ und da $R \subseteq \sim$ folgt somit $Z_1' \sim Z_2'$.

- Die Umkehrung erfolgt analog. □

2.4.2 Kontextuelle Gleichheit

Kontextuelle Gleichheit ist neben der Bisimulation eine weitere Möglichkeit, Prozesse auf ihre Gleichheit hin zu untersuchen. Da sich diese Arbeit in ihrem weiteren Verlauf mit der Untersuchung von Prozessausdrücken des Pi-Kalküls im Hinblick auf deren kontextuelle Gleichheit beschäftigt, folgt hier lediglich eine kurze und knappe Einführung in das Konzept der kontextuellen Gleichheit/Äquivalenz und wird im laufenden Kontext der Arbeit noch genauer beleuchtet.

Zwei Dinge sind nach dem Leibniz'schen Prinzip genau dann gleich, wenn sie die gleichen Eigenschaften bezüglich aller Eigenschaften besitzen und somit untereinander ausgetauscht werden können.[5] Im Hinblick auf Prozesskalküle kann dies wie folgt aufgefasst werden:

> *Zwei Prozesse P,Q sind gleich, wenn man sie im Hinblick auf ihr Verhalten, nicht unterscheiden kann, egal, in welchem Kontext man sie verwendet.*

Kontexte stellen Ausdrücke dar, die anstelle eines Unterausdrucks ein *Loch* enthalten, in das wiederum Ausdrücke eingesetzt werden können, um so einen neuen Ausdruck zu erhalten.

Definition 2.7 (Kontext)
Ein Kontext $C[\cdot]$ ist ein Prozess, der genau an einer Stelle ein „Loch" enthält. Das Konstrukt $C[P]$ beschreibt den Prozess, der entsteht, nachdem Prozess P in das „Loch" des Kontextes $C[\cdot]$ eingesetzt wurde.

Um an dieser Stelle nicht schon zu weit vorzugreifen, kann angenommen werden, dass für eine gegebene Syntax einer Programmiersprache deren Beobachtungsprädikat[6] gegeben ist. Somit ist es nun möglich, die kontextuelle Gleichheit wie folgt zu definieren.

Definition 2.8 (Kontextuelle Gleichheit)
Seien P und Q zwei Prozesse und $\diamond$ das Beobachtungsprädikat, so gilt:

$$P \sim_c Q \text{ genau dann wenn } \forall C : \diamond C[P] \Longleftrightarrow \diamond C[Q]$$

Dann sind P,Q kontextuell gleich (geschrieben $\sim_c$).

5 Analog zu [Lei90, Charactersitica XIX und XX].
6 Beobachtungsprädikate bilden Programme bzw. deren Verhalten auf Boole'sche Werte ab.

Bis zu diesem Punkt wurde nicht näher angegeben, welches Verhalten es genau zu beobachten gilt. I.Allg. handelt es sich jedoch bei deterministischen Sprachen um deren Terminierung, was wiederum bei nichtdeterministischen Sprachen erweitert werden muss, da es mitunter vorkommen kann, dass unterscheidende Programme aufgrund des Nichtdeterminismus als gleich angesehen werden.

Mithilfe von Kontexten sollen Prozessausdrücke auf deren *kontextuelle Äquivalenz* geprüft werden. Im nächsten Kapitel soll eine solche Aussage in Bezug auf die kontextuelle Semantik des Pi-Kalküls getroffen werden. Hierbei wird das eine oder andere Konzept noch etwas genauer in Bezug auf den Pi-Kalkül dargestellt. Weiterhin erfolgt eine nähere und ausführlichere Beschreibung der *kontextuellen Gleichheit*.

Vorwegnehmend sei gesagt, dass der Nachweis sich kontextuell unterscheidender Prozesse zumeist leichter zu führen ist als der auf kontextuelles äquivalentes Verhalten.
Zum Nachweis von sich kontextuell unterscheidenden Prozesse genügt es, einen passenden Kontext C zu finden, der gerade die untersuchten Prozesse in Bezug auf ihr Terminierungsverhalten unterscheidet.
Im Falle der kontextuellen Äquivalenz hingegen ist zu zeigen, dass sich die untersuchten Prozesse in keinem Kontext (d.h. in keinem *aller möglichen* Kontexte) unterscheiden lassen.

Kapitel 3

3 Der Pi-Kalkül

Der Pi-Kalkül [Mil99] ist ein Kalkül zur Beschreibung mobiler Prozesse. Vorgestellt als eine Erweiterung des bereits in den 1970er-Jahren von Robin Milner erdachten *Calculus of Communication Systems (CCS)*, wurde er Ende der 1980er-Jahren von Robin Milner, Joachim Parrow und David Walker. Der Pi-Kalkül liefert ein Modell zur Beschreibung der Kommunikation sich gegenseitig beeinflussender (kommunizierender) Systeme. Mit ihm ist es möglich, die dynamische Vernetzung solcher Systeme formal darzustellen und zu beschreiben. Dabei zählt er heute zu den bekanntesten und bedeutendsten interaktionsbasierten Prozesskalkülen.

Als ausführliche Einführung in die Theorie des Pi-Kalküls sei an dieser Stelle die Literatur von [Mil99, San01] und [Par01] Erwähnt, da im Umfang dieser Arbeit lediglich jene Konzepte des Pi-Kalküls Erläuterung finden, die notwendig sind, um die hier gemachten Untersuchungen durchzuführen.

Um Vorstellung der Theorie dem Lese zu erleichtern, kann Mobilität im Pi-Kalkül als das Wandern von Verbindungen zwischen Prozessen im virtuellen Raum aufgefasst werden. In der realen Welt steht hierfür beispielsweise die Bewegung eines Handys im Mobilfunknetz.
Wie bereits in der Einführung beispielhaft erwähnt, spannen diverse Antennen der Mobilfunkbetreiber ein Funknetz auf, in die sich ein mobiles Endgerät (Handy) einwählen muss, um erreichbar zu sein. Hierbei verbindet sich das Mobiltelefon mit einer Funkantenne, die gerade mit ihrem Funkradius die Position des Handys abdeckt. Bewegt sich nun das Mobiltelefon aus der Reichweite dieser Antenne, ist es, um weiterhin erreichbar zu bleiben nötig, die Verbindung mit der nun nicht mehr erreichbaren Antenne zu trennen und mit einer weiteren Antenne, welche die neue Position des Handys mit ihrem Funkradius abdeckt, eine neue Verbindung zu etablieren. Genau solche sich dynamisch ändernden Verbindungen zwischen unterschiedlichen Entitäten können mithilfe des Pi-Kalküls modelliert werden.

Im Pi-Kalkül stellen sich die einzelnen Entitäten als miteinander kommunizierende Prozesse dar, die in einer sich ändernden Umgebung Nachrichten untereinander austauschen. Ihre Verbindungen werden über sog. Kommunikationskanäle (kurz *Kanäle*) realisiert.

Die hierüber gesendeten Informationen werden als Nachrichtennamen (kurz *Namen*) bezeichnet. Dies ist auch der fundamentalste Unterschied zu bisherigen Kalkülen, wie etwa dem Vorgänger CCS, in dem das Mitteilen eines Kanalnamens nicht vorgesehen war.

Im Pi-Kalkül ist es möglich, über *Kanäle* Nachrichten zu versenden als auch das diese selbst Nachrichteninhalt sind (deren Kanalname) , also selber empfangen oder gesendet werden können. Somit repräsentieren Namen sowohl Nachrichtenkanäle als auch gesendete Daten und ermöglichen einen Modellierungsansatz für Mobilität, da nunmehr auch Kommunikationskanäle selbst zwischen Prozessen kommuniziert werden können. Es stehen damit neben gewöhnlichen Datenwerten und Variablen auch Namen als übertragbare Objekte zur Verfügung[1].

Abbildung 3.1 stellt anschaulich dar, wie sich Kommunikation vorgestellt werden kann. Ausgangssituation ist ein Zugriff auf einen Drucker in einem bestehenden Netzwerk. Zu Beginn hat lediglich der *Server* über *Kanal a* sowohl Verbindung mit dem Drucker als auch über *Kanal b* eine Verbindung mit einem *Client*. Durch die oben beschriebene Kommunikation der Verbindungen ist es dem Client möglich, nach der Übertragung der Verbindung vom Server, den Drucker zu nutzen
Aus allgemeiner Sicht steht somit einem Prozess (hier dem Client) nach dem Empfang eines Kanalnamens dieser zur Nutzung bereit, was wiederum einer Veränderung der Verbindungsstruktur gleichkommt [Nes98].

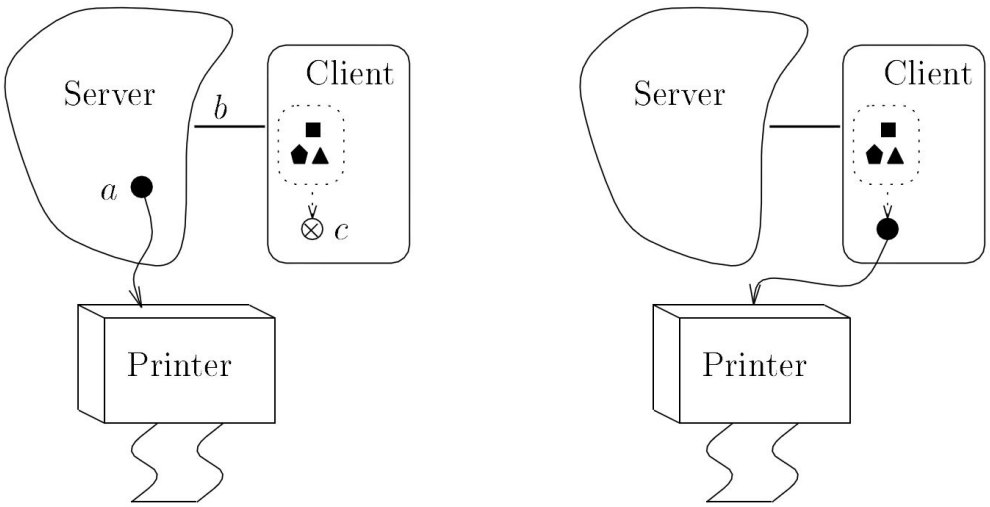
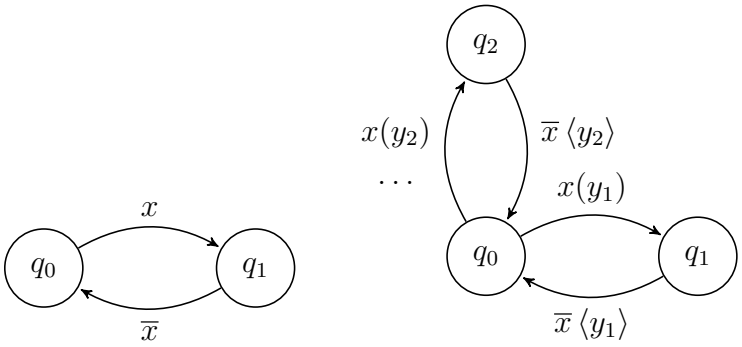


Abbildung 3.1: Link Passing Mobility: Vor und nach der Interaktion[Par01]

1 In mancher Literatur auch *Link Passing Mobility* genannt [Puh07].

Kommen wir auf das Mobieltelefonbeispiel zurück. Mit seiner Hilfe kann ein solch Verhalten sich ändernder Verbindungen im Pi-Kalkül wie folgt wiedergefunden werden. Das Löschen eines Kanals x und das Empfangen eines anderen Kanals y lässt sich hier als Bewegung eines Objektes von einem Ort x zu einem anderen Standpunkt y auffassen. Hierbei kommunizieren alle Objekte am Ort x über Kanal x und alle am Ort y sich befindenden Objekte über Kanal y. Mit dieser Erweiterung kann nun Kommunikation sich dynamisch verändernder Kommunikationsstrukturen abgebildet werden. Dies führt dazu, dass die Beschreibung des Pi-Kalküls – anders als in CCS – i.Allg. nicht über ein statisches Transitionssystem erfolgen kann. Der potentiell unendlichen Zahl an möglichen *Namen* müsste mit unendlich vielen Transitionsregeln Rechnung getragen werden. Vergleiche hierzu. Abbildung 3.2, in der diesem Umstand mit den ggf. unendlich vielen Kanalnamen (hier $y_1, y_2, \cdots$) Rechnung getragen wird.



(a) CCS **(b)** Pi-Kalkül

Abbildung 3.2: Unterschied der Transitionen in CCS zum Pi-Kalkül

Da in obiger Abbildung der Syntax des Pi-Kalküls teilweise vorgegriffen wurde, wird dieser nun neben weiteren Grundlagen im nachfolgenden Abschnitt vorgestellt.

3.1 Syntax des Pi-Kalküls

Im Folgenden sei $\mathcal{L} = \{u,v,w,x,y,z,u',v',w',x',y',z',u_1,v_1,w_1,x_1,y_1,z_1,...\}$ eine abzählbar unendliche Menge an Namen[2] sowie $\mathcal{K}$ die Menge der Prozesse. Weiterhin existiert die Menge $\overline{\mathcal{L}} = \{\overline{x} \mid x \in \mathcal{L}\}$. Die Elemente von $\mathcal{L}$ und $\overline{\mathcal{L}}$ entsprechen möglichen Aktionen. Hierbei stellen x und $\overline{x}$ zueinander komplementäre Aktionen dar, die das Empfangen bzw. Senden einer Nachricht beschreiben.

Der Syntax des *synchronen* Pi-Kalküls ist wie nachfolgend definiert:

2 Auch engl. *Links* genannt.

Definition 3.1 (Der Pi-Kalkül-Syntax)
Die Menge $\mathcal{K}$ der Pi-Kalkül-Prozesse ergibt sich aus:

$\mathbf{0}$	inaktiver Prozess
$P\|Q$	parallele Komposition
$x(y).P$	Input-Prefix
$\overline{x}\langle y\rangle.P$	Output-Präfix
$\nu x.P$	Restriktion
$!P$	Replikation

Wobei P,$Q \in \mathcal{K}$ und x,$y \in \mathcal{L}$.

Es sei an dieser Stelle nochmals angemerkt, dass in dieser Diplomarbeit lediglich eine Variante des *synchronen* Pi-Kalküls betrachtet wird und hierüber hinaus noch weitere Varianten bzw. Erweiterungen existieren. Eine kurze Beschreibung des *asynchronen* Pi-Kalküls wird in Abschnitt 3.3 gegeben.

In Anlehnung an Abschnitt 2.3.1, lassen sich also alle Prozessausdrücke des Pi-Kalküls mit folgender Grammatik darstellen:

$$P ::= \quad \mathbf{0} \quad | \quad \pi.P \quad | \quad \nu x.P \quad | \quad !P \quad | \quad P \,|\, P$$

Der Aktionspräfix π stellt sich gemäß folgender Grammatik dar:

$$\pi ::= \quad x(y) \quad | \quad \overline{x}\langle y\rangle$$

Damit die Übersicht bei komplexen, aus vielen Komponenten zusammengesetzten Prozessen gewahrt bleibt, sei an dieser Stelle festgelegt, dass die definierten Operatoren von links nach rechts abnehmend stark binden.

$$\left.\begin{array}{c}\text{Präfix}\\ \text{Restriktion}\end{array}\right\} \quad > \quad \text{Replikation} \quad > \quad \text{Composition}$$

Dies hat zur Folge, dass beispielsweise der Prozess

$$(\overline{x}\langle y\rangle.\mathbf{0} \,|\, !x\,(z)\,\mathbf{0}) \quad | \quad \overline{u}\langle v\rangle.\mathbf{0}$$

dem Prozess

$$((\overline{x}\langle y\rangle.\mathbf{0}) \,|\, !\,(x(z).\mathbf{0})) \quad | \quad (\overline{u}\langle v\rangle.\mathbf{0})$$

entspricht.

Eine informelle Erläuterung der einzelnen Pi-Kalkül Prozesse, welche in dieser Arbeit

stets durch die Buchstaben P und Q aus der Menge $\mathcal{K}$ Verwendung finden, wird im Folgenden gegeben:

- $\mathbf{0}$ stellt einen untätigen Prozess dar, der weder zur Kommunikation noch zu einem anderen Verhalten fähig ist.

- $\pi.P$ bestimmt die Aktionsmöglichkeiten eines Prozesses P durch das Präfix π. Hierbei sind, wie in der Grammatik zu erkennen, unterschiedliche Aktionen möglich:
  - $\overline{x}\langle y\rangle$ (Output) erlaubt das synchronisierte Senden des Namens y über den Kanal x. Im Anschluss verhält sich der Prozess gemäß der Definition des Prozesses P. Synchronisation bedeutet hierbei, dass eine solche Aktion nur dann möglich ist, wenn zu einem Prozess mit Output-Präfix auch ein parallel ausgeführter Prozess vorhanden ist, der mit einem Input-Präfix beginnt, d.h. der einen Namen über den entsprechenden Kanal empfangen kann.
  - $x(y)$ (Input) erlaubt das synchronisierte Empfangen des Namens y über den Kanal x, wobei alle Vorkommen von y durch den empfangenen Namen ersetzt werden. y dient hier lediglich als Platzhalter für jeden Namen, den der Prozess über Kanal x empfängt. Im Anschluss steht der empfangene Name für weitere Aktion innerhalb des Prozesses P zur Verfügung.

    Beispiel 3.1 (Verknüpfung von Aktionen)
    Mehrere Aktionen lassen sich mit einem „·“ verknüpfen und kennzeichnen eine sequentielle Ausführung.

    $$\overline{x}\langle y\rangle.\overline{y}\langle z\rangle.P$$

    Hierbei wird zunächst y über Kanal x und anschließend z über y gesendet, bevor Prozess P fortgesetzt wird.

- $\nu x.P$ wird als Restriktion bezeichnet. Hierbei wird der Name x auf den Prozess P begrenzt. Ein Senden von Namen an andere Prozesse ist über den Kommunikationskanal x nicht möglich. Lediglich innerhalb des Prozesses P darf dieser Kanal genutzt werden. Aus diesem Grund wird x auch als *privat/lokal* in P bezeichnet. Es ist jedoch möglich, den Gültigkeitsbereich eines privaten Namens zu erweitern. Hierzu muss lediglich der private Name einem Prozess außerhalb des Gültigkeitsbereiches über einen nicht privaten Kanal kommuniziert werden. Solch ein Vorgang wird als *Scope Extrusion* (Definition 3.7) bezeichnet.

- $P\,|Q$ ist die parallele Komposition. Zwei Prozesse P und Q laufen gleichzeitig und unabhängig voneinander ab, können jedoch über gemeinsame Kanäle miteinander kommunizieren.

- $!P$ stellt die Replikation dar. „!“ erlaubt es, unendliches Verhalten zu formalisieren. Dabei steht $!P$ für eine unbegrenzte Zahl an parallel laufenden Prozessen

$P \mid P \mid ... \mid P$ von Prozess P. Diese Vervielfachung kann von Nutzen sein, wenn beispielsweise ein System auf eine Vielzahl von eingehenden Nachrichten reagieren muss, ohne diese zu blockieren, weil gerade eine Anfrage bearbeitet wird. Sodann kann ein neuer Prozess instanziiert werden, der diese dann bearbeitet.

Bevor weitere grundlegende Definitionen des Pi-Kalküls erfolgen, nachfolgend einige Beispiele möglicher Kommunikation.

