TECHNISCHE
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik

Bernhard Schätz und Katharina Spies

TUM-I9529
SFB-Bericht Nr.342/16/95 A
Oktober 1995

TUM–INFO–10-95-I29-350/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
Arcisstr. 21 / Postfach 20 24 20
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

Formale Syntax zur logischen Kernsprache der FOCUS-Entwicklungsmethodik *

Bernhard Schätz und Katharina Spies

Institut für Informatik
Technische Universität München
Postfach 20 24 20, D-80290 München
{*schaetz,spiesk*}*@informatik.tu-muenchen.de*

17. Oktober 1995

Zusammenfassung

Mit der Methodik FOCUS wird ein allgemeiner Rahmen zur Spezifikation und Entwicklung verteilter Systeme vorgegeben. Die Systeme werden als Netzwerke bestehend aus sogenannten Basisagenten aufgebaut und im allgemeinen durch Mengen von stromverarbeitenden Funktionen modelliert. Um dem Anwender das Erstellen formaler Spezifikationen zu erleichtern, die Spezifikationen zu vereinheitlichen und die oft vorhandene Hemmschwelle beim Umgang mit komplexen, formalen Methoden zu überwinden, wird in der vorliegenden Arbeit die Syntax der logischen Kernsprache ANDL ("Agent Network Description Language") zur Spezifikation von Agenten festgelegt. Auf der Basis von HOLCF, einer Logik höherer Stufe zur Beschreibung berechenbarer Funktionen, und mit Hilfe der Syntax des Beweisunterstützungswerkzeugs *Isabelle* werden die in FOCUS für Ströme festgelegten Basisoperatoren definiert. Auf der so festgelegten theoretischen Basis wird dann ein leicht verständliche und möglichst einfacher syntaktischer Rahmen für FOCUS festgelegt, der insbesondere eine gut lesbare Schnittstellenbeschreibung umfaßt.

*Diese Arbeit wurde unterstützt vom Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen"

1 Einleitung

Zur Entwicklung und Beschreibung komplexer Aufgabenstellungen und Strukturen, die zudem hohe Anforderungen an die Korrektheit und Zuverlässigkeit der zu erarbeitenden Lösung stellen, werden in zunehmendem Maße formale Beschreibungstechniken und Methoden eingesetzt. Je ausgereifter und komplexer eine formale Methodik jedoch ist, um so mehr stützt sie sich auf eine umfangreiche mathematische Basis. Dies hat zur Folge, daß der mit einer Methodik zur Verfügung gestellte Rahmen nur von geübten und mit dem Begriffsapparat vertrauten Benutzern in seinem ganzen Umfang eingesetzt werden kann. Freiheiten und Alternativen beim Einsatz der formalen Methodik, die einem geübten Benutzer als besonderer Vorteil erscheinen, verwirren Nichtspezialisten oftmals nur und geben zuviel Spielraum, so daß sich ein Einstiegspunkt für eine erste Formalisierung nur sehr schwer erkennen läßt. Da es aber wünschenswert ist, daß auch Nichtspezialisten in der Lage sind, die vorgegebenen Systemanforderungen innerhalb eines formalen Rahmens anzugeben, erscheint es sinnvoll, zu einer formalen Entwicklungsmethodik eine formale Syntax – im Sinne der Syntax einer Programmiersprache – zu definieren. Diese soll auf der einen Seite leicht verständlich und in natürlicher Weise einsetzbar sein und auf der anderen Seite die bereits ausgereiften Möglichkeiten der formalen Methodik nicht einschränken, sondern diese im Idealfall sogar noch erweiteren.
FOCUS ist eine derartige komplexe Entwurfsmethodik zur schrittweisen, formalen Entwicklung verteilter Systeme. Eine Systementwicklung durchläuft mehrere Phasen, und dem Benutzer steht ein ausgereifter mathematischer Apparat zur Verfügung, der eine Vielzahl aufeinander abgestimmter Formalismen und Techniken zur Erstellung der Spezifikationen anbietet. Die verteilten Systeme werden als Netzwerke von Komponenten, die asynchron über unbeschränkte, gerichtete Kanäle kommunizieren, spezifiziert und durch Mengen stromverarbeitender Funktionen modelliert. Motiviert durch die oben beschriebenen Beobachtungen im Umgang mit FOCUS wird hier die formale Syntax zur Erstellung funktionaler Spezifikationen definiert.
Basierend auf einem einheitlichen semantischen Modell wird in diesem Bericht eine möglichst kleine, logische Kernsprache definiert, mit der zulässige Bausteine in FOCUS beschrieben werden können. Diese Kernsprache erlaubt die syntaktische Beschreibung funktional spezifizierter FOCUS-Komponenten, die mit Hilfe spezieller FOCUS-Operatoren und der Prädikatenlogik spezifiziert werden. Im Rahmen von FOCUS müssen bei der Entwicklung des vollständigen strukturellen Aufbaus und des gewünschten Detaillierungsgrades eines korrekten verteilten Systems eine Vielzahl oftmals komplexer Beweise geführt werden. Um die Durchführung dieser Beweise durch den Einsatz eines Beweisunterstützungswerkzeugs zu erleichtern, werden die elementaren, stromverarbeitenden Funktionen mittels der *Isabelle*-Syntax ([Pau94b]) beschrieben. Dies wird ermöglicht, indem auf HOLCF (Higher Order Logic of Computable Functions) [Reg94] und dort bereits definierte Theorien (`ccc1` und `stream`) zurückgegriffen wird. Da in HOLCF auch die Prädikatenlogik höherer Stufe angeboten wird, ist mit der Kernsprache eine syntaktische Beschreibung funktional spezifizierter Focus-Komponenten möglich, und zwar in dem Umfang, in dem sie bisher benutzt wurden.
Auf der Basis von HOLCF wird also eine FOCUS-Theorie `Focus` entwickelt; für die Kernsprache ergibt sich jedoch auf diesem Weg zunächst keine leicht handhabbare, technische