Beispiel 3.2 (Kommunikation)
Bei Prozess

$$\overline{x}\langle y\rangle.\mathbf{0} \mid x(z).\overline{x'}\langle z\rangle.\mathbf{0}$$

können beide Komponenten miteinander kommunizieren. $\overline{x}\langle y\rangle.\mathbf{0}$ sendet den Namen y an die zweite Komponente, in der so sämtliche Vorkommen von z dadurch ersetzt werden. Dies führt zu Prozess

$$\mathbf{0} \mid \overline{x'}\langle z\rangle.\mathbf{0},$$

welcher den empfangenen Namen über Kanal x' senden könnte, wenn ein weiterer Interaktionspartner für diesen Kanal vorhanden wäre.

Anmerkung: In einigen Literaturstellen wird im Zusammenhang mit den Aktionspräfixen auch τ als Präfix genannt, der an dieser Stelle nur kurz Erwähnung finden soll, jedoch in weiteren Betrachtungen keine Berücksichtigung findet. Der τ-Präfix beschreibt nicht beobachtbare Aktionen *(Silent Actions)*. Hierbei handelt es sich um Aktionen, die von *außen* nicht zu erkennen sind, also jene, die keine sichtbaren Operationen über ihren Prozessrumpf hinaus kommunizieren (vgl. Beispiel 3.3).

Beispiel 3.3 (Silent Action)
Zwei parallele Prozesse P und Q können eine, bezogen auf den zusammengesetzten Prozess $P \mid Q$, *stille Aktion* erzeugen.

$$P = x(y).P' \quad \text{und} \quad Q = \overline{x}\,\langle z\rangle\,.Q'$$

führt bei paralleler Ausführung zu

$$P \mid Q \xrightarrow{\tau} P' \mid Q'$$

Da dieser zusammengesetzte Prozess nun nicht mehr mit andern Prozessen kommunizieren muss, lässt sich diese Reduktion durch die *stille Aktion* kennzeichnen.

In der vorangegangenen informellen Erläuterung der einzelnen Konstrukte des Pi-Kalküls wurde bereits von Gültigkeitsbereichen gesprochen. In diesem Zusammenhang ist es nötig, um weitere syntaktische Untersuchungen betreiben zu können, zunächst die Begriffe der gebundenen sowie freien Vorkommen von Namen zu definieren.

Definition 3.2 (Namensbindung)
In Prozessen der Form $x(y).P$ und $\nu y.P$ bezeichnen wir alle Vorkommen der Namen y als *gebunden*. P heißt hierbei *Gültigkeitsbereich* der Restriktion bzw. des Input-Präfixes. Namen, die in keinem Geltungsbereich gebunden sind, heißen *freie* Namen. $fn(P)$ bezeichnet die Menge der Namen, die in P wenigstens ein freies Vorkommen haben, wobei $bn(P)$ die Menge der in P gebundenen Namen bezeichnet.

Definition 3.2 führt zu folgender induktiv definierter Menge der freien Namen $fn(P)$:

$$\begin{aligned} fn(x(y).P) &= \{y\} \cup (fn(P) \setminus \{y\}) \\ fn(\overline{x}\,\langle y\rangle\,.P) &= \{x,y\} \cup fn(P) \\ fn(P_1 \mid P_2) &= fn(P_1) \cup fn(P_2) \\ fn(\mathbf{0}) &= \emptyset \\ fn(\nu x.P) &= fn(P) \setminus \{x\} \\ fn(!P) &= fn(P) \end{aligned}$$

Weiterhin zur induktiv definierten Menge der gebundenen Namen $bn(P)$:

$$\begin{aligned} bn(x(y).P) &= \{x\} \cup (fn(P) \\ bn(\overline{x}\,\langle y\rangle\,.P) &= bn(P) \\ bn(P_1 \mid P_2) &= bn(P_1) \cup bn(P_2) \\ bn(\mathbf{0}) &= \emptyset \\ bn(\nu x.P) &= \{x\} \cup bn(P) \\ bn(!P) &= bn(P) \end{aligned}$$

Alle Namen eines Prozesses werden als Vereinigung der *freien* und *gebundenen* Namen mit $n(P) := fn(P) \cup bn(P)$ bezeichnet.

Beispiel 3.4 (Gebundene/Freie Namen)

$$\underbrace{\overline{x}\langle y\rangle.0}_{x,y\ frei} \mid \underbrace{x(z). \quad \underbrace{\nu x.\overline{z}\langle x\rangle.0}_{x\ gebunden,\, z\ gebunden}}_{x\ frei,\, z\ gebunden}$$

Bei dieser parallel ablaufenden Komposition empfängt die zweite Komponente den Namen z und deklariert x als *lokal*, sodass sie den Zugriff auf das *globale/freie* x verliert.

In Beispiel 3.2 fand zum ersten Mal ein Einsetzen des empfangenen Namens y in den

Prozess $x(z).\overline{x'}\langle z\rangle.0$ für z statt. Diese Ersetzung in einen Prozess der Form $x(y).P$ folgt dem Prinzip der syntaktischen Substitution gemäß folgender Definition 3.3.

Definition 3.3 (Substitution)
Eine *Substitution* ist eine Funktion $\sigma : \mathcal{N} \mapsto \mathcal{N}$, die eine Menge von Namen auf eine Menge von Namen abbildet. Es sei

$$\sigma(x) = \begin{cases} y_i & \text{für } x = x_i \in \{x_1,...,x_n\}\,,\ n \in \mathbb{N}, \\ x & \text{sonst.} \end{cases}$$

Notiert wird die Substitution durch $\left\{{}^{y_1,...,y_n}/_{x_1,...,x_n}\right\}$ bzw. $\left\{{}^{y_1}/_{x_1},\ldots,{}^{y_n}/_{x_n}\right\}$, hierbei werden jedes Vorkommen von x_i durch y_i ersetzt als auch für jedes $x \notin \{x_1,...,x_n\}$ die identische Abbildung beschrieben. *Freie* Namen dürfen nicht gebunden werden. Die Substitution σ, angewendet auf einen Term x, wird durch $x\sigma$ bzw. $\sigma(x)$ angegeben, wobei sie folgenden Regeln unterliegt:

$$\begin{aligned} \mathbf{0}\sigma &\stackrel{def}{=} \mathbf{0} \\ (\pi.P)\sigma &\stackrel{def}{=} \pi\sigma.P\sigma \\ (P \mid P')\sigma &\stackrel{def}{=} P\sigma \mid P'\sigma \\ \nu x P\sigma &\stackrel{def}{=} \nu x P\sigma \\ (!P)\sigma &\stackrel{def}{=} !P\sigma \end{aligned}$$

Prozessoperatoren binden schwächer als eine Substitution, d.h. $\pi.P\sigma$ entspricht $\pi.(P\sigma)$. Darüber hinaus kann für jede Substitution σ der *Support* angegeben werden, der diejenigen Elemente spezifiziert, die durch die Substitution σ nicht auf sich selbst abgebildet werden; $supp(\sigma) = \{x \mid x\sigma \neq x\}$. Komplementär hierzu ergibt sich der *Cosupport* als $cosupp(\sigma) = \{x\sigma \mid x \in supp(\sigma)\}$. $n(\sigma) = supp(\sigma) \cup cosupp(\sigma)$ definiert die Menge der an einer Substitution σ auftretenden Namen.

Bei der Anwendung einer Substitution σ ist darauf zu achten, dass keine zuvor *freien* Namen nach der Ersetzung gebunden sind. Würde beispielsweise im Prozess

$$P = x(y).\overline{y}\,\langle z\rangle\,.\mathbf{0}$$

nach der Anwendung der Substitution $\sigma = \{y/z\}$ sich fälschlicherweise der Prozess

$$P' = x(y).\overline{y}\,\langle y\rangle\,.\mathbf{0}$$

ergeben, so wäre zuvor der Name x in P frei, jedoch das entsprechende Vorkommen

des Namens y in P' gebunden. Diese Fehlern lassen sich jedoch vermeiden, indem problematische Namen wie hier y vor der eigentlichen Substitution in einen nicht verwendeten Namen konvertiert werden. Dies würde in diesem Fall zu Prozess

$$P'' = x(y').\overline{y'}\,\langle y\rangle\,.\mathbf{0}$$

führen. Diese Umbenennung wird als *α-Konversion* oder *α-Äquivalenz* bezeichnet.

Definition 3.4 (α-Äquivalenz)
Zwei Prozesse P und Q heißen α-äquivalent ($P =_\alpha Q$) genau dann, wenn Q in einer endlichen Zahl an Substitutionen von gebundenen Variablen aus P hervorgehen kann.
Eine Ersetzung von gebundenen Namen in einem Prozess P ist die Ersetzung eines Teilterms;

- $x(y).Q$ von P durch $x(z).Q\,[z/y]$, oder
- $\nu y.Q$ von P durch $\nu z.Q\,[z/y]$ mit $z \notin n(Q)$.

Nachfolgendes Beispiel soll nochmals die Notwendigkeit von Umbenennungen verdeutlichen.

Beispiel 3.5 (α-Äquivalenz)
Prozess P und Q lassen sich durch Umbenennung ineinander überführen. Für

$$\nu x.\big(x(y).\overline{y}\,\langle z\rangle\,.\mathbf{0}\big) =_\alpha \nu a.\big(a(b).\overline{b}\,\langle c\rangle\,.\mathbf{0}\big)$$

kann somit $P =_\alpha Q$ geschrieben werden.

Beispiel 3.6 (α-Äquivalenz und Substitution)
Der Prozess

$$\underbrace{\nu z.\,[x(y).\overline{z}\,\langle y\rangle\,.\mathbf{0} \mid z(u).\mathbf{0}]}_{P_1} \mid \underbrace{\overline{x}\,\langle z\rangle\,.\overline{z}\,\langle y\rangle\,.\mathbf{0}}_{P_2}$$

kann nicht ohne Beachtung eventueller Umbenennungen, alle Interaktionen seiner Teilprozesse durchführt werden. Zunächst führt die Interaktion von P_1 und P_2 über Kanal x zu

$$\underbrace{\nu z.\,[\overline{z}\,\langle y\rangle\,.\mathbf{0}\,\{z/y\} \mid z(u).\mathbf{0}]}_{P_1'} \mid \underbrace{\overline{z}\,\langle y\rangle\,.\mathbf{0}}_{P_2'}$$

wobei nun beachtet werden muss, dass, bevor die Substitution $\{^z/_y\}$ durchgeführt werden kann, zunächst der gebunden vorkommende Name z umzubenennen ist, beispielsweise in z'.

$$\underbrace{\nu z'.\left[\overline{z'}\langle y\rangle.\mathbf{0} \mid z'(u).\mathbf{0}\right]}_{P_1''} \mid \underbrace{\overline{z}\langle y\rangle.\mathbf{0}}_{P_2''}$$

Da in P_2'' der Name z frei vorkommt, bleibt er von der Umbenennung unberührt.

Konvention 3.5
Von nun an sei für jeden Prozess P stets angenommen, das dessen gebundene Namen paarweise verschieden sind, als auch das $fn(P) \cap bn(P) = \emptyset$ gilt.

Es ist mitunter nicht unproblematisch, auf einer konsequente Anwendung von Konvention 3.5 zu bestehen. Bei Beweisen, in denen komplexere Prozessausdrücke auftreten, erfordert die Einhaltung von Konvention 3.5 die Verwendung einer großen Zahl an verschiedenen Symbolen für einen einzelnen Namen. Da dies jedoch erfahrungsgemäß der Übersichtlichkeit und Lesbarkeit von Beweisen nicht zuträglich ist, wird folgende Ausnahmeregelung getroffen:

Konvention 3.6
Bestehen bei der Verwendung ein und desselben Namens zur Formalisierung mehrerer Bindungen in der Umgebung mathematischer Kontexte ***keine*** *Mehrdeutigkeiten, d.h., zu jeder Zeit ist es für jeden Namen eindeutig, wie und wo er gebunden ist, so ist in Abschwächung zu Konvention 3.5 eine Mehrfachbenutzung von Namen zulässig.*

Eine Erweiterung des Gültigkeitsbereichs von lokal gebundenen Namen ist, wie zuvor schon erwähnt, möglich. Somit können auch Prozesse auf Namen zugreifen, die zunächst nicht zu ihrem Gültigkeitsbereich gehörten [Mil92].

Definition 3.7 (Scope Extrusion)
Der Gültigkeitsbereich eines lokalen Namens x mit $\nu x.P$ kann auf einen Prozess Q mit $x \notin fn(Q)$ erweitert werden, indem der Name über einen von P und Q geteilten und von x verschiedenen Kanal kommuniziert wird. Hierbei müssen Prozesse P und Q über passende Ein- bzw. Ausgabe-Präfixe für den geteilten Kanal verfügen.

Beispiel 3.7 (Scope Extrusion)
Lokale Namen können ihren Bindungsbereich verlassen. In Prozess

$$x(y).P \mid \nu z.\overline{x}\langle z\rangle.Q \qquad \text{mit } z \notin n(P),$$

kann z über den beiden Prozessen bekannten Kanal x kommuniziert werde. Dies führt zu Prozess

$$\nu z.(P\{^z/_y\} \mid Q).$$

Für weitere und vor allem auch ausführlichere Erläuterungen zum Verhalten der *Scop-Extrusion* sei an dieser Stelle die Literatur von [Mil92] empfohlen, in der anhand einiger Beispiele die Bedeutung von expandierenden Gültigkeitsbereichen erklärt wird. Zusätzlich wird dem Leser auch grafisch ein solches Verhalten nähergebracht.

3.1.1 Strukturelle Kongruenz im Pi-Kalkül

Nachdem bisher beschrieben wurde, welche syntaktischen Möglichkeiten zur Modellierung von Prozessen im Pi-Kalkül gegeben sind, so ist eine genauere Betrachtung der Prozesse selbst Ziel dieses Abschnitts. Bevor auf die semantischen Eigenschaften des Pi-Kalküls eingegangen wird, sollen an dieser Stelle syntaktische Untersuchungen angestellt werden.
Die strukturelle Kongruenz als rein syntaktische Äquivalenzrelation auf Prozessausdrücken, bietet die Möglichkeit, potentielle Reaktion zwischen Prozessen auch stattfinden zu lassen. Ziel hierbei ist es, eine formale Beschreibung für *intuitiv* gleiche Prozesse, also für Prozesse, die sich trotz syntaktischer Unterschiede durch gleiches Verhalten auszeichnen, zu erlangen. Beispielsweise mag es sinnvoll sein bei gleichberechtigten Teilprozessen diese untereinander umzuordnen, ohne dass dies Einfluss auf das Verhalten des Prozesses im Ganzen hat. Ebenso wenig ist es von Interesse, dass das Weglassen von inaktiven bzw. untätigen Prozessen das Verhalten des gesamten Systems beeinflusst. Fortan heißen solch syntaktisch leicht veränderten Prozesse *strukturell kongruent (geschr. $\equiv$)*, wenn sie sich als α-äquivalent erweisen und anhand der in Definition 3.15 gegebenen Regeln ineinander überführen lassen.

Da Prozesse nicht „alleine auf weiter Flur“ vorkommen, sondern sich zumeist in „Gesellschaft“ von weiteren Prozessen befinden, erfolgt zunächst die Definition des Kontextes (Definition 3.8), um hierauf aufbauend und unter Zuhilfenahme der allgemeinen *Prozesskongruenz (Definition 3.13)* die bereits angesprochene *strukturelle Kongruenz (Definition 3.15)* zu definieren.

Definition 3.8 (Prozesskontext)
Ein Prozesskontext C ist definiert gemäß der Grammatik:

$$C := [\,] \mid \pi.C \mid \nu x.C \mid !C \mid C|P \mid P|C \qquad \text{und } x \in \mathcal{N}$$

wobei π und P wie zuvor definiert sind. Insbesondere ist für $C = [\,]$ mit $C\,[Q] = Q$ der Identitätskontext definiert.

Ein Prozess-Kontext ist somit ein Prozessausdruck mit *Loch (notiert $[\,]$)*. $C\,[Q]$ bezeichnet den Prozessausdruck, der sich ergibt wenn Prozess Q in den Kontext C eingesetzt wird.

Bevor im Folgenden die zuvor angesprochenen Kongruenzen definiert werden, soll nachfolgende allgemeine Definition der *Relation* und der *Ordnung* helfen, später auftauchende Termini, sodenn sie Verwendung finden, besser verstehen zu können.

Definition 3.9 (Binäre Relation)
Eine *binäre Relation* R zwischen zwei Mengen X und Y ist eine Menge von Paaren (x,y) mit $x \in X$ und $y \in Y$, also $R \subseteq X \times Y$.

Statt $(x,y) \in R$ schreibt man xRy.

Ist $X = Y$, so ist R eine *binäre Relation auf der Menge* X.

$R^{-1} =^{def} \{(y,x) \mid (x,y) \in R\}$ ist die zu R *inverse Relation.*

Definition 3.10 (Besondere binäre Relationen)
Eine binäre Relation R auf X heißt:

reflexiv g.d.w $\forall x \in X : xRx$,

symmetrisch g.d.w. $\forall x,y \in X$: wenn xRy, dann yRx,

transitiv g.d.w. $\forall x,y,z \in X$: wenn xRy und yRz, dann yRx.

Definition 3.11 (Äquivalenzrelation)
Eine binäre Relation R auf einer Menge X ist eine *Äquivalenzrelation*, falls R reflexiv, symmetrisch und transitiv ist.

Ist R eine Äquivalenzrelation auf X und $x,y \in X$, so schreibt man statt xRy auch $x \sim_R y$ und sagt: x ist äquivalent zu y bezüglich R.

Definition 3.12 (Präordnung)
Eine binäre Relation R auf einer Menge X ist eine Präordnung (geschr. $\leq$), wenn sie transitiv und reflexiv ist.

Auf zuvor definierte Prozess-Kontexte zurückkommend, ist es nun möglich, eine *allgemeine Prozess-Kongruenz* zu definieren, die eine *Kongruenz-Relation (geschr. $\cong$)* über $\mathcal{K}$ definiert.

Definition 3.13 (Prozess-Kongruenz)
Sei $\cong$ eine Äquivalenzrelation auf der Menge $\mathcal{K}$, so heißt $\cong$ Prozess-Kongruenz, wenn für alle Prozessausdrücke P,Q und alle Kontexte C Folgendes gilt:

$$P \cong Q \Rightarrow C[P] \cong C[Q]$$

Eine Äquivalenzrelation $\cong$ ist somit kongruent genau dann, wenn die obige Bedingung erfüllt wird.

Wie im Fall der Äquivalenzrelation kann auch bei der Prozess-Kongruenz eine Präordnung definiert werden, die Präkongruenz.

Definition 3.14 (Präkongruenz)
Eine partielle Ordnung oder auch Präordnung ist eine *Präkongruenz*, wenn für alle Prozessausdrücke P, Q und alle Kontexte C gilt:

$$P \leq Q \Rightarrow C[P] \leq C[Q]$$

Nach dieser allgemeinen Beschreibung der Prozesskongruenz ist es möglich, bestimmte Prozess-Kongruenzen wie beispielsweise die *strukturelle Kongruenz* unter Zuhilfenahme eines Gleichungssystems $\mathcal{E}$ zu definieren. Hierbei ist $\cong$ die kleinste Prozess-Kongruenz, welche die nachfolgenden Bedingungen erfüllt:

- $\cong$ erfüllt $\mathcal{E}$,
- $Q_1 \cong Q_n$ für jede Folge von Prozessausdrücken $Q_1 ... Q_n$ mit $Q_i = C[P_i]$ und $Q_{i+1} = C[P_{i+1}]$ für alle $i \in \mathbb{N}$, wobei C ein Prozess-Kontext ist und $P_i \cong P_{i+1} \in \mathcal{E}$ oder $P_{i+1} \cong P_i \in \mathcal{E}$.

Es ist $Q \cong R$ genau dann, wenn sich Q durch wiederholte Anwendung einer Gleichung aus $\mathcal{E}$ auf beliebige Prozess oder Teilprozesse von Q nach P transformieren lässt.

Das Gleichungssystem, das die strukturelle Kongruenz bestimmt, ist nachfolgend definiert.

Definition 3.15 (Strukturelle Kongruenz)
Die strukturelle Kongruenz $\equiv$ ist definiert als die kleinste Prozess-Kongruenz auf P, die nachfolgende Axiome erfüllt.

$$
\begin{aligned}
P &\equiv Q, \text{ falls } P =_\alpha Q \\
P \mid \mathbf{0} &\equiv P \\
P_1 \mid (P_2 \mid P_3) &\equiv (P_1 \mid P_2) \mid P_3 \\
P_1 \mid P_2 &\equiv P_2 \mid P_1 \\
\nu u.\mathbf{0} &\equiv \mathbf{0} \\
\nu u.\nu v.P &\equiv \nu v.\nu u.P \\
\nu u.(P_1 \mid P_2) &\equiv P_1 \mid \nu u.P_2, \text{ falls } u \notin fn(P_1) \\
!P &\equiv P \mid !P
\end{aligned}
$$

Bemerkungen zur strukturellen Kongruenz:

- Namen gebundener Variablen können jederzeit umbenannt werden. Hierbei sei zusätzlich noch einmal erwähnt, dass bei einem Prozess $x(y).P$ die Variable y als Binder fungiert.
- Die Reihenfolge in einer parallelen Komposition von Prozessen spielt ebenfalls keine Rolle. Es ist somit möglich, Prozesse, die über einen gemeinsamen Kanal verfügen, so umzuordnen, dass diese direkt nebeneinanderstehen, um so eventuelle Interaktionen leichter deutlich werden zu lassen.
- Bei $\nu u.(P_1 \mid P_2) \equiv P_1 \mid \nu u.P_2$ kann das Kommunizieren nach *außen* eines zunächst *privaten* Kanals zu Problemen führen (vgl. Scope Extrusion (Definition 3.7)). Durch passende Umbenennung – ohne Gültigkeitsbereiche in P zu verletzen – kann hier dennoch erreicht werden, dass die neue Bindung nach außen gezogen werden kann, oder umgekehrt einem bestimmten Prozess zugeordnet wird.

Im Vorfeld zu einem Beispiel von kongruenten Prozessen wird zunächst nachfolgendes Lemma zur strukturellen Kongruenz bewiesen und als Beleg dafür dienen, dass weitere Ableitungen von Eigenschaften aus der Definition der strukturellen Kongruenz problemlos möglich sind.

Lemma 3.16
Es gilt: $\nu x.Q \equiv Q$, falls $x \notin fn(Q)$.

Beweis. Für Prozess Q mit $x \notin fn(Q)$ ist:

$$\nu x.Q \equiv \nu x.Q \mid \mathbf{0} \equiv Q \mid \nu x.\mathbf{0} \equiv Q \mid \mathbf{0} \equiv Q\,. \qquad \square$$

Beispiel 3.8 (Strukturell kongruente Prozesse)
Die hier aufgeführten Prozesse erweisen sich als zueinander kongruent.

$$\begin{aligned}
& u(v).v(w).\mathbf{0} \mid !(\nu v)\big(v(x).\mathbf{0} \mid \overline{v}\,\langle w\rangle\,.\mathbf{0}\big) \\
\equiv\ & u(v).v(w).\mathbf{0} \mid (\nu v)\big(v(x).\mathbf{0} \mid \overline{v}\,\langle w\rangle\,.\mathbf{0}\big) \mid !(\nu v)\big(v(x).\mathbf{0} \mid \overline{v}\,\langle w\rangle\,.\mathbf{0}\big) \\
\equiv\ & u(v).v(w).\mathbf{0} \mid (\nu v')\big(v'(x).\mathbf{0} \mid \overline{v'}\,\langle w\rangle\,.\mathbf{0}\big) \mid !(\nu v)\big(v(x).\mathbf{0} \mid \overline{v}\,\langle w\rangle\,.\mathbf{0}\big) \\
\equiv\ & (\nu v')\big(u(v).v(w).\mathbf{0} \mid (v'(x).\mathbf{0} \mid \overline{v'}\,\langle w\rangle\,.\mathbf{0}\big) \mid !(\nu v)\big(v(x).\mathbf{0} \mid \overline{v}\,\langle w\rangle\,.\mathbf{0}\big)
\end{aligned}$$

Besonderes Augenmerk gilt dem gebundenen Namen z, der bei jeder neuen Replikation der letzten Komponente erzeugt wird. Da dieser zu jedem vorherigen Auftreten von z different ist, muss darauf geachtet werden, dass eine Umbenennung (hier von z nach z') zu erfolgen hat, bevor evtl. weitere Aktionen durchgeführt werden können, um Probleme mit *freien* und *gebundenen* Namen zu vermeiden.

Es ist von Vorteil, zunächst ein gewisses strukturelles Verhalten von Prozessen in Kongruenzrelationen zu verpacken. So ist es möglich, die Reduktionsregeln eines Kalküls gering und einfach zu halten, da gewisse Umformungsschritte nicht mit zusätzlichen Reduktionsregeln erfasst werden müssen, sondern bereits durch die strukturelle Kongruenz mit berücksichtigt werden. Im nächsten Schritt geht es um die Beschreibung der möglichen Reduktionsregeln, die im nachfolgenden Kapitel ihre Definition finden.

3.2 Operationale Semantik des Pi-Kalküls

Wie sich ein System, repräsentiert durch Prozesse des Pi-Kalküls, über die Zeit verändert, wird durch die *Reduktionssemantik* definiert. Hierbei wird die Interaktion der einzelnen Prozesse untereinander durch die *Reduktionssemantik* unter Zuhilfenahme der bereits definierten *strukturellen Kongruenz* definiert. Die strukturelle Kongruenz ist insoweit hilfreich, als dass sie die Umordnung von miteinander agierenden Prozessen ermöglicht, sodass korrespondierende Input- und Output-Präfixe nebeneinander geschrieben werden und diese miteinander in Interaktion treten können. Somit ist es möglich, die tatsächliche *Reduktionssemantik* relativ kurz zu gestalten, da sich um keine Umordnungen bzw. die Reihenfolge der auftretenden Prozesse gesorgt werden muss.
Die Reduktions-Relationen setzen sich aus zwei Arten von Axiomen zusammen. Prozess P kann zu einem Prozess P' aufgrund eigener „interner“ Reduktionen transformiert werden oder Prozess P tritt in Interaktion mit einem weiteren Prozess aus demselben System. Dabei definiert die Reduktionssemantik nur das Verhalten eines Systems – unabhängig von seiner Umgebung.

Anmerkung: Im Gegensatz zu $\xrightarrow{\alpha}$ für CCS (Definition 2.3) wird im Pi-Kalkül eine Reduktionssemantik definiert, d.h. $\rightarrow$ ohne eine *Markierung*. In CCS war solch eine *Markierung* für die dort beschriebene Bisimulation (Abschnitt 2.4.1) notwendig. Für die später betrachtete kontextuelle Gleichheit im Pi-Kalkül genügt $\xrightarrow{\tau}$, was gerade $\rightarrow$ entspricht.

An dieser Stelle sei nochmals an Abschnitt 2.2 wie auch an Abschnitt 2.3 erinnert, da auch nachfolgend einzelne Regeln der Reduktionssemantik gemäß der Form

$$\frac{\textit{Vorbedingung}}{\textit{Schlussfolgerung}}$$

dargestellt werden, und bereits dort nähere Erklärung fanden.

Definition 3.17 (Reduktionssemantik)
Die (small-step)[3] Reduktion „$\longrightarrow$“ im Pi-Kalkül ist definiert durch:

(INTERACT) $$x(y).P \mid \overline{x}\langle z\rangle.Q \longrightarrow P\,[z/y] \mid Q$$

(PAR) $$\frac{P \to P'}{P \mid Q \to P' \mid Q}$$

(NEW) $$\frac{P \to P'}{\nu\, x.P \to \nu\, x.P'}$$

(STRUCT) $$\frac{Q \equiv P \quad \wedge \quad P \to P' \quad \wedge \quad P' \equiv Q'}{Q \to Q'}$$

Erfolgt die Reduktion in einem Schritt, so wird dies mittels „$\longrightarrow$“ dargestellt. Mit $\xrightarrow{*}$ wird die reflexiv-transitive Hülle von $\longrightarrow$ beschrieben.
Beachte:
Eine Reduktion unterhalb einer Replikation (!) sowie unterhalb eines Aktions-Präfixes (π) ist nicht definiert. Allerdings lässt sich ggf. der Restriktionsoperator durch Umstrukturierung der Prozesse untereinander verschieben (siehe strukturelle Kongruenz).

Bemerkungen zur Reduktionssemantik:

- (INTERACT) definiert hierbei die wesentliche Reduktionsregel, da sie die Kommunikation zweier parallel ablaufender Prozesse realisiert. Mittels Substitution wird der Name z bei dem empfangenden Prozess durch y ersetzt. Dies wird mitunter auch durch $\xrightarrow{int.}$ gekennzeichnet.
- (PAR) sowie (NEW) beschreiben die Möglichkeit zur Reduktion von Prozessen sowohl im Fall der parallelen Komposition als auch unter einer Restriktion.

3 Vgl. Abschnitt über *operationale Semantik* in Kapitel 2.1.1.

- (STRUCT) ermöglicht es, auch Prozesse zu reduzieren, zu denen sich strukturell kongruente Prozesse finden lassen, die selbst wieder, entsprechend der Reduktionssemantik, reduzierbar sind.

Nachdem nun die möglichen Transformationen im Pi-Kalkül soweit definiert wurden, sollen diese anhand einiger Prozesse exemplarisch dargestellt werden, sodass einzelne Regeln besser verstanden wie auch nachvollzogen werden können. Im weiteren Verlauf dieser Diplomarbeit kann somit darauf verzichtet werden, explizit auf jeden einzelnen Umformungsschritt und dessen Konsequenz für das jeweilige Prozessverhalten einzugehen.

Beispiel 3.9
Es sei Prozess Q gegeben mit

$$Q = u(v).\overline{v}\,\langle w\rangle\,.\mathbf{0} \mid \nu x.\big(x(y).P \mid \overline{u}\,\langle x\rangle\,.R\big)$$

und den nicht näher definierten Prozessen P und R. Es ist nun möglich, durch Anwendung der Regeln der strukturellen Kongruenz wie auch der Reduktionsregeln den Prozess $Q \longrightarrow Q'$ abzuleiten, wobei Prozess Q' syntaktisch wie folgt aufgebaut ist:

$$Q' = \nu x.\big(x(y).P \mid \overline{x}\,\langle w\rangle\,.\mathbf{0} \mid R\big).$$

Gemäß Definition 3.7 kann innerhalb von Prozess Q eine Kommunikation der beiden Teilprozesse über den sich teilenden Kanal p ermöglicht werden, sodass eine weitere Reduktion $Q' \longrightarrow Q''$, mit

$$Q'' = \nu x.(R \mid P\,\{^{v}/_{y}\}),$$

unter erneuter Zuhilfenahme der strukturellen Kongruenz erfolgen kann.

Beispiel 3.10
Gegeben sei Prozesse P mit:

$$P \equiv \underbrace{x(y).\overline{y}\,\langle y\rangle\,.\mathbf{0}}_{P_1} \mid \underbrace{\overline{x}\,\langle z\rangle\,.\mathbf{0}}_{P_2} \mid \underbrace{z(w).\mathbf{0}}_{P_3}$$

Hieraus ergeben sich folgende Kommunikationsmöglichkeiten:

- P_1 und P_2 können über Kanal x Nachricht z kommunizieren:

$$P \rightarrow \underbrace{\overline{z}\,\langle z\rangle\,.\mathbf{0}}_{P_1'} \mid \underbrace{\mathbf{0}}_{P_2'} \mid \underbrace{z(w).\mathbf{0}}_{P_3} \equiv P'$$

- Nun können P_1' und P_3 über Kanal z Nachricht w kommunizieren:

$$P' \rightarrow \mathbf{0} \mid \mathbf{0} \mid \mathbf{0} \equiv P''$$

- Somit ist Prozess P'' strukturell kongruent zu 0.

Beispiel 3.11
Gegeben sei Prozesse P mit:

$$P \equiv \underbrace{x(y).\mathbf{0}}_{P_1} \mid \underbrace{\nu x(\overline{z}\,\langle x\rangle\,.\mathbf{0})}_{P_2} \mid \underbrace{z(v).\overline{v}\,\langle w\rangle\,.\mathbf{0}}_{P_3}$$

Aufgrund der Bindung des Namens x durch den ν-Binder ist zunächst keine direkte Kommunikation möglich. Erst durch die strukturell kongruente Erweiterung des Bindungsbereichs des Namens x von P_2 mit P_3 ist nachfolgender Reduktionsschritt möglich.

$$\frac{\dfrac{\text{(NEW)} \qquad x(y).\mathbf{0} \mid \nu x(\overline{z}\,\langle x\rangle\,.\mathbf{0}) \mid z(v).\overline{v}\,\langle w\rangle\,.\mathbf{0} \longrightarrow x(y).\mathbf{0} \mid \nu x(z(v).\overline{v}\,\langle w\rangle \mid \overline{z}\,\langle x\rangle\,.\mathbf{0})}{\text{(REACT)} \qquad x(y).\mathbf{0} \mid \nu x(z(v).\overline{v}\,\langle w\rangle \mid \overline{z}\,\langle x\rangle\,.\mathbf{0}) \longrightarrow x(y).\mathbf{0} \mid \nu x(\overline{x}\,\langle w\rangle\,.\mathbf{0} \mid \mathbf{0})}}{\text{(STRUCT)} \qquad x(y).\mathbf{0} \mid \nu x(\overline{x}\,\langle w\rangle\,.\mathbf{0} \mid \mathbf{0}) \longrightarrow \nu x'\big(x(y).\mathbf{0} \mid \overline{x'}\,\langle b\rangle\,.\mathbf{0}\big)}$$

$$\nu x'\big(x(y).\mathbf{0} \mid \overline{x'}\,\langle b\rangle\,.\mathbf{0}\big) \qquad \equiv \qquad P'$$

Wie oben zu sehen, ist es möglich, den Scope weiter auszudehnen. Jedoch verbietet die strukturelle Kongruenz das *Einfangen* von freien Namen, sodass eine Umbenennung erfolgen muss. So ergibt sich an dieser Stelle der irreduzible Prozess P'.

Wie schon mehrfach daraufhin gewiesen wurde, ist stets der *synchrone*-Pi-Kalkül Grundlage der im Umfang dieser Diplomarbeit gemachten prozessalgebraischen Untersuchungen. Das Präfix *synchron* lässt jedoch bereits vermuten, dass auch eine *asynchrone* Variante des Pi-Kalküls existiert, auf die nun kurz eingegangen wird.

3.3 Der asynchrone Pi-Kalkül

Der *asynchrone* Pi-Kalkül, vorgestellt von [Hon91, Hon92] und [Bou92] als Spezialfall des *synchronen* Pi-Kalküls, unterscheidet sich vom Letztgenannten lediglich dadurch, dass im asynchronen Pi-Kalkül dem Output-Präfix *nur* der inaktive Prozess $\mathbf{0}$ folgen darf. Somit ändert sich die Grammatik des asynchronen Pi-Kalküls gegenüber dem synchronen, wie folgt:

$$P ::= \quad \mathbf{0} \quad \mid \quad \overline{x}\langle y\rangle.\mathbf{0} \quad \mid \quad x(y).P \quad \mid \quad \tau.P \quad \mid \quad \nu x.P \quad \mid \quad !P \quad \mid \quad P \,|\, P$$

Die Einschränkung der Syntax des asynchronen gegenüber dem synchronen Pi-Kalkül hinsichtlich des Output-Präfixes führt im Falle des asynchronen Pi-Kalküls zu einer anderen Sichtweise der Kommunikation. Die „Asynchronität“ ergibt sich hierbei aus der Vorstellung, dass gesendete Nachrichten, also solche der Form $\overline{x}\langle y\rangle.\mathbf{0}$, im „virtuellen“ Raum, wie „*chemische* Moleküle“ umherschwirren [Bou92].

Während im synchronen Pi-Kalkül zum Senden einer Nachricht über einen Kanal auch immer ein passender Kommunikationspartner vorhanden sein und so evtl. auf

diesen ein Prozess warten muss, ist dies im Fall des asynchronen Pi-Kalküls nicht nötig. Dieses Verhalten ist dergestalt zu verstehen, dass die Regel (INTERACT) aus Definition 3.17 im asynchronen Pi-Kalkül so nicht berücksichtigt werden muss. Im asynchronen Pi-Kalkül findet eben nicht ein *gleichzeitiges* Senden und Empfangen einer Nachricht zwischen zwei Prozessen statt, sondern eine zu empfangende Nachricht wird aus der Menge der im virtuellen Raum „schwebenden“ Nachrichten passend ausgewählt und empfangen. Gesendete Nachrichten bestehen nur aus der Nachricht selbst, sowie dem dazu gehörenden Kanalnamen und können so jederzeit von korrespondierenden Prozessen empfangen werden.

Mitunter ist neben der Schreibweise $\overline{x}\langle y\rangle.\mathbf{0}$ für einen „sendenden Prozess“ auch $\overline{x}\langle y\rangle$ gebräuchlich. Aufgrund der Struktur der empfangbaren Nachrichten ist ein Empfangen einer solchen Nachricht mit ihrem gleichzeitigen entfernen aus der Menge aller vorhandenen Nachrichten vorstellbar.

Für weiterführende Erläuterungen des asynchronen Pi-Kalküls sei an dieser Stelle auf die Werke von [Hon91, Hon92, Bou92] sowie zusammenfassend auf [Hen07] und [Sab10a] verwiesen. Dort wird auch gezeigt, wie der *synchrone* in den *asynchronen* Pi-Kalkül kodiert wird bzw. synchrone Kommunikation im asynchronen Pi-Kalkül simuliert werden kann.

3.4 Prozessgleichheit im synchronen Pi-Kalkül

Wie bereits aus den Grundlagen in Abschnitt 2.4 bekannt, ist es von Interesse, Programme miteinander zu vergleichen. Hierbei liegt ein besonderes Augenmerk auf der Überprüfung von gleichem Programmverhalten. Im Zusammenhang mit CCS wurde in Abschnitt 2.4.1 das Konzept der Bisimulation erläutert und auch im Kontext des Pi-Kalküls eine gut untersuchte Möglichkeit zur Untersuchung von gleichem Programmverhalten darstellt [San01].
An dieser Stelle soll jedoch eine weitere Möglichkeit zur Untersuchung von *gleichen* Prozessen erfolgen. Bekannt bereits aus anderen Kalkülen – wie beispielsweise dem Lambda-Kalkül nach [Bar85] –, soll nun das *kontextuelle Verhalten* von Pi-Kalkül-Prozessen untersucht werden. Dies geschieht ähnlich wie in [SS03], bezogen auf Lambda-Ausdrücke. Der in diesem Zusammenhang übliche Gleichheitsbegriff ist der Begriff der *kontextuellen Gleichheit*, nach dem zwei Teilprozesse genau dann gleich sind, wenn sie an beliebiger Stelle eines Prozesses gegeneinander ausgetauscht werden können, ohne dass ein anderes Prozessverhalten festgestellt werden kann (in der Regel wird hier die Terminierung betrachtet).

3.4.1 Die Standardreduktion

Die in Kapitel 3.1.1 eingeführte Reduktionssemantik (Definition 3.17) wird zunächst leicht verändert. Die Regel (INTERACT) stellt die eigentliche Kommunikation zwi-

schen Prozessen dar. Im Hinblick auf das Prozessverhalten ist gewünscht, dass so wenig wie mögliche Regeln zu berücksichtigen sind, die eine Programmveränderung herbeiführen bzw. hierbei genannt werden müssen. Infolgedessen erfährt die Regel (INTERACT) eine leichte Veränderung.

Definition 3.18
Aus bekannter Regel (INTERACT):

$$x(y).P \,|\, \overline{x}\,\langle z\rangle\,.Q \longrightarrow P\,[{}^{z}\!/_{y}]\,|\,Q$$

sei (REACT):

$$D\,[x(y).P \,|\, \overline{x}\,\langle z\rangle\,.Q] \longrightarrow D\,[P\,[{}^{z}\!/_{y}]\,|\,Q]$$

die sich ergebende Regeln unter Verwendung von *Reduktionskontext* D mit Grammatik

$$D = [\,] \quad | \quad \nu\, x.D \quad | \quad D \,|\, P \quad | \quad P \,|\, D.$$

Unter Zuhilfenahme der neu definierten Regel (REACT) ist es nun möglich, die Standardreduktion wie folgt zu definieren:

Definition 3.19 (Standardreduktion)
Sei die Standardreduktion $\xrightarrow{(\text{SR})}$ bestimmt durch:

$$\frac{P \equiv P' \quad \wedge \quad P' \xrightarrow{(\text{REACT})} Q' \quad \wedge \quad Q \equiv Q'}{P \xrightarrow{(\text{SR})} Q}.$$

Der Vorteil dieser beiden neuen Definitionen besteht darin, dass es nun möglich ist, die beiden Regeln (PAR) und (NEW) der ursprüngliche Definition 3.17 nicht mehr zu beachten. Die Standardreduktion aus Definition 3.19 stellt sich als eine unter struktureller Kongruenz abgeschlossene (REACT)-Reduktion dar und berücksichtigt bereits die Regeln (PAR) sowie (NEW). Somit ist es möglich, Reduktionen auf Basis der Standardreduktion zu führen, was wiederum den Aufwand verringert, der nötig ist, um das kontextuelle Verhalten von Prozessen zu untersuchen.

Konvention 3.20
Im Hinblick auf die Reduktionssemantik des Pi-Kalküls, gelte von nun stets für „$\longrightarrow = \xrightarrow{(\text{SR})}$*“. Wobei „*$\longrightarrow$*“ gemäß Definition 3.17 definiert ist.*

3.4.2 Kontextuelle Äquivalenz

Da das Konzept der Bisimulation äquivalentes Verhalten von Prozessen dergestalt definiert, dass alle erreichbaren Zustände der zu vergleichenden Prozesse in der Lage sein müssen, sich gegenseitig zu simulieren, bisimulare Prozesse also dazu befähigt sind, alle Aktionen des anderen Prozesses zu simulieren, so ist das Ziel der *kontextuellen Programmäquivalenz*[4] ein anderes. Als kontextuell äquivalente Prozesse werden jene Prozesse identifiziert, deren Auswertung in allen möglichen Programmkontexten $C[\,]$ dasselbe *Terminierungsverhalten*[5] zeigt.
Alleine der Begriff der Terminierung mag beispielsweise im Lambda-Kalkül einheitlich definiert sein, in reaktiven Systemen wie dem Pi-Kalkül ist die einheitliche Definition eines terminierenden Prozesses ungleich schwieriger.
Reaktive Systeme zeichnen sich dadurch aus, dass sie auf ihre Umwelt reagieren sowie warten, bis eine passende Eingabe verarbeitet werden kann. Was soll also als Terminierung bei solch evtl. unendlich laufenden Prozessen, angenommen werden? Dieser Frage vorangestellt sei eine weitere nach einem *erfolgreich* abgearbeitet Prozess. Hierbei gilt zu berücksichtigen, dass Prozesse im Pi-Kalkül mitunter auf einen *INPUT* warten, bevor ein weiteres Abarbeiten möglich ist. Aus diesem Grund wird ein als *erfolgreich* zu identifizierender Prozess wie folgt definiert:

Definition 3.21 (Erfolgreicher Prozesse)
Prozess P ist *erfolgreich*, wenn gilt;

- P ist irreduzibel, d.h. es existiert kein Prozess Q mit $P \xrightarrow{(sr)} Q$ **und**
- P ist strukturell kongruent zu einem Prozess $\nu x_1,...,x_n.(x(u).P'|Q)$, für irgendeinen Namen x, wobei $x \neq x_i$ für $i = 0,...,n$ und $n \geq 0$.

Bemerkungen zur Definition von *erfolgreichen* Prozessen:

- Ein Prozess P heißt *irreduzibel*, wenn keine weiteren Reduktionsregeln mehr anwendbar sind. Er also nicht weiter zu einem *anderen* Prozess Q ausgewertet werden kann, der nicht selbst wieder strukturell kongruent, gemäß obiger Definition, zu sich selbst ist. Somit gelten all die Prozesse als erfolgreich, die entweder nicht weiter ausgewertet werden können und auf weitere Eingaben warten müssen oder aufgrund verschiedener Gültigkeitsbereiche keine weiteren Reduktionen mehr möglich sind.

Ein Beispiel für einen erfolgreichen Prozess ist nachfolgend gegeben.

4 Angelehnt an den Lambda-Kalkül nach [SS03].
5 Oft auch als konvergentes Verhalten bezeichnet.

Beispiel 3.12
Prozess P sei gegeben mit:

$$P = \underbrace{\overline{x}\langle y\rangle.u(v).\mathbf{0}}_{P_1} \mid \underbrace{x(w).\mathbf{0}}_{P_2} \mid \underbrace{\overline{x}\langle z\rangle.\mathbf{0}}_{P_3}.$$

P ist reduzibel und somit nicht *erfolgreich*. Nach Anwendung von (REACT) ergibt sich für P folgender Prozess P' mit:

$$P' = u(v).\mathbf{0} \mid \overline{x}\langle z.\rangle\,\mathbf{0},$$

da Teilprozess P_1 mit P_2 über Kanal x Nachricht y kommunizieren können und y für w in P_2 ersetzt wird. Im Anschluss daran besitzt Prozess P' einen *offenen Input* an Kanal u und ist zudem irreduzibel, was ihn somit zu einem erfolgreichen Prozess gemäß Definition 3.21 macht.

Nachdem nun definiert wurde, wann ein Prozess als *erfolgreich* identifiziert werden kann, gilt es nun zu definieren, wann ein Prozess auch *terminiert/konvergiert*; um weiter gehende kontextuelle Untersuchungen anstellen zu können.

Definition 3.22 (Prozesskonvergenz)
Ein Prozess P ist *konvergent* ($P\downarrow$),

- wenn es möglich ist, ihn mit einer beliebigen Folge von Standardreduktionen ($\xrightarrow{(sr,*)}$) in einen *erfolgreichen* Prozess zu überführen.

So gilt für P:

- $P\downarrow$, genau dann, wenn es einen Prozess Q gibt mit $P \xrightarrow{sr,*}$ Q und Q *erfolgreich* ist.

Anderenfalls ist $P\uparrow$ divergent.

Anmerkung: Obige Definition der Prozesskonvergenz wird auch als *May*-Konvergenz bezeichnet, da lediglich gefordert wird, dass ein Prozess konvergieren kann, jedoch *nicht* muss. Im Gegensatz zur *Must*-Konvergenz die gerade die Konvergenz fordert.

Ein Beispiel für einen zwar irreduziblen, aber nicht konvergenten Prozess ist im nachfolgenden Beispiel zu sehen.

Beispiel 3.13
Prozess P sei gegeben mit:

$$P = \overline{x}\,\langle y\rangle\,.u(v).\mathbf{0} \mid \mathbf{0}.$$

So ist dieser zwar *irreduzibel*, da es keine weiteren Reduktionsmöglichkeiten für P gibt, jedoch kann er auch nicht konvergieren, denn keiner seiner Komponenten besitzt einen offenen Input. Somit kann Prozesse P nicht in einen *erfolgreichen* Prozess überführt werden und infolge dessen konvergieren.

Die Untersuchung des beobachtbaren Verhaltens von Prozessen bzw. Prozesskontexten kann allgemein nach folgender Definition erfolgen.

Definition 3.23 (Kontextuelle Präordnung)
Seien P und Q Prozesse und C ein Kontext, so ist die kontextuelle Präordnung (geschr.($\leq_c$) wie folgt definiert:

$$P \leq_c Q \text{ genau dann wenn } \forall C : C\,[P] \Longrightarrow C\,[Q] \text{ also:}$$

$$C\,[P]\downarrow \Longrightarrow C\,[Q]\downarrow$$

Korollar 3.24
Die kontextuelle Präordnung[6] $\leq_c$ *ist eine Präkongruenz auf Prozessausdrücken.*

Beweis. Zu zeigen sind die Reflexivität, Transitivität, sowie die Kompatibilität mit allen Kontexten:

(Reflexivität)
$C\,[P]\downarrow \Longrightarrow C\,[P]\downarrow$

(Transitivität)
Seien P, P', Q Prozessausdrücke. Für alle Kontexte C mit $C\,[P]\,, C\,[P']\,, C\,[Q]$ gilt:
$C\,[P]\downarrow \Longrightarrow C\,[P']\downarrow \quad \wedge \quad C\,[P']\downarrow \Longrightarrow C\,[Q]\downarrow \quad \Longrightarrow \quad C\,[P]\downarrow \Longrightarrow C\,[Q]\downarrow$

(Kompatibilität)
Sei C ein beliebiger Kontext. Für alle Kontexte C_1 existiert ein Kontext C_2, mit $C_1\,[C\,[\,]] \equiv C_2\,[\,]$. Für jeden Kontext C_2 gelte: $C_2\,[P]\downarrow \Longrightarrow C_2\,[Q]\downarrow$. So gilt für alle Kontexte $C_1 : C_1\,[C\,[P]]\downarrow \Longrightarrow C_1\,[C\,[Q]]\downarrow$. □

Diese Ordnungsrelation ermöglicht es nun, die kontextuelle Äquivalenz wie folgt zu definieren.

6 Eine binäre Relation P heißt Präordnung, wenn P reflexiv und transitiv ist.

Definition 3.25 (Kontextuelle Äquivalenz)
Die kontextuelle Äquivalenz ($\sim_c$) ist:

$$P \sim_c Q \text{ genau dann, wenn } P \leq_c Q \wedge Q \leq_c P$$

Anmerkung: Die *kontextuelle Äquivalenz* stellt per Definition eine *Kongruenz* dar, d.h., ihre Gleichheiten sind selbst wiederum in Kontexte einsetzbar und erfüllen die Implikation $P \sim_c Q \Longrightarrow (\forall C : C[P] \sim_c C[Q])$.

Solche Gleichheiten sind mitunter nicht leicht nachzuweisen, da die *Terminierung* der Auswertung der Prozesse in allen möglichen Kontexten betrachtet werden muss.[7] Nachfolger Korollar dient der Schlussfolgerung in vielen späteren Beweisen der kontextuellen Äquivalenz.

Korollar 3.26
Es sei $\xrightarrow{r}$ *eine kompatible Reduktion auf Pi-Kalkülprozessen. Wenn für alle Prozesse* $P, Q \in \mathcal{K}$ *mit* $P \xrightarrow{r} Q$

$$P\downarrow \Longleftrightarrow Q\downarrow \text{ gilt,}$$

dann gilt auch für alle Kontexte C :

$$C[P]\downarrow \Longleftrightarrow C[Q]\downarrow .$$

Beweis. Offensichtlich – es folgt unmittelbar aus der Kompatibilität, dass $C[P] \xrightarrow{r} C[Q]$ gilt. □

Nachdem die allgemeinen Bedingungen zur Untersuchung der kontextuellen Äquivalenz von Prozessausdrücken im Pi-Kalkül gelegt wurden, schließt sich eine genauere Betrachtung der strukturellen Kongruenz und der kontextuellen Gleichheit an. Im Speziellen wird die Frage beantwortet, ob die strukturelle Kongruenz die kontextuelle Gleichheit erhält. Es wird die Behauptung überprüft, ob

$$P \equiv Q \Longrightarrow P \sim_c Q$$

gilt.

7 Im Allgemeinen ist die kontextuelle Gleichheit unentscheidbar, da mit der Beantwortung der Frage nach der Terminierung auch das Halteproblem gelöst würde.

Satz 3.27. *Die strukturelle Kongruenz erhält die kontextuelle Gleichheit.*

Beweis. Der Beweis gliedert sich in zwei Teile:

(1) Es reicht zu zeigen, dass für alle Prozesse P, Q gilt:

$$P \equiv Q \Longrightarrow (P\downarrow \Rightarrow\ Q\downarrow) \qquad (\textit{(A)nnahme})$$

Behauptung: Aus *(A)* folgt bereits, dass für alle Prozesse P, Q gilt:

$$P \equiv Q \Longrightarrow P \leq_c Q$$

Seien nun P und Q Prozesse für die $P \equiv Q$ gilt, so muss gezeigt werden, dass für alle Kontexte C gilt:

$$C\,[P]\downarrow \Longrightarrow\ C\,[Q]\downarrow$$

Ist nun C ein Kontext, sodass $C\,[P]$ *konvergiert*, so folgt unter Berücksichtigung der Definition von $\equiv$, die eine Kongruenz darstellt, $C\,[P] \equiv C\,[Q]$. Somit kann aus obiger *(A)nnahme* sofort

$$C\,[P]\downarrow \Longrightarrow\ C\,[Q]\downarrow$$

gefolgert werden.

(2) Zeige, dass die *(A)nnahme* gilt. Hierzu sei angenommen, dass für Prozess P gilt:

$$P \equiv Q \qquad \text{mit} \qquad P\downarrow,$$

d.h., es existiert eine Reduktionsfolge gemäß

$$P \xrightarrow{(sr,k)} P' \qquad \text{mit} \qquad P' \text{ ist } \textit{erfolgreich}$$

Fallunterscheidung für $k = 0$ und $k > 0$.

$k = 0$:
Die strukturelle Kongruenz ändert *erfolgreiche* Prozesse nicht, d.h., erfolgreiche Prozesse, die vor der Anwendung der Regel (STRUCT) erfolgreich waren, bleiben dies über die Regelanwendung hinaus. Es sei also Prozess P *erfolgreich* mit:

- $P \equiv \nu x_1,...,x_n.(x(u).P_0 \mid Q_0$ wobei $x \neq x_i$ (für $i = 0,...,n$ und $n \geq 0$ **und**
- P ist irreduzibel.

Behauptung: Da P irreduzibel ist, muss auch Prozess Q irreduzibel sein.

Beweis durch Widerspruch: Angenommen, Prozess P sei irreduzibel, aber es gelte $Q \xrightarrow{(sr)} Q'$. Dann ist jedoch $P \equiv Q \xrightarrow{(sr)} Q'$ ebenfalls eine Reduktion per Definition von $\xrightarrow{(sr)}$ und somit kann P nicht irreduzibel sein.

$k > 0$:
Für Prozess P gilt somit:

$$P \xrightarrow{(sr)} P_1 \xrightarrow{(sr,k-1)} P'$$

Da die Reduktion $P \xrightarrow{(sr)} P_1$ gültig ist und die Definition der Standardreduktion die Möglichkeit der Reduktion von $Q \xrightarrow{(sr)} P_1$ zulässt (es gilt $P \equiv Q$, egal ob $\equiv$ -Schritte vor, oder nach einer Standardreduktion angewendet werden) folgt sofort:

$$Q \xrightarrow{(sr)} P_1 \xrightarrow{(sr,k-1)} P'$$

d.h., Prozess Q konvergiert und die strukturelle Kongruenz beeinflusst die kontextuelle Gleichheit nicht. □

Nachfolgend werden einige „einfache“ kontextuelle Ungleichheiten bewiesen.

3.4.3 Nachweis „einfacher“ kontextueller Ungleichheiten

Die zuvor eingeführten Definitionen zur Untersuchung von Pi-Kalkül-Prozessen sollen nun dazu dienen, Prozessterme und Reduktionsregeln auf deren kontextuelle Gleichheit hin zu überprüfen.

Gibt es Prozesse – und wenn ja welche – die, wenn, sie sich in einem Kontext wiederfinden unterscheiden lassen oder aber sogar als „gleich“ anzusehen sind?
Konkret wird die Frage erörtert, welche Prozesse bzw. Prozesskonstrukte verträglich mit der vorangegangenen Definition der kontextuellen Gleichheit sind. Hierzu werden unterschiedlichste Prozesse, wie auch Reduktionsregeln, einer kontextuellen Prüfung unterzogen, um so kontextuelle Gleichheiten oder Ungleichheiten herauszuarbeiten.

Der Nachweis von sich kontextuell nicht gleich verhaltenden Prozessen geschieht i. Allg. so, dass versucht wird, **einen** Kontext C durch geschickte Wahl zu finden, der die zu untersuchenden Prozesse oder Reduktionsregeln als kontextuell nicht gleich identifiziert. Ist es mittels Kontext C (d.h. eingesetzt in dessen „Loch“) möglich, einen Prozesse als *konvergent* zu identifizieren, wobei der zweite Prozess stets divergiert, so konnte die kontextuelle Ungleichheit gezeigt werden.

Zur besseren Darstellung kontextuell sich unterscheidender Prozesse wird der Begriff der *korrekten* Programmtransformation eingeführt. Dieser dient dazu, gleiches Prozessverhalten exakter zu formulieren sowie nicht nur einzelne Prozesse, sondern „schematisch“ Prozesse zu vergleichen.

Definition 3.28 (Korrekte Programmtransformation)
Eine *Programmtransformation* ist eine binäre Relation auf Pi-Kalkül-Prozessen.

Eine Programmtransformation T ist *korrekt*, g.d.w. für alle Prozesse $(P, Q) \in T$ gilt:

$$P \sim_c Q\,.$$

Bei einigen Prozessen kann es möglich sein, diese bereits als „intuitiv“ ungleich zu erkennen. Allgemein ist es jedoch nicht immer sofort möglich. Dennoch müssen auch als „intuitiv“ ungleich erkannte Prozessterme noch formal daraufhin untersucht werden.

Als eine der ersten zu überprüfenden Regeln wird die (REACT)-Reduktion daraufhin getestet, ob sie mit dem Begriff der kontextuellen Äquivalenz verträglich ist, d.h. eine korrekte Programmtransformation darstellt.

Lemma 3.29
Die Regel (REACT) *ist im Allgemeinen keine korrekte Programmtransformation.*

Beweis. Für einen Prozess der Form $x(y).P \,|\, \overline{x}\,\langle z\rangle\,.Q$ gilt:

$$x(y).P \,|\, \overline{x}\,\langle z\rangle\,.Q \;\not\sim_c\; P\,[^z/_y] \,|\, Q$$

Mit $P = \mathbf{0}$ und $Q = \mathbf{0}$ d.h.

$$P_1 = x(y).\mathbf{0} \,|\, \overline{x}\,\langle z\rangle\,.\mathbf{0}$$
$$P_2 = \mathbf{0} \,|\, \mathbf{0} \;=\; \mathbf{0}$$

und unter Betrachtung des Kontextes $C = \overline{x}\,\langle w\rangle\,.u(v).\mathbf{0} \,|\, [\,]$ wird gezeigt, dass gilt:

$$C\,[P_1] = \overline{x}\,\langle w\rangle\,.u(v).\mathbf{0} \,|\, [x(y).\mathbf{0} \,|\, \overline{x}\,\langle z\rangle\,.\mathbf{0}] \text{ kann } \textit{konvergieren.}$$

Unter Verwendung der Regel (REACT) ist für P_1 folgende Reduktion möglich:

$$\frac{\text{(REACT)} \qquad x(y).\mathbf{0} \,|\, \overline{x}\,\langle z\rangle\,.\mathbf{0} \;\longrightarrow\; u(v).\mathbf{0} \,|\, \overline{x}\,\langle z\rangle\,.\mathbf{0}}{u(v).\mathbf{0} \,|\, \overline{x}\,\langle z\rangle\,.\mathbf{0}}$$

Der entstehende Prozess $u(v).\mathbf{0} \,|\, \overline{x}\,\langle z\rangle\,.\mathbf{0}$ besitzt einen offenen INPUT am Kanal u und ist zudem noch irreduzibel und somit gemäß Definition 3.22 konvergent.

Jedoch ist es für $C\,[P_2] = \overline{x}\,\langle w\rangle\,.u(v).\mathbf{0} \,|\, \mathbf{0}$ nicht möglich zu konvergieren. Der Prozess ist zwar nicht weiter reduzierbar, also irreduzibel, aber er besitzt keinen offenen INPUT, wie in Definition 3.22 gefordert.

Der Kontext C macht somit eine Unterscheidung der beiden Prozesse P_1 und P_2 möglich und daher ist die (REACT)-Regel im Allgemeinen nicht korrekt. □

Unterscheidet die kontextuelle Äquivalenz auch „intuitiv“ unterschiedliche und vor allem weniger komplexe Prozessterme? Hierzu nun weitere Ungleichheiten.

Lemma 3.30
Die kontextuelle Äquivalenz unterscheidet Prozessterme der Form $x(y).P$ *und* $\overline{x}\langle y\rangle.Q$.

Beweis. Die Prozesse $x(y).P$ und $\overline{x}\langle y\rangle.Q$ können mithilfe der kontextuellen Äquivalenz unterschieden werden.
Mit $P, Q = \mathbf{0}$ und Kontext $C = [\,] \mid \mathbf{0}$ ist offensichtlich, dass $C[x(y).\mathbf{0}]$ nicht weiter reduziert werden kann, als auch einen offenen INPUT an x hat und somit konvergiert. $C[\overline{x}\langle y\rangle.\mathbf{0}]$ ist zwar ebenfalls irreduzibel, jedoch nicht erfolgreich und somit divergent[8]. D.h., der Kontext C unterscheidet $x(y).\mathbf{0}$ und $\overline{x}\langle y\rangle.\mathbf{0}$ bezüglich der kontextuellen Äquivalenz. □

Die Replikation (!) als syntaktisches Konstrukt hat ebenfalls Einfluss auf die kontextuelle Äquivalenz von Prozesstermen. Wie sich solch Ungleichheiten aufgrund der Eigenschaften der Replikation ergeben, ist nachfolgend zu sehen.

Lemma 3.31
Reduktionen unterhalb einer Replikation verändern i. Allg. das kontextuelle Verhalten von Pi-Kalkül-Prozessen.

Beweis. Der Prozess $P_1 = !(x(y).P \mid \overline{x}\langle y\rangle.Q)$ ist kontextuelle von Prozesse $P_1' = !(P[y/z] \mid Q)$ verschieden. Es ist also zu zeigen, dass gilt:

$$!(x(y).P \mid \overline{x}\langle y\rangle.Q) \not\sim_c !(P[y/z] \mid Q)$$

Wähle nun $P = \mathbf{0}$ und $Q = u(v).\mathbf{0}$. Der Kontext sei mit $C = [\,]$ gewählt.

Für $C[P_1] = !(x(y).\mathbf{0} \mid \overline{x}\langle y\rangle.\mathbf{0}$ ist aufgrund der Replikation (!) eine unendliche Zahl an Reduktionen möglich.

$$\begin{aligned}
&!(x(y).\mathbf{0} \mid \overline{x}\langle y\rangle.\mathbf{0}) \\
&= (x(y).\mathbf{0} \mid \overline{x}\langle y\rangle.\mathbf{0}) \mid !(x(y).\mathbf{0} \mid \overline{x}\langle y\rangle.\mathbf{0}) \\
&\to u(v) \mid !(x(y).\mathbf{0} \mid \overline{x}\langle y\rangle.\mathbf{0}) \qquad \text{(REACT)} \\
&\ldots \\
&\to u(v) \mid u(v) \mid !(x(y).\mathbf{0} \mid \overline{x}\langle y\rangle.\mathbf{0}) \qquad \text{(REACT)} \\
&\ldots \\
&\to u(v) \mid u(v) \mid u(v) \mid !(x(y).\mathbf{0} \mid \overline{x}\langle y\rangle.\mathbf{0}) \qquad \text{(REACT)} \\
&\ldots
\end{aligned}$$

8 D.h. nicht konvergent.

Dies führt dazu, dass P_1 kein *erfolgreicher* Prozess ist.

Prozess P_1' kann jedoch in einen konvergenten Prozess reduziert werden. Unter Berücksichtigung des Kontextes C und P,Q wie oben, ist folgende Reduktion möglich:

$$\begin{aligned} C\,[P_1'] &= !(P\,[y/z] \mid Q) \\ &= !(\mathbf{0} \mid u(v).\mathbf{0}) \\ &= !(u(v).\mathbf{0}) \end{aligned}$$

Somit lässt sich P_1' in einen *erfolgreichen* Prozess (d.h. irreduzibel mit offenem INPUT an Kanal u) reduzieren, womit er Definition 3.22 erfüllt und so konvergiert.

Kontext C unterscheidet also Prozess P_1 von P_1', sodass für beide die kontextuelle Ungleichheit nachgewiesen werden konnte. Die Reduktion unter einer Replikation stellt somit i. Allg. keine korrekte Programmtransformation dar. □

Die nächste Ungleichung die auf Gültigkeit überprüft wird, stellt die *lokale Extrusion* von Prozessen unter Beobachtung.
Wie verhält es sich mit der kontextuellen Äquivalenz von Prozessen, wenn mittels *lokaler Extrusion* Namen nach „außen“ gesendet werden?

Lemma 3.32
Die lokale Extrusion erfüllt i.Allg. nicht die Definition der kontextuellen Äquivalenz.

Beweis. Es lässt sich zeigen, dass Prozess $\nu x.!\,P$ kontextuell verschieden zu Prozess $!\,\nu x.P$ ist, also die Ungleichung gilt:

$$\underbrace{\nu x.!\,P}_{P_1} \not\sim_c \underbrace{!\,\nu x.P}_{P_2}$$

Es gelte für $P = \overline{w}\,\langle x\rangle\,.\mathbf{0}$ und der betrachtete Kontext C sei wie folgt gewählt:

$$C = \underbrace{w(u).\overline{u}\,\langle v\rangle\,.y(z).\mathbf{0}}_{Q_1} \mid \underbrace{w(u).u(z).\mathbf{0}}_{Q_2} \mid [\,]$$

Wird nun P_1 mit $P = \overline{w}\,\langle x\rangle\,.\mathbf{0}$ in Kontext C eingesetzt, so ist

$$C\,[P_1] = w(u).\overline{u}\,\langle v\rangle\,.y(z).\mathbf{0} \mid w(u).u(z).\mathbf{0} \mid \nu x.\,(!\,\overline{w}\,\langle x\rangle\,.\mathbf{0})\ .$$

Der ν-Binder in Verbindung mit der Replikation verhindert zwar zunächst die direkte Kommunikation von P_1 mit den anderen Prozesstermen, jedoch ist es möglich, mittels *lokaler Extrusion* den Gültigkeitsbereich des Namens x so zu erweitern (da $x \notin n(Q_1,Q_2)$), dass weitere Kommunikationsmöglichkeiten in $C\,[P_1]$ möglich werden,

die sich wie folgt darstellen:

$$
\begin{aligned}
& w(u).\overline{u}\langle v\rangle.y(z).\mathbf{0} \mid w(u).u(z).\mathbf{0} \mid \nu x.(!\,\overline{w}\langle x\rangle.\mathbf{0}) \\
&= w(u).\overline{u}\langle v\rangle.y(z).\mathbf{0} \mid w(u).u(z).\mathbf{0} \mid \nu x.(\overline{w}\langle x\rangle.\mathbf{0} \mid !\,\overline{w}\langle x\rangle.\mathbf{0}) \\
&= w(u).\overline{u}\langle v\rangle.y(z).\mathbf{0} \mid w(u).u(z).\mathbf{0} \mid \nu x.(\overline{w}\langle x\rangle.\mathbf{0} \mid \overline{w}\langle x\rangle.\mathbf{0} \mid !\,\overline{w}\langle x\rangle.\mathbf{0}) \\
&\to \nu x.(w(u).\overline{u}\langle v\rangle.y(z).\mathbf{0} \mid w(u).u(z).\mathbf{0} \mid \overline{w}\langle x\rangle.\mathbf{0} \mid \overline{w}\langle x\rangle.\mathbf{0} \mid !\,\overline{w}\langle x\rangle.\mathbf{0}) \qquad \text{(NEW)} \\
&\to \nu x.(\overline{x}\langle v\rangle.y(z).\mathbf{0}\,[x/u] \mid x(z).\mathbf{0}\,[x/u] \mid \mathbf{0} \mid \mathbf{0} \mid !\,\overline{w}\langle x\rangle.\mathbf{0}) \qquad \text{(2x REACT)} \\
&\to \nu x.(y(z).\mathbf{0} \mid \mathbf{0}\,[v/z] \mid \mathbf{0} \mid \mathbf{0} \mid !\,\overline{w}\langle x\rangle.\mathbf{0}) \qquad \text{(REACT)} \\
&= \nu x.(y(z).\mathbf{0} \mid !\,\overline{w}\langle x\rangle.\mathbf{0})
\end{aligned}
$$

Prozess $\nu x.(y(z).\mathbf{0} \mid !\,\overline{w}\langle x\rangle.\mathbf{0})$ ist irreduzibel, besitzt einen offenen INPUT an y und ist somit konvergent.

Wird Prozess P_2 in Kontext C eingesetzt, so ist die Kommunikation eingeschränkt. Der strukturelle Aufbau von P_2 zeichnet sich dadurch aus, dass der ν-Binder nun für jede neu entstehende Instanz des replizierten Prozesses einen eigenen separaten, sich nicht überlappenden, Bindungsbereich für den Namen x erzeugt. Dies hat zur Folge, dass die beiden Prozesse Q_1 und Q_2 nun nicht mehr über denselben Kanal x kommunizieren können.
Formal stellt sich dies für $C\,[P_2] = w(u).\overline{u}\langle v\rangle.y(z).\mathbf{0} \mid w(u).u(z).\mathbf{0} \mid !\,\nu x.\overline{w}\langle x\rangle.\mathbf{0}$ wie folgt dar:

$$
\begin{aligned}
& w(u).\overline{u}\langle v\rangle.y(z).\mathbf{0} \mid w(u).u(z).\mathbf{0} \mid !\,\nu x.\overline{w}\langle x\rangle.\mathbf{0} \\
&= w(u).\overline{u}\langle v\rangle.y(z).\mathbf{0} \mid w(u).u(z).\mathbf{0} \mid \nu x_1.\overline{w}\langle x_1\rangle.\mathbf{0} \mid !\,\nu x.\overline{w}\langle x\rangle.\mathbf{0} \\
&= w(u).\overline{u}\langle v\rangle.y(z).\mathbf{0} \mid w(u).u(z).\mathbf{0} \mid \nu x_1.\overline{w}\langle x_1\rangle.\mathbf{0} \mid \nu x_2.\overline{w}\langle x_2\rangle.\mathbf{0} \mid !\,\nu x.\overline{w}\langle x\rangle.\mathbf{0} \\
&\to \nu x_1(\overline{x_1}\langle v\rangle.y(z).\mathbf{0}\,[x_1/u] \mid \overline{w}\langle x_1\rangle.\mathbf{0}) \mid \nu x_2(x_2(z).\mathbf{0}\,[x_2/u] \mid \overline{w}\langle x_2\rangle.\mathbf{0}) \mid !\,\nu x.\overline{w}\langle x\rangle.\mathbf{0}
\end{aligned}
$$

Die letzte Ableitung von P_2, also die *Extrusion* der Namen x_1 und x_2, verhindert, dass die beiden Prozessterme Q_1 und Q_2 miteinander kommunizieren können, da ihnen nun kein gemeinsamer Name mehr zur Verfügung steht. Für P_2 hat dies zur Folge, zwar irreduzibel zu sein, jedoch ohne einen offenen INPUT zu besitzen, d.h. er konvergiert nicht.

Kontext C unterscheidet somit die Prozesse P_1 und P_2 hinsichtlich ihrer kontextuellen Äquivalenz und die Ungleichung $\nu x.!\,P \;\not\sim_c\; !\,\nu x.P$ gilt. □

Nachdem die bisherigen kontextuellen Ungleichungen mitunter nicht als „intuitiv“ erkennbar zu bezeichnen waren und dies über etwas komplexere Prozessterme gezeigt werden musste, so verhält es sich mit nachfolgender Ungleichung anders. Dass der ν-Binder kontextuelles Verhalten verändert, konnte bereits zuvor gezeigt werden. Offensichtlicher wird dies, wenn bereits vorgegebene Gültigkeitsbereiche schon zu Anfang eine Kommunikation von Prozessen in bestimmte Richtungen drängen, d.h., eine Extrusion nicht möglich ist. Wie dies formal zu verstehen ist, wird nun gezeigt.[Bou92]

Lemma 3.33
Der ν-Binder ändert das kontextuelle Verhalten von Prozessen, sodass es Prozesse P,Q gibt, die die Ungleichung $\nu x.(P \mid Q) \not\sim_c \nu x.P \mid Q$ erfüllen.

Beweis. Es sei für $P = \overline{v}\langle x\rangle.\mathbf{0} \mid v(y).\overline{y}\langle w\rangle.\mathbf{0}$ und Prozess $Q = x(z).\mathbf{0}$ gewählt, sowie für Kontext $C = [\,]$. Zu zeigen ist nun, dass folgende Ungleichung gilt:

$$C[\nu x.(\overline{v}\langle x\rangle.\mathbf{0} \mid v(y).\overline{y}\langle w\rangle.\mathbf{0} \mid x(z).\mathbf{0})] \not\sim_c C[\nu x.(\overline{v}\langle x\rangle.\mathbf{0} \mid v(y).\overline{y}\langle w\rangle.\mathbf{0}) \mid x(z).\mathbf{0}]$$

Die linke Seite dieser Ungleichung gestattet eine komplette Kommunikation aller Teilprozessterme, die nachfolgend dargestellt sind.

$$\begin{aligned}
&\nu x.(\overline{v}\langle x\rangle.\mathbf{0} \mid v(y).\overline{y}\langle w\rangle.\mathbf{0} \mid x(z).\mathbf{0}) \\
&\rightarrow \nu x.(\mathbf{0} \mid \overline{x}\langle w\rangle.\mathbf{0} \mid x(z).\mathbf{0}) \qquad \text{(REACT)} \\
&\rightarrow \nu x.(\mathbf{0} \mid \mathbf{0} \mid \mathbf{0}) \qquad \text{(REACT)}
\end{aligned}$$

Prozess $\nu x.(\mathbf{0} \mid \mathbf{0} \mid \mathbf{0})$ ist zwar irreduzibel, jedoch ohne offenen INPUT und somit divergent.

Die rechte Seite der oben aufgeführten Ungleichung lässt nur eine beschränkte Kommunikation der einzelnen Teilprozesse zu. Der durch den ν-Binder eingeschränkte Gültigkeitsbereich des Namens x lässt sich nicht auf Prozess Q erweitern, ohne dabei den in Q verwendeten Namen x umzubenennen, sodass eine Kommunikation zwischen Q und P verhindert wird. Dies führt zu folgenden möglichen Reduktionen:

$$\begin{aligned}
&\nu x.(\overline{v}\langle x\rangle.\mathbf{0} \mid v(y).\overline{y}\langle w\rangle.\mathbf{0}) \mid x(z).\mathbf{0} \\
&\rightarrow \nu x.(\mathbf{0} \mid \overline{x}\langle w\rangle.\mathbf{0}) \mid x(z).\mathbf{0} \qquad \text{(REACT)} \\
&\rightarrow \nu x.(\mathbf{0} \mid \overline{x}\langle w\rangle.\mathbf{0}) \mid x'(z).\mathbf{0} \qquad \text{(NEW)}
\end{aligned}$$

Der entstehende Prozess $\nu x.(\mathbf{0} \mid \overline{x}\langle w\rangle.\mathbf{0}) \mid x'(z).\mathbf{0}$ ist erfolgreich, d.h. konvergent, da er sowohl einen offenen INPUT an x hat als auch irreduzibel ist.

Die Gültigkeit obiger Ungleichung ist somit erfolgreich gezeigt, da Kontext C das kontextuelle Verhalten der gewählten Prozesse unterscheidet. □

Bevor weitere Untersuchungen von Pi-Kalkül-Prozessen dem Nachweis der kontextuellen (Un)gleichheit zweier Prozesse dienen, soll nachfolgend ein Hilfsmittel zum Nachweis von korrekten Programmtransformationen vorgestellt werden, die Reduktionsdiagramme.

3.4.4 Reduktionsdiagramme

Im Anschluss an diesen Abschnitt soll für einige *Transformationen* gezeigt werden, dass deren Anwendung zu zwei kontextuell äquivalenten Prozessen führt. Die Nachweise gehorchen hierbei jeweils einem ähnlichen Schema, sodass dies nachfolgend kurz vorgestellt wird.

Wie bereits erwähnt, ist der Nachweis von kontextueller Äquivalenz mitunter sehr schwer. Kontextuell äquivalente Prozessausdrücke dürfen sich hinsichtlich ihrer Terminierung bzw. Konvergenz in keinem Kontext (d.h. in keinem *aller möglichen* Kontexte) unterscheiden lassen. Der entsprechende Nachweis benötigt somit die Betrachtung *unendlich* vieler Kontexte. Um dennoch weiterführende kontextuelle Untersuchungen anstellen zu können, wurde bereits der Begriff der *korrekten* Programmtransformation eingeführt (vgl. Definition 3.28).

Der Nachweis einer korrekten Programmtransformation wird mithilfe von Gabel- und Vertauschungsdiagrammen (sog. Reduktionsdiagramme) geführt. Die diagrammartige Darstellung von Reduktionszusammenhängen und ihrer Mächtigkeit wurde bereits in [vO94, Cos96] und [Bez98] beschrieben. Beispielsweise wurde diese Art der Beweisführung – der Nachweis kontextueller Äquivalenz – bei Untersuchungen im Lambda-Kalkül von [Kut98, Kut00] sowie [Sab08] angewendet.

Nachfolgend werden einige Begriffe, die im Zusammenhang mit der diagrammartigen Darstellungen von Reduktionszusammenhängen stehen, formal definiert. Für ein weitergehendes Interesse sei an dieser Stelle auf die zuvor genannte Literatur verwiesen.[9]

Definition 3.34 (Reduktionsdiagramm)
Ein Reduktionsdiagramm ist ein gerichteter Graph, dessen Knoten Ausdrücke sind und dessen Kanten Reduktionen entsprechen sowie markiert sein können. Es werden zwei Kantenformen unterschieden:

$\longrightarrow$ Ein durchgängiger Pfeil bezeichnet eine gegebene Reduktion.

$\dashrightarrow$ Ein gestrichelter Pfeil bezeichnet eine existenzquantifizierte Reduktion.

Zusätzliche Informationen zur Art der Reduktion können den Kanten angeheftet sein.

Reduktionsdiagramme stellen also Reduktionsfolgen in Kalkülen diagrammartig dar. Hierbei wird zwischen Gabel- und Vertauschungsdiagrammen unterschieden.
Schematisch ist der Unterschied beider Diagramme in Abbildung 3.3 dargestellt. Hier sind die dargestellten Pfeile gemäß Definition 3.34 zu deuten. Mit „ r_*^* “ ist der Platzhalter für zusätzliche Informationen zur Art der Reduktion dargestellt. Die Ecken „ $\circ$ “ der Diagramme entsprechen Platzhaltern für die jeweiligen Prozessausdrücke bzw. ihren Redukten.

9 Hinweis: Je nach untersuchtem Kalkül können sich die Definitionen der Gabel- bzw. Vertauschungsdiagramme unterscheiden.

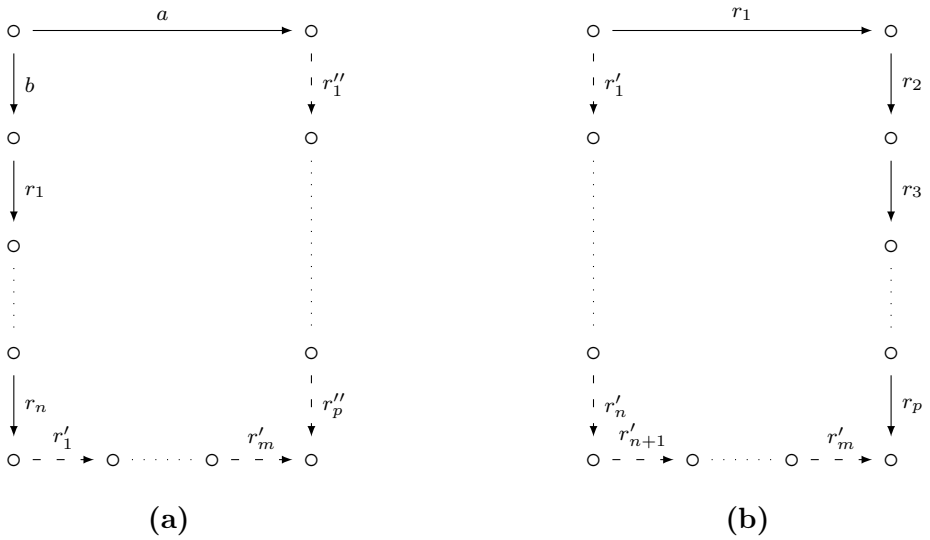


Abbildung 3.3: Allgemeine schematische Darstellung der **(a)** Gabel- und **(b)** Vertauschungsdiagramme

Bei **Gabeldiagrammen** wird von der Existenz einer Reduktion, beispielsweise der Standardreduktion gemäß Definition 3.19, ausgegangen und versucht, eine mögliche Reduktionsfolge in Überlappung mit der bereits existenzquantifizierten Regel zu bringen, d.h. die grafisch entstandene „Gabel“, bestehend aus zwei gegebenen Reduktionen (hier a und b, sowie r_1,...,r_n), über die gestrichelten, oder gepunkteten Pfeile zu schließen.

Vertauschungsdiagramme basieren auf einer anderen Annahme. Hier ist „irgendeine“ Transformation mit anschließender Standardreduktion Ausgangspunkt des zu schließenden Diagramms. Die ermittelte Reduktionsfolge zum Schluss eines Reduktionsdiagramms stellt sodann eine Meta-Reduktionsregel für eine zu untersuchende Reduktionsfolge dar.

In späteren Beweisen werden zumeist nicht, wie in Abbildung 3.34 dargestellt, solch „große“ Diagramme Verwendung finden, oft genügen zum Nachweis von korrekten Programmtransformationen kleinere Diagramme, wie in Abbildung 3.4 dargestellt. Hierbei weisen die *Mengenklammern* auf die eventuelle Existenz einer Vielzahl von Reduktionsdiagrammen hin.

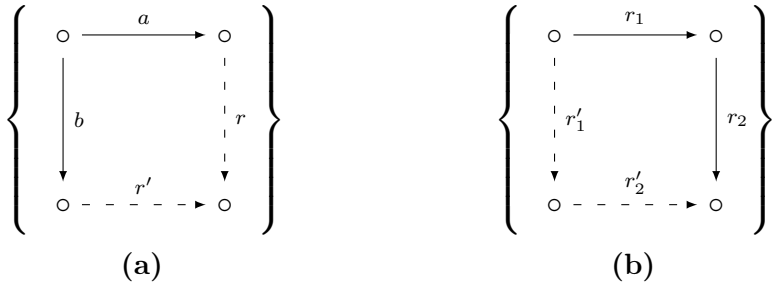


Abbildung 3.4: Eine Menge von **(a)** Gabel- und **(b)** Vertauschungsdiagrammen

Anmerkung: Anders als bei der Widerlegung der kontextuellen Äquivalenz zweier Prozesse, bei denen ein Gegenbeispiel genügt, um dies zu beweisen, ist es bei Beweisen kontextuell sich gleich verhaltender Prozesse nötig, diese über alle möglichen Kontexte zu führen. Hieraus ergibt sich, dass bei der Beweisführung mit Reduktionsdiagrammen nicht nur ein Reduktionsdiagramm einer Reduktionsfolge zu berücksichtigen ist, sondern eine Vielzahl, eine komplette Menge *(engl.: conplete set)* aller möglichen Reduktionen.

Eine spezielle Notation zur kompakteren Darstellung von Gabel- und Vertauschungsdiagrammen ist nachfolgend beschrieben. Hierbei wird ebenfalls „ ∘ „ als Platzhalter für Prozessausdrücke verwendet.

Definition 3.35 (Gabeldiagramm)
Ein Gabeldiagramm G ist eine Meta-Regel der Gestalt:

$$\xleftarrow{r_n} \circ \cdots \circ \xleftarrow{r_1} \circ \xleftarrow{b} \circ \xrightarrow{a} \rightsquigarrow \xrightarrow{r'_1} \circ \cdots \circ \xrightarrow{r'_m} \circ \xleftarrow{r''_p} \circ \cdots \circ \xleftarrow{r''_m}$$

$$\text{mit } m, n, p \geq 0$$

Der entsprechende Graph ist in Abbildung 3.3a dargestellt.

Definition 3.36 (Vertauschungsdiagramm)
Ein Vertauschungsdiagramm G ist eine Meta-Regel der Gestalt:

$$r_1 \circ \cdots \circ r_p \rightsquigarrow r'_1 \circ \cdots \circ r'_p \qquad \text{mit } m \geq 0,\ p \geq 2$$

Der entsprechende Graph ist in Abbildung 3.3b dargestellt.

Mithilfe dieser Reduktionsdiagramme wird später versucht kontextuelle Gleichheiten von Prozessausdrücken, d.h. die Korrektheit von Programmtransformationen, zu beweisen. Zumeist werden die Beweise über Gabeldiagramme geführt.
Die existenzquantifizierten Regeln sind hierbei stets angegeben. Zumeist handelt es sich hierbei um die in diesem Zusammenhang auf Korrektheit zu prüfende Transformation selbst.
Beweise der kontextueller Äquivalenz von zwei Prozessen folgen zumeist nachfolgendem Schema:

1. Die Enwicklung eines *vollständigen Satzes* von Gabel- bzw. Vertauschungsdiagrammen für die untersuchte Reduktion r.

2. Die wechselseitige Erhaltung der Prozesskonvergenz wird gezeigt, d.h. $P \xrightarrow{r} Q$ ist korrekt, g.d.w. P ist konvergent (geschr. $P\downarrow$) g.d.w. Q konvergent ist (geschr. $Q\downarrow$).

3. Unter Zuhilfenahme der zuvor gezeigten Gabel- und Vertauschungsdiagramme sowie der Erhaltung der wechselseitigen Konvergenzeigenschaft der Prozesse, wird induktiv über die wechselseitige Übertragbarkeit einer Standardreduktion zu einem konvergenten Prozess argumentiert.
   Formal muss für zwei kontextuell äquivalente Prozesse Folgendes gezeigt werden: Gegeben sind zwei Prozesse P, Q mit $P \xrightarrow{r} Q$. So ist zu zeigen:

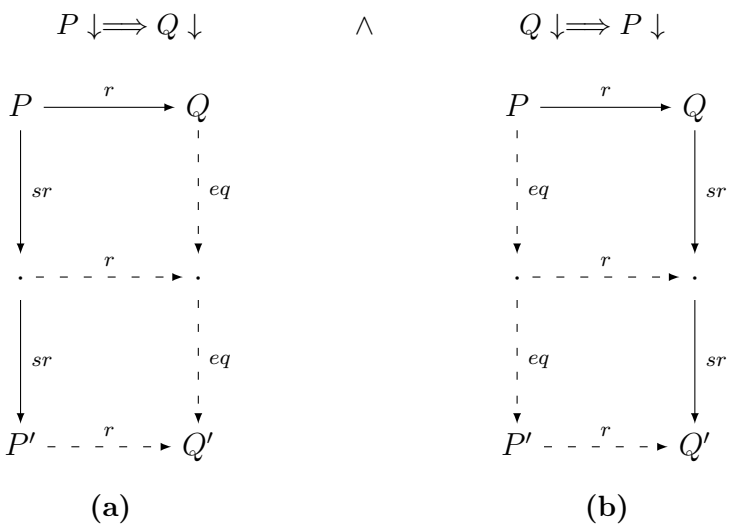


Abbildung 3.5: (a) zeigt den Schluss eines Gabeldiagramms der Transformation $P \xrightarrow{r} Q$ über die existent quantifizierte Regel *(eq)* vom *erfolgreichen* Prozess P', über die als korrekt zu beweisende Regel (r) und unter Verwendung der bereits als korrekt gezeigten Transformation (sr), hin zu dem *erfolgreichen* Prozess Q'. In **(b)** wird die „Rückrichtung" des Schlusses von **(a)** unter Voraussetzung des *erfolgreichen* Prozesses Q' hin zu P' schematisch dargestellt.

4. Aus den unter 3. bewiesenen Aussagen folgt gemäß Korollar 3.26 die zu zeigende Erhaltung der kontextuellen Äquivalenz der Reduktion r.

Anmerkung: Übertragen auf eine grafische Sicht, basieren diese Korrektheitsbeweise auf der Idee von sich *überlappenden* Diagrammen. Eine solche Überlappung von Diagrammen existiert genau dann, wenn die gegebene und die existenzquantifizierte Reduktion ihren Ursprung in demselben Prozessterm haben und somit zusammengefasst werden können.

Die induktive Argumentation unter Punkt 3 kann bei manchen Reduktionen signifikant vereinfacht werden, indem beispielsweise das im nachfolgenden Abschnitt vorgestellte Kontextlemma bei der Beweisführung Verwendung findet, sodass die meisten hier geführten Beweise von obigem Schema leicht abweichen werden.

3.4.5 Das Kontextlemma

Das so genannte Kontextlemma aus [Sab10b] dient der Reduzierung der zu betrachtenden Kontexte, wenn es um den Nachweis von kontextuell äquivalentem Programm- bzw. Prozessverhalten geht.
Es besagt, dass es genügt, Reduktionskontexte zu betrachten, um eine $\leq_c$-Beziehung zwischen zwei Prozessen nachzuweisen.

Lemma 3.37 (Kontextlemma)
Seien P und Q Prozesse. Wenn für alle Reduktionskontexte D und alle Umbenennungen von Namen σ gilt, dass $D[\sigma(P)]\downarrow \Longrightarrow D[\sigma(Q)]\downarrow$, dann gilt $P \leq_c Q$.

Beweis. Siehe [Sab10b, Definition 5.3 und Lemma 5.6]. □

Als weiteres Kriterium zum Nachweis kontextueller Äquivalenz sei nachfolgendes Lemma genannt.

Lemma 3.38
P und Q seien Prozesse und D-Reduktionskontexte. Eine hinreichende Bedingung für $P \sim_c Q$ ist:

$$\forall D : D[\sigma(P)]\downarrow \Longleftrightarrow D[\sigma(Q)]\downarrow$$

Der Beweis von kontextuell äquivalenten Prozessen lässt sich somit zumeist über eine geringere Zahl an zu betrachtenden Kontexten unter Verwendung der Reduktionsdiagramme führen.

In Zusammenhang mit dem Beweis des Kontextlemmas aus [Sab10b] wurden einige Relationen eingeführt, die auch hier zu einem späteren Zeitpunkt Verwendung finden, sodass die wichtigsten nachfolgend definiert werden.

Definition 3.39 (Strukturelle Reduktion)
Die Reduktion $\overset{sc}{\to}$ ist definiert durch:[1]0

$$\begin{aligned}
P &\overset{sc}{\to} Q, \text{ falls } P =_\alpha Q \\
P \mid \mathbf{0} &\overset{sc}{\to} P \\
P &\overset{sc}{\to} P \mid \mathbf{0} \\
P_1 \mid (P_2 \mid P_3) &\overset{sc}{\to} (P_1 \mid P_2) \mid P_3 \\
(P_1 \mid P_2) \mid P_3 &\overset{sc}{\to} P_1 \mid (P_2 \mid P_3) \\
P_1 \mid P_2 &\overset{sc}{\to} P_2 \mid P_1
\end{aligned}$$

10 Analog zu [Sab10b, Definition 4.1].

$$\begin{aligned}
\nu u.\mathbf{0} &\xrightarrow{sc} \mathbf{0} \\
\mathbf{0} &\xrightarrow{sc} \nu u.\mathbf{0} \\
\nu u.\nu v.P &\xrightarrow{sc} \nu v.\nu u.P \\
\nu u.(P_1 \mid P_2) &\xrightarrow{sc} P_1 \mid \nu u.P_2, \text{ falls } u \notin fn(P_1) \\
P_1 \mid \nu u.P_2 &\xrightarrow{sc} \nu u.(P_1 \mid P_2), \text{ falls } u \notin fn(P_1) \\
!P &\xrightarrow{sc} P \mid !P
\end{aligned}$$

Die Relation $\xrightarrow{D,sc}$ ist definiert als:

$P \xrightarrow{D,sc} Q$ genau dann, wenn ein Reduktionskontext D und Prozess P', Q' existieren mit $P' \xrightarrow{sc} Q'$ und $D[P'] = P$ sowie $D[Q'] = Q$.

Mit Einführung der strukturellen Reduktion sowie des D-Reduktionskontexts lässt sich eine weitere Reduktion – ähnlich der Standardreduktion – sr definieren.

Satz 3.40. *Findet ein Reduktionsschritt und Verwendung der Regel* (INTERACT) *statt, so ist diese eine D-Standardreduktion (geschr.* $\xrightarrow{dsr}$*) g.d.w. sie in einem Reduktionskonext (modulo* $\xrightarrow{D,sc,*}$*) stattfindet, d.h.:*

$$\frac{P \xrightarrow{D,sc,*} D[P'], P' \xrightarrow{int.} Q', D[Q'] \xrightarrow{D,sc,*} Q}{P \xrightarrow{dsr} Q}$$

In [Sab10b, Theorem 4.14] konnte gezeigt werden, dass sich die Reduktionen (dsr) und (sr) hinsichtlich der Konvergenz nicht unterscheiden. In den nachfolgenden Betrachtungen des Konvergenzverhaltens von Reduktionen, kann somit auch immer die Reduktion (sr) gedanklich gegen die Reduktion (dsr) ausgetauscht werden, ohne dass dies Auswirkungen auf die getroffenen Schlüsse hat. Dies ist insoweit sinnvoll, als dass unter Verwendung der (dsr) Reduktion eine i. Allg. geringe Zahl an Kontexten zu betrachten ist.

Satz 3.41. *Es gilt:* $P\downarrow \iff P \xrightarrow{dsr,*} P'$ *und* P' *ist* erfolgreich.

Anmerkung: Im Folgenden wird eine $\xrightarrow{int.}$-Reduktion, die in einem D-Kontext stattfindet, mit $\xrightarrow{D,int.}$ bezeichnet. Demzufolge entspricht eine $\xrightarrow{dsr}$-Reduktion einer Folge von Transformationen: $\xrightarrow{D,sc,*} \cdot \xrightarrow{D,int.} \cdot \xrightarrow{D,sc,*}$.

3.4.6 Nachweis kontextueller Gleichheiten

Bis zu diesem Zeitpunkt konnte bereits eine kontextuelle Gleichheit im Pi-Kalkül nachgewiesen werden, denn wie in Satz 3.27 mit anschließendem Beweis gezeigt wurde,

ist die strukturelle Kongruenz mit der kontextuellen Äquivalenz verträglich, d.h., die strukturelle Kongruenz erhält die kontextuelle Gleichheit.
Im weiteren Verlauf werden nun weitere kontextuelle Gleichheiten gezeigt. Nicht immer ist es hierbei nötig, auf Reduktionsdiagramme zurückzugreifen. Ein Beispiel hierfür ist nachfolgend gegeben.

Lemma 3.42
Es gilt: $\nu x.P \sim_c P$ *falls* $x \notin fn(P)$.

Beweis. Beim Nachweis dieser Äquivalenz kann über die strukturelle Kongruenz argumentiert werden, da für diese bereits gezeigt wurde, dass sie die kontextuelle Gleichheit erhält, obgleich obiger Fall nicht direkt aus ihr abzulesen ist.

Es gilt:

$$\nu x.P \equiv \nu x.(P \mid \mathbf{0}) \equiv P \mid \nu x.\mathbf{0} \equiv P \mid \mathbf{0} \equiv P \quad \text{falls } x \notin fn(P)$$

Und nach Satz 3.27 ist: $\nu x.P \sim_c P$ falls $x \notin fn(P)$. □

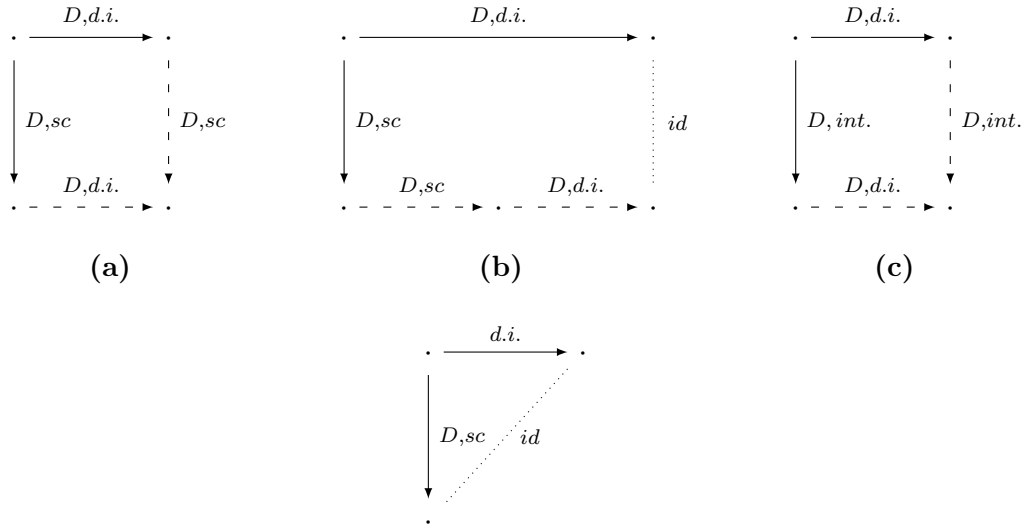
Eine weitere Transformation, für die angenommen wird, dass es sich um eine *korrekte* Programmtransformation handelt, ist die Reduktion *deterministic interact* aus Definition 3.43. Hierbei ist die Restriktion der Kommunikation zweier Prozesse auf einen Kanal und die damit einhergehende „Abschottung“ gegenüber weiterer Kommunikation über diesen Kanal, Grundlage der Vermutung, dass es sich um eine korrekte Programmtransformation handelt.

Definition 3.43 (Deterministic Interact)
Die *deterministic interact*-Reduktion (geschr. $\xrightarrow{d.i.}$) ist wie folgt definiert:

$$\nu x(x(y).P \mid \overline{x}\langle z\rangle.Q) \longrightarrow \nu x(P\,[^{z}/_{y}] \mid Q)$$

Bevor im Anschluss die Korrektheit der $\xrightarrow{d.i.}$ - Reduktion gezeigt und hierbei auf Reduktionsdiagramme zurückgegriffen wird, ist es nötig, den Zusammenhang zwischen der zu überprüfenden $\xrightarrow{D,d.i.}$ - Reduktion (d.h. $\xrightarrow{d.i}$ angewendet in einem Reduktionskontext D) und den Transformationen $\xrightarrow{D,sc}$ und $\xrightarrow{D,int.}$ zu klären. Dies ist erforderlich, da mit diesen Transformationen die Reduktion $\xrightarrow{d.i.}$ überlappt wird und so auf korrekte Programmtransformationen zurückzuführen ist. Solch eine Überlappung wiederum ist machbar, da nach Satz 3.40 gerade Transformation $\xrightarrow{D,sc}$ und $\xrightarrow{D,int.}$ die möglichen Transformationen von $\xrightarrow{dsr}$ darstellen.

Lemma 3.44
Die Reduktion deterministic interact *verfügt über den folgenden vollständigen Satz an Gabeldiagrammen. In ihnen bezeichnen die durchgängigen Pfeile die gegebene Transformation, die gestrichelten Pfeile die existenzquantifizierten Reduktionen und* id := identische *Prozesse.*



Nachfolgend gilt es, die Korrektheit der obigen Reduktionsdiagramme zu beweisen.

Beweis.

- Reduktionsdiagramm **(a)** beschreibt all jene Prozessterme, in denen die Reduktion $\xrightarrow{D,d.i.}$ unabhängig von strukturellen Veränderungen des Prozessterms durchgeführt werden kann, da sich diese Veränderungen *nicht* auf den Redex beziehen. D.h., die Redexe der $\xrightarrow{D,sc}$-Reduktion und der $\xrightarrow{D,d.i.}$-Transformation überlappen einander nicht, sodass die Gabel in diesem Fall immer durch entsprechende Transformationen geschlossen werden kann.

- Reduktionsdiagramm **(b)** berücksichtigt den Fall, dass eine strukturelle Reduktion $\xrightarrow{D,sc}$ den Redex der $\xrightarrow{D,d.i.}$-Transformation verletzt. Dies kann beispielsweise durch folgende Veränderungen geschehen:
  - Vertauschung von Prozessen.
  - Hinzunahme von inaktiven Prozessen.
  - Verschiebung des ν-Binders.

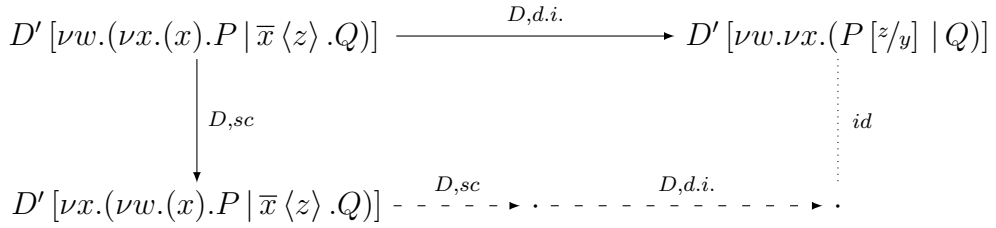
  Somit ist zunächst – nach Anwendung der $\xrightarrow{D,sc}$-Reduktion – die „passende“ $\xrightarrow{D,d.i.}$-Transformation nicht möglich. Erst nachdem die $\xrightarrow{D,sc}$-Reduktion rückgängig

gemacht wurde, d.h. nach einer weiteren $\xrightarrow{D,sc}$-Reduktion, kann im Anschluss die $\xrightarrow{D,d.i.}$-Transformation durchgeführt werden.

Hierbei ist zu erwähnen, dass die $\xrightarrow{D,sc}$-Reduktion zur Entfaltung einer Replikation (d.h. die Anwendung der Regl $!P \to P \,|\, !P$) zwar nicht rückgängig gemacht werden kann, jedoch in diesem Fall auch nicht auftritt: Eine solche Reduktion kann nicht mit dem $\xrightarrow{D,d.i.}$-Redex überlappen, da dieser aufgrund des Reduktionskontextes D nicht unter einer Replikation liegen kann.

Als Beispiel wird das Verschieben eines ν-Binders betrachtet.

Gegeben sei Prozess $R := \nu x.(x(y).P \,|\, \overline{x}\,\langle z\rangle\,.Q)$ und Reduktionskontext D mit $D := D'\,[\nu w.[\,]]$ und D,sc vertauscht νw mit νx, d.h:



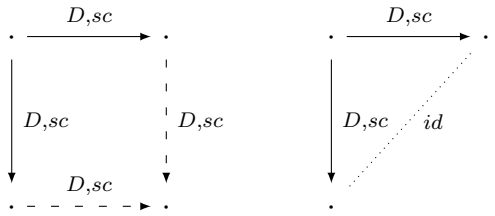
Damit die Reduktion $\xrightarrow{D,d.i.}$ möglich ist, muss zunächst die "Vertauschung"von $\xrightarrow{D,d.i.}$ rückgängig gemacht werden. Hieran schließt sich die Reduktion $\xrightarrow{D,d.i.}$, hiernach ist der entstandene Prozessterm identisch mit der direkten Ausführung von $\xrightarrow{D,d.i.}$ (gekennzeichnet mit *id*).

- Per Definition der Reduktion $\xrightarrow{D,d.i.}$ tritt eine Überlappung von Redexen, der $\xrightarrow{D,d.i.}$ sowie der normalen (INTERACT)-Reduktion nicht auf, außer bei identischen Redexen (Diagramm (d)). Somit ist ausgeschlossen, dass irgendeine *äußere* Reduktion im Gültigkeitsbereich des Redexes von $\xrightarrow{D,d.i.}$ reduziert. Dies führt zu Reduktionsdiagramm **(c)**, wonach die Anwendung der herkömmlichen Reduktionsregel-(INTERACT) (gekennzeichnet mit $\xrightarrow{D,int.}$) vor oder nach $\xrightarrow{D,d.i.}$ durchgeführt werden kann. □

Wie mit Reduktionsdiagramm **(c)** aus dem vorherigem Lemma zu erkennen ist, tauchen beim Schluss des Diagramms, über die untere Kante, zwei aufeinander folgende Reduktionen auf. Es ist also nötig, dass zuerst eine D,sc - Reduktion durchgeführt wird, bevor *d.i.* das Gabeldiagramm schließen kann.
Da ein direkter Schluss – also lediglich über die Reduktion *d.i.* – nicht möglich ist, erfolgt eine Überlappung des Reduktionsdiagramms von $\cdots \xleftarrow{D,sc} \circ \xrightarrow{d.i.} \rightsquigarrow \xrightarrow{D,sc} \circ \cdots$ mit dem Satz an Reduktionsdiagrammen von $\cdots \xleftarrow{D,sc} \circ \xrightarrow{D,sc} \cdots$.
Damit auch die Korrektheit der *deterministischen Reduktion* nachgewiesen werden kann, müssen zunächst die vollständigen Sätze an Reduktionsdiagrammen der verwendeten

Transformationen ermittelt werden, sodass eine vollständige Überlappung mit korrekten Programmtransformationen für den späteren Induktionsbeweis der *deterministischen*-Reduktion gegeben ist.

Lemma 3.45
Der vollständige Satz an Gabeldiagrammen für die Reduktion D,sc, gemäß Definition 3.39, ist:



(a) **(b)**

Es genügen zwei Reduktionsdiagramme, um alle möglichen sich ergebenden Prozessterme zu überlappen.

Beweis.

- Bis auf eine Reduktion ist jede *D,sc*-Reduktion umkehrbar, d.h., für eine Folge von *D,sc*-Reduktionen, die lediglich strukturelle Veränderungen darstellen, findet sich immer eine passende *andere* Folge, sodass *identische* Prozesse entstehen und Diagramm **(a)** gilt.

  Als Beispiel wird das Hinzufügen des „leeren“ Prozesses (0) betrachtet.

  Gegeben sei Prozess $R := \nu w.(\nu x.P)$ und Reduktionskontext D mit $D := D'[\,]$.

$$\begin{array}{ccc} D'[\nu w.(\nu x.P)] & \xrightarrow{D,sc} & D'[\nu x.(\nu w.P)] \\ \Big\downarrow{\scriptstyle D,sc} & & \Big\downarrow{\scriptstyle D,sc} \\ D'[\nu w.(\nu x.P \mid \mathbf{0})] & \overset{D,sc}{\dashrightarrow} & D'[\nu w.(\nu x.P \mid \mathbf{0})] \end{array}$$

- Diagramm **(b)** stellt hierbei lediglich den „trivialen“ Fall der Anwendung derselben *D,sc*-Reduktion auf den *gleichen* Prozessterm dar.

- Die einzig nicht umkehrbare Transformation ist $!P \longrightarrow P \mid !P$. Per Definition des Reduktionskontextes D ist jedoch *keine* Reduktion unterhalb einer Replikation erlaubt, sodass dieser Fall hier nicht berücksichtigt werden muss. □

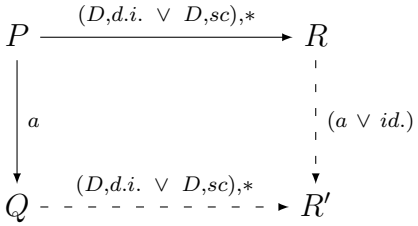
Nachfolgend gilt es nachzuweisen, dass die Reduktion aus Lemma 3.44 auch tatsächlich eine *korrekte* Programmtransformation darstellt.

Zur Erinnerung: Eine Programmtransformation T ist genau dann *korrekt*, wenn für alle Prozesse $(P, Q) \in T$ gilt: $P \sim_c Q$.

Somit bleibt zu zeigen, dass sich das Terminierungsverhalten hinsichtlich der Transformation $d.i$ nicht ändert. Dies wird mithilfe von Lemma 3.46, 3.47, 3.49 gezeigt. Hierbei bedeutet $\xrightarrow{a \vee b}$ für Transformationen $\xrightarrow{a}$, $\xrightarrow{b}$, dass eine Transformation $\xrightarrow{a}$ oder eine Transformation $\xrightarrow{b}$ Verwendung findet, d.h. $\xrightarrow{a \vee b} = \xrightarrow{a} \cup \xrightarrow{b}$. Wobei „oder“ hier nicht ausschließend zu verstehen ist.

Lemma 3.46
Gegeben seien Prozesse P, Q, R, mit $P \xrightarrow{a} Q$, $a \in \{D, int., D,sc\}$ und $P \xrightarrow{(D,d.i. \vee D,sc),*} R$. So existiert ein Prozess R' mit: $Q \xrightarrow{(D,d.i. \vee D,sc),*} R'$ und $R \xrightarrow{a} R'$, oder $R = R'$.

In grafischer Repräsentation:

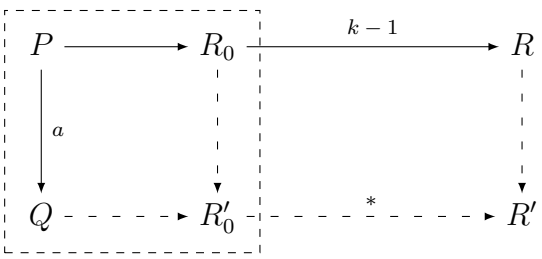


Beweis. Zu zeigen ist: Es ex. eine Reduktionsfolge der Länge k mit k:= fest, aber beliebig, mit $P \xrightarrow{(D,d.i. \vee D,sc),k} R$.

Induktion über k:

$k = 0$:
Dies ist genau dann der Fall, wenn Prozess P und R bereits identisch sind. Wähle $Q = R'$ und die Behauptung gilt.

$k - 1 \longrightarrow k$:
Dies entspricht in grafischer Repräsentation:



Der linke (umrahmte) Teil des obigen Diagramms beschreibt gerade vorherige Sätze an Reduktionsdiagrammen, da hier nur *ein* Reduktionsschritt durchgeführt wird, der genau durch vorherige Diagramme abgedeckt wird.
Der rechter Teil gilt nach Induktionsvoraussetzung. An dieser Stelle ist es wichtig zu erwähnen, dass es bei der Überlappung mit vorherigen Reduktionsdiagrammen, im Speziellen mit Diagramm **(c)** aus Lemma 3.44, zu einer um *eins* längeren Reduktionsfolge – je nach Häufigkeit – kommen kann. Dies stellt jedoch kein Problem dar, da lediglich *eine* Reduktionsfolge beliebiger Länge existieren muss. □

Anders aufgefasst: Die Behauptung aus Lemma 3.46 lässt sich auch als „großes“ Gabeldiagramm für die Transformation $\xrightarrow{(D,d.i. \vee D,sc),*}$ sehen.

Nachdem in Lemma 3.46 gezeigt wurde, dass eine $d.i.$-Reduktion möglich ist, d.h. korrekte Programmtransformationen erlauben den Schluss des Gabeldiagramms der $d.i.$-Reduktion, wird in Lemma 3.47 und Lemma 3.49 gezeigt, dass sich das Terminierungsverhalten eines Prozesses nach einer $d.i.$-Reduktion nicht ändert.

Lemma 3.47
Sei Prozess P mit $P \xrightarrow{(D,d.i. \vee D,sc),*} R$ gegeben und P konvergiert (d.h. $P \downarrow$), dann konvergiert auch R.

Behauptung: Es ex. ein Prozess P mit: $P \xrightarrow{(D,sc,\, D,int),k} P'$ und P' ist erfolgreich.

Beweis. Analog zu Lemma 3.46

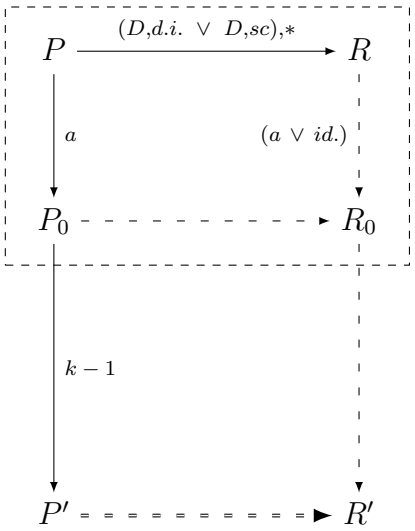
Induktion über k:

$k = 0$:
Fallunterscheidung für: Prozess P ist *erfolgreich*, dann *konvergiert* Prozess R.

1. Fall Prozess P ermöglicht eine weitere $\xrightarrow{D,d.i.}$-Reduktion: Dies ist nicht möglich, da Prozess P bereits *erfolgreich* ist. Jede $\xrightarrow{D,d.i}$-Reduktion stellt per Definition auch eine *Standardreduktion* dar und erfolgreiche Prozesse sind definitionsgemäß bereits irreduzibel.

2. Fall Auf Prozess P wird eine $\xrightarrow{D,sc.}$-Reduktion angewendet: Da die $\xrightarrow{D,sc.}$-Reduktion in der *strukturellen Kongruenz* enthalten ist und diese die kontextuelle Gleichheit erhält, folgt aus „P ist erfolgreich“ sofort „R konvergiert“.

$k - 1 \longrightarrow k$:
Dies entspricht in grafischer Repräsentation mit $a \in \{D, int., D,sc\}$:



Der obere (umrahmte) Teil des Reduktionsdiagramms ergibt sich aus Lemma 3.46, sodass eine Prozess P_0 mit $P_0 \longrightarrow R_0$ existiert. Unter Verwendung der Induktionsbehauptung (P ist *erfolgreich*) folgt somit, $P' \downarrow \longrightarrow R' \downarrow$. □

Als kleiner Zwischenschritt lässt sich jetzt bereits nachfolgendes Lemma beweisen.

Lemma 3.48
Seien P und Q Prozesse, so gilt:

$$P \xrightarrow{d.i.} Q \Longrightarrow P \leq_c Q$$

Beweis. Folgt aus Kontextlemma 3.37 + Lemma 3.47. □

Somit bleibt zum Nachweis der Korrektheit der Reduktion *d.i.* zu zeigen, dass, wenn $P \xrightarrow{d.i.} Q$ gilt und $Q \downarrow$ (erfolgreich) ist, ebenfalls P erfolgreich sein muss, d.h. $P \downarrow$ folgt.

Lemma 3.49
Sei $P \xrightarrow{d.i.} Q$ und $Q \downarrow$ (erfolgreich), so folgt sofort $P \downarrow$, da gilt:

$$\xrightarrow{d.i.} \subseteq \xrightarrow{D,int.} \text{ und } \xrightarrow{D,int.} \subseteq \sim_c$$

Beweis. Folgt aus Lemma 3.47, Lemma 3.49 sowie dem Kontextlemma 3.37. □

Zusammengefasst konnte somit die Gültigkeit des nachfolgenden Satzes gezeigt werden.

Satz 3.50. *Die Reduktion* deterministic interact *ist eine korrekte Programmtransformation.*

Beweis. Folgt direkt aus Lemma 3.47 und 3.49. □

Nachdem nun für einige Transformationen gezeigt werden konnte, dass diese die kontextuelle Gleichheit verletzen wie auch für einige Reduktionen deren Korrektheit bewiesen werden konnte, muss an dieser Stelle auf weitere Untersuchungen von Gleichheiten sowie Ungleichheiten verzichtet werden, da das den Rahmen der Diplomarbeit sprengen würden.
Einige Ideen für mögliche weitere interessante Gleich- bzw. Ungleichheiten werden im Zusammenhang der abschließender Betrachtung der Arbeit in Kapitel 4.2.1 gegeben.

Kapitel 4

4 Zusammenfassung und Ausblick

4.1 Zusammenfassung

Ziel dieser Arbeit war es, einen neuen Ansatz für eine kontextuelle Semantik im Pi-Kalkül zu untersuchen, analysieren und mit bereits bekannten Gleichheitsbegriffen im Pi-Kalkül zu vergleichen. Hierfür wurde eine eingeschränktere Syntax des Pi-Kalküls gewählt, um in einem ersten Schritt kontextuelle Untersuchungen von Prozessen leichter zugänglich zu machen. Hierbei musste leider festgestellt werden, dass ein Vergleich mit bereits bekannten Gleichheitsbegriffen im Pi-Kalkül sehr aufwändig ist, den Rahmen diese Diplomarbeit überschritten hätte und somit diese Analyse unmöglich gemacht hätte.

In *Kapitel 2* wurden zunächst die allgemeinen Grundlagen von Prozesskalkülen erörtert, um eine Übersicht über die Unterschiede sowie über die verschiedenen hiermit im Zusammenhang stehenden Begriffe zu schaffen. Anschließend konnten anhand des Vorgängers des Pi-Kalküls, dem *Calculus of Communicating Systems*, ein Einblick in die Verwendung und den Einsatzzweck von Prozesskalkülen gegeben werden. An dieser Stelle wird auch die Notwendigkeit für eine Erweiterung des CCS zum ersten Mal ersichtlich, da keine Darstellung von mobilen Prozessen im CCS möglich ist. In diesem Zusammenhang ist mithilfe einiger Beispiele die operationale Semantik von CCS verdeutlicht worden.
Was und vor allem wie Prozessgleichheit definiert werden kann, wurde anhand der Bisimulation mit Beispielen, wie auch am Konzept der kontextuellen Äquivalenz zunächst einführend dargestellt. Das Kapitel endet mit der allgemeinen Definition der kontextuellen Gleichheit, sowie den dafür notwendigen weiteren Definitionen.

Da der Schwerpunkt dieser Arbeit der Pi-Kalkül darstellt, wurde in *Kapitel 3* zunächst die Notwendigkeit einer Erweiterung des CCS anhand eines Beispiels näher erläutert. Hierbei wurde auf die ungenügende Möglichkeit der Modellierung von mobilen Prozessen im CCS hingewiesen, sodass eine Erweiterung des CCS nötig war, um die sich immer mehr verändernden Strukturen der realen Welt weiterhin formal abbilden zu können. Hierzu wurden der entsprechende Syntax sowie in diesem Zusammenhang stehende

weitere Begriffe definiert wie auch beispielhaft dargestellt.
Die strukturelle Kongruenz des Pi-Kalküls ist ebenso definiert worden wie die operationale Semantik. Beispiele dienten auch hier dem Aufzeigen einiger wichtiger prozessalgebraischer Eigenschaften des Pi-Kalküls. Im Anschluss daran wurde zum einen auf die Unterschiede des *synchronen* wie auch *asynchronen* Pi-Kalkül hingewiesen, zum anderen die notwendigen neuen Transformationen im Zusammenhang mit der Prozess-Gleichheit im synchronen Pi-Kalkül eingeführt.
Die kontextuelle Äquivalenz wurde definiert und bewiesen, dass die strukturelle Kongruenz diese erhält. Wie korrekte Programmtransformationen sich mithilfe von Reduktionsdiagrammen nachweisen lassen, wurde genauso geklärt wie für einige Programmtransformationen gezeigt werden konnte, dass diese die kontextuelle Gleichheit verletzen. Unter Verwendung des Kontextlemmas aus [Sab10b] konnten die zu betrachtenden Kontexte auf Reduktionskontexte beschränkt werden, was einer Reduzierung der zu betrachtenden Fälle gleich kommt. Ob die Reduktion *deterministik interact* eine korrekte Programmtransformation darstellt und somit die kontextuelle Äquivalenz erhält, konnte zum Ende des Kapitels unter Verwendung des Kontextlemmas wie auch der Reduktionsdiagramme nachgewiesen werden.

Es konnte somit gezeigt werden, dass die kontextuelle Äquivalenz sehr wohl in der Lage ist, neben der Bisimulation, nützliche und überraschende Aussagen über Prozessausdrücke zu treffen. Der Nachweis von kontextueller Gleichheit ist unter Zuhilfenahme des Kontextlemmas etwas weniger komplex, wenn auch noch immer sehr schwierig bzw. aufwändig. Die Einschränkung der zu betrachtenden Kontexte auf Reduktionskontexte kann in einigen Fällen zwar eine Verringerung der zu betrachtenden Fälle darstellen, i. Allg. bleiben jedoch weiterhin eine Vielzahl von ihnen zu betrachten. Leichter bzw. schneller ist hingegen der Nachweis von sich kontextuell unterscheidenden Prozessen. Es genügt hierbei lediglich, einen Kontext zu finden, in dem sie sich unterscheiden.
Inwieweit sich die kontextuelle Äquivalenz bei Veränderung bzw. Erweiterung der Syntax verhält, bleibt im Einzelfall zu prüfen, sodass allgemeine Aussagen wie im Lambda-Kalkül schwer zu treffen sind.

4.2 Ausblick

Abschließend seien einige weiter Fragen bzgl. kontextuellen Verhaltes im Pi-Kalkül gestellt, die im Hinblick auf zukünftige Forschungen von Interesse sein könnten, beantwortet zu werden.

4.2.1 Weitere Gleichheiten

Neben den in dieser Arbeit bereits nachgewiesenen Gleichheiten und Ungleichheiten, mögen weiterführende kontextuelle Untersuchungen von Prozessen im Hinblick auf spezielle Transformationen von weiterem Interesse sein.

Es gibt neben den hier gezeigten Reduktionen noch viele weitere, die sich nicht aus der strukturellen Kongruenz des Pi-Kalküls ableiten lassen, sich jedoch, bezogen auf die kontextuelle Äquivalenz, als evtl. gleich herausstellen. Als ein Beispiel sei hier nachfolgende Transformation genannt:

$$x\,(y)\,.\nu z.P \quad \overset{?}{\sim_c} \quad \nu z'.x\,(y)\,.P\,[z'/z]$$

Wobei der ν-Binder über ein INPUT-Präfix "geschoben"wird. Analog dazu die Beantwortung der Frage, ob auch

$$\overline{x}\,\langle y\rangle\,.\nu z.P \quad \overset{?}{\sim_c} \quad \nu z'.\overline{x}\,\langle y\rangle\,.P\,[z'/z]$$

ist.

4.2.2 Erweiterung der Syntax

Würde eine Erweiterung der Syntax des Pi-Kalküls die in dieser Arbeit getroffenen Aussagen ebenfalls erfüllen? Oder reichen bereits kleine Veränderungen aus, um die bereits als korrekt gezeigten Programmtransformationen wieder verwerfen zu müssen? Da es im Pi-Kalkül möglich ist, die Syntax beliebig zu erweitern oder einzuschränken, muss somit für jede Erweiterung die Gültigkeit bereits gemachter Feststellungen erneut nachgeprüft und bewiesen werden.
Als eine mögliche Erweiterung sei die Hinzunahme der Möglichkeit zur Rekursion genannt. Hierbei bietet sich nicht nur via Replikation eine mehrfache Ausführung von Prozessen an, sondern Prozesse können sich wie allgemein bei der Rekursion möglich, selbst mit veränderter Eingabe „aufrufen“.

4.2.3 Übertragbarkeit von Ergebnissen

Von Interesse mag die Klärung der Frage sein, inwieweit sich Aussagen über bereits als bisimular gezeigte Prozesse auch auf kontextuell Äquivalente übertragen lassen. Bisher wurden beide Konzepte, sowohl die Bisiumaltion als auch die kontextuelle Äquivalenz, eher nebeneinander verwendet, ohne dabei die Frage zu beantworten, inwieweit sich evtl. äquivalente Aussagen über die Gleichheit, bezogen auf beide Konzepte, treffen lassen. D.h., erfüllen bisimulierbare Prozesse auch gleichzeitig die Eigenschaften der kontextuellen Äquivalenz und oder umgekehrt? Oder anders: Sind alle kontextuell äquivalenten Prozesse auch bisimulierbar?

4.2.4 May- und Must-Konvergenz

Bereits im Zusammenhang mit der Definition der Prozesskonvergenz in 3.22 wurde erwähnt, dass über die hier verwendete Definition der Konvergenz *(May-Konvergenz)* hinaus eine „schärfere“ Form des Konvergenzbegriffs – die *Must-Konvergenz* – möglich ist.

Somit mag eine Untersuchung, ähnlich dieser Arbeit, unter Verwendung des strengeren Konvergenzbegriffes für zukünftige Forschungen von Interesse sein.

4.2.5 Kontextuelle Äquivalenz ungleich kontextuelle Äquivalenz

Neben der Möglichkeit zur freien Definition des Syntax des Pi-Kalküls wie auch der Möglichkeit zur Beschränkung der operationalen Semantik und der sich somit ergebenden Vielzahl an unterschiedlich definierten Pi-Kalkülen existiert auch *nicht nur eine* Definition der *kontextuellen Äquivalenz*.
Die Beantwortung der Frage nach dem Verhalten der einzelnen unterschiedlichen Definitionen der kontextuellen Äquivalenz und auf deren Basis getroffener Aussagen zueinander könnte für die zukünftige Forschung interessant sein.
Robin Milner selbst hat in [Mil90] neben Gérad Boudol in [Bou92] andere Definitionen von kontextueller Äquivalenz geliefert, sodass ein Vergleich der einzelnen Definitionen und deren gegenseitige Verträglichkeit für zukünftige Untersuchungen von Interesse sein mag.

Literaturverzeichnis

[Ace07] Aceto, Luca; Ingólfsdóttir, Anna; Larsen, Kim Guldstrand und Srba, Jiri: *Reactive Systems: Modelling, Specification and Verification*, Cambridge University Press, New York, NY, USA, 2007, - ISBN 0521875463

[Bar85] Barendregt, Hendrik. P.: *The Lambda Calculus, Its Syntax and Semantics*, Bd. 103 von *Studies in Logic and the Foundations of Mathematics*, North Holland, November 1985, überarbeitete Aufl., - ISBN 0444875085

[Bez98] Bezem, Marc; Klop, Jan Willem und van Oostrom, Vincent: Diagram Techniques for Confluence. *Information and Computation*, 1998, Bd. 141(2): S. 172 – 204, `http://www.sciencedirect.com/science/article/B6WGK-45J54NW-1S/2/7925ff3c1ea2b1a7ae112567de94cb0a`

[Bou92] Boudol, Gérard: Asynchrony and the Pi-calculus, Research Report RR-1702, INRIA, 1992, `http://hal.inria.fr/inria-00076939/en/`

[BPE02] BPEL4WS, BPML: A Convergence Path toward a Standard BPM Stack, 2002, `http://www.bpmi.org/downloads/BPML-BPEL4WS.pdf`

[Chu32] Church, Alonzo: A set of postulates for the foundation of logic (1). *Annals of Mathematics*, 1932, Bd. 33: S. 346–366

[Chu36] Church, Alonzo: An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics*, 1936, Bd. 58(2): S. 345–363

[Chu41] Church, Alonzo: *The Calculi of Lambda-Conversion*, Princeton University Press, Princeton, 1941

[Cop96] Coplien, James O.: *Software Patterns*, SIGS Management Briefings, SIGS, New York, 1996

[Cos96] Cosmo, Roberto Di: On the Power of Simple Diagrams, In: *RTA '96: Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, Springer-Verlag, London, UK, 1996, S. 200–214

[Har85] Harel, David und Pnueli, Amir: *Logics and Models of Concurrent Systems*, Kap. On the development of reactive systems, NATO Advanced Study Institute on Logics and Models for Verification and Specification of Concurrent Systems, Springer-Verlag, New York, USA, 1985, S. 477–498, - ISBN 0387151818

[Hen07] Hennessy, Matthew: *A Distributed Pi-Calculus*, Cambridge University Press, New York, NY, USA, 2007, - ISBN 0521873304

[Hoa69] Hoare, C. A. R.: An axiomatic basis for computer programming. *Communications of the ACM*, 1969, Bd. 12(10): S. 576–580

[Hoa85] Hoare, C. A. R.: Communicating Sequential Processes. *Communications of the ACM*, 1985, Bd. 21: S. 666–677

[Hon91] Honda, Kohei und Tokoro, Mario: An Object Calculus for Asynchronous Communication, In: *ECOOP '91: Proceedings of the European Conference on Object-Oriented Programming*, Springer-Verlag, London, UK, 1991, S. 133–147

[Hon92] Honda, Kohei und Tokoro, Mario: On Asynchronous Communication Semantics, In: *ECOOP '91: Proceedings of the Workshop on Object-Based Concurrent Computing*, Springer-Verlag, London, UK, 1992, S. 21–51

[Hop06] Hopcroft, John E.; Motwani, Rajeev und Ullman, Jeffrey D.: *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*, Addison Wesley, Juli 2006, 3. Aufl., - ISBN 0321462254

[Joh08] John, Mathias; Ewald, Roland und Uhrmacher, Adelinde M.: A Spatial Extension to the pi Calculus. *Electr. Notes Theor. Comput. Sci.*, 2008, Bd. 194(3): S. 133–148

[Kah09] Kahane, Howard und Cavender, Nancy: *Logic and Contemporary Rhetoric: The Use of Reason in Everyday Life*, Wadsworth Publishing Company, Belmont, Kalifornien, 2009, 11. Aufl., - ISBN 0495804118

[Kle35] Kleene, Stephen, C.: A Theory of Positive Integers in Formal Logic. *American Journal of Mathematics*, 1935, Bd. 57

[Kön10] König, Barbara: Folien zur Vorlesung - Modellierung nebenläufiger Systeme, Wintersemester 2009/2010, `http://www.ti.inf.uni-due.de`

[Kow96] Kowalk, Wolfgang Peter: *System, Modell, Programm*, Spektrum, Akad. Verl., Heidelberg [u.a.], 1996, - ISBN 3827400627

[Kut98] Kutzner, Arne und Schmidt-Schauss, Manfred: A non-deterministic call-by-need lambda calculus, In: *International Conference on Functional Programming*, ACM Press, 1998, S. 324–335

[Kut00] Kutzner, Arne: *Ein nichtdeterministischer call-by-need Lambda-Kalkül mit erratic Choice: Operationale Semantik, Programmtransformationen und Anwendungen*, Dissertation, J. W. Goethe-Universität, Frankfurt am Main, 2000

[Lee03] Lee, Ethan; Salic, Adrian; Krüger, Roland; Heinrich, Reinhart und Kirschner, Marc W: The Roles of APC and Axin Derived from Experimental and Theoretical Analysis of the Wnt Pathway. *PLoS Biol*, Oktober 2003, Bd. 1(1): S. 116–132, http://dx.doi.org/10.1371%2Fjournal.pbio.0000010

[Lei90] Leibniz, Gottfried Wilhelm: *Die philosophischen Schriften von Gottfried Wilhelm Leibniz*, Bd. 7, Weidmannsche Buchhandlung, Berlin, 1875-1890

[Müh04] Mühl, Gero: Vorlesung - Verteilte Systeme - Architekturen und Dienste - Messaging, Sommersemester 2004, http://kbs.cs.tu-berlin.de/staff/muehl/muehl.htm

[Mil82] Milner, Robin: *A Calculus of Communicating Systems*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982, - ISBN 0387102353

[Mil89] Milner, Robin: *Communication and concurrency*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989, - ISBN 0131150073

[Mil90] Milner, Robin: Functions as processes, In: Michael Paterson (Herausgeber) *Automata, Languages and Programming*, Bd. 443 von *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 1990, S. 167–180, http://dx.doi.org/10.1007/BFb0032030, 10.1007/BFb0032030

[Mil92] Milner, Robin; Parrow, Joachim und Walker, David: A Calculus of Mobile Processes, parts I and II. *Information and Computation*, September 1992, Bd. 100(1): S. 1–77

[Mil99] Milner, Robin: *Communicating and Mobile Systems: the Pi-Calculus*, Cambridge University Press, Juni 1999, - ISBN 0521658691

[Nes98] Nestmann, Uwe. und Victor, Björn: Calculi for Mobile Processes - Bibliography and Web Pages. *Bulletin of the European Association for Theoretical Computer Science*, 1998, Bd. 64: S. 139–144

[Ove05] Overdick, Hagen; Puhlmann, Frank und Weske, Mathias: Towards a Formal Model for Agile Service Discovery and Integration, In: *ICSOC Workshop on Dynamic Web Processes at the 3rd International Conference on Service-Oriented Computing*, IBM Tech Report RC 23822, 2005, S. 25–37

[Par81] Park, David: Concurrency and Automata on Infinite Sequences, In: *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, Springer-Verlag, London, UK, 1981, S. 167–183

[Par01] Parrow, Joachim: *Handbook of Process Algebra*, Kap. An introduction to the pi-calculus, Elsevier, 2001, S. 479–543

[Pet62] Petri, Carl Adam: *Kommunikation mit Automaten.*, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962

[Pit97] Pitts, Andrew: Operationally-based theories of program equivalence, In: *Semantics and Logics of Computation*, Cambridge University Press, 1997, S. 241–298

[Puh05] Puhlmann, Frank und Weske, Mathias: Using the Pi-Calculus for Formalizing Workflow Patterns, In: *Business Process Management, 3rd International Conference*, Bd. 3649 von *LNCS*, Springer-Verlag, Berlin, September 2005, S. 153–168, `http://bpt.hpi.uni-potsdam.de/pub/Public/MathiasWeske/bpm2005.pdf`

[Puh07] Puhlmann, Frank: *On the application of a theory for mobile systems to business process management*, Dissertation, Universität Potsdam, Juli 2007

[Reg01] Regev, Aviv; Silverman, William und Shapiro, Ehud: Representation and simulation of biochemical processes using the pi-calculus process algebra. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, 2001: S. 459–470, `http://view.ncbi.nlm.nih.gov/pubmed/11262964`

[Roj00] Rojas, Raúl und Hashagen, Ulf (Herausgeber): *The First Computer - History and Achitectures*, MIT Press, Paderborn, August 2000

[Sab08] Sabel, David: *Semantics of a Call-by-Need Lambda Calculus with McCarthy's amb for Program Equivalence*, Dissertation, J. W. Goethe-Universität, Frankfurt am Main, 2008

[Sab10a] Sabel, David: Skript zur Vorlesung - Nebenläufige Programmierung: Praxis und Semantik, Wintersemester 2009/2010, `http://www.ki.informatik.unifrankfurt.de`

[Sab10b] Sabel, David und Schmidt-Schauss, Manfred: A Context Lemma for Contextual Equivalence in the Pi-Calculus, 2010, Entwurf vom 23. September

[San01] Sangiorgi, Davide und Walker, David: *PI-Calculus: A Theory of Mobile Processes*, Cambridge University Press, New York, NY, USA, 2001, - ISBN 0521781779

[SS03] Schmidt-Schauss, Manfred: FUNDIO: A Lambda-Calculus with a letrec, case, Constructors, and an IO-Interface: Approaching a Theory of unsafePerformIO, Frank report 16, Institut für Informatik, J.W. Goethe-Universität Frankfurt am Main, September 2003

[SS07] Schmidt-Schauss, Manfred: Skript zur Vorlesung - Einführung in die funktionale Programmierung, Wintersemester 2006/2007, `http://www.ki.informatik.unifrankfurt.de`

[Tur36] Turing, Alan M.: On Computable Numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.*, 1936, Bd. 2(42): S. 230–265

[vdA98] van der Aalst, Wil M. P.: The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 1998, Bd. 8(1): S. 21–66

[vdA02] van der Aalst, Wil M. P. und ter Hofstede, Arthur H. M.: YAWL: Yet Another Workflow Language, Techn. Ber. FIT-TR-2002-06, Queensland University of Technology, Brisbane, Australia, 2002

[vO94] van Oostrom, Vincent: Confluence by decreasing diagrams. *Theor. Comput. Sci.*, 1994, Bd. 126(2): S. 259–280

[WKE01] Wilhelm K. Essler, Rosa F. Martínez Cruzado und Labude, Joachim: *Grundzüge der Logik I*, Vittorio Klostermann, Frankfurt am Main, 2001, 5., neu bearbeitete und erw. Aufl.

Index

Symbole

A

B

C

D

G

I

K

L

N

P

R

S

T

V