Beschreibungssprache. Durch die Einführung passender syntaktischer Abkürzungen kann die *Isabelle*-spezifische Notation jedoch auf einer Zwischenebene wieder verborgen und so die Benutzerfreundlichkeit gesteigert werden. Besonderes Augenmerk wird hierbei auf eine gut verständliche Beschreibung der Schnittstellen eines Bausteins gerichtet. Auf der Basis der so festgelegten logischen Kernsprache werden dann in zukünftigen Arbeiten Erweiterungen der Syntax vorgenommen, beispielsweise durch die Anbindung praxisorientierter Beschreibungstechniken (Tabellen, State-Charts, etc.).
Der vorliegende Bericht ist wie folgt aufgebaut: **Kapitel 2** gibt eine kurze Zusammenfassung und einen Überblick über die allgemeinen Grundlagen, die zum Verständnis der später definierten Syntax notwendig sind. Hier werden ein kurzer Überblick über die Methodik FOCUS und eine knappe Beschreibung der wesentlichen Charakteristika von *Isabelle* und HOLCF gegeben. In **Kapitel 3** werden die für die Realisierung der Syntax in HOLCF und damit die für die Definition der HOLCF-Theorie `Focus` erforderlichen Grundlagen erläutert und definiert. Hierauf aufbauend werden in **Kapitel 4** die FOCUS-Operatoren definiert und in **Kapitel 5** ein Beispiel zur Anwendung der so festgelegten – an HOLCF und *Isabelle* orientierten – Syntax gegeben. **Kapitel 6** enthält die Festlegung der FOCUS-Syntax zur Beschreibung von Basisagenten einschließlich einer Veranschaulichung dieser syntaktischen Festlegungen an einem Beispiel. Basisagenten werden in FOCUS als kleinste, nicht weiter strukturierbare Bausteine der Systeme verstanden. Zum Aufbau und zur Spezifikation von verteilten Systemen ist es daher notwendig festzulegen, wie verteilte Systeme aus den Basisagenten zusammengesetzt werden. Die Syntax hierzu, einschließlich eines Beispiels, findet sich in **Kapitel 7**. **Kapitel 8** beinhaltet abschließende Bemerkungen, einen Ausblick auf geplante Arbeiten bzgl. der Syntax und einen Vorschlag für eine mögliche Erweiterung der bisher vorgestellten Syntax.

2 Allgemeine Grundlagen

2.1 Die Entwicklungsmethodik FOCUS

Focus (siehe [BDD+92] und [BDD+93]) ist eine Entwurfsmethodik zur schrittweisen, formalen Entwicklung verteilter Systeme. Sie ist formal ausgerichtet und mathematisch fundiert, so daß insbesondere Verfeinerungsschritte und der Nachweis der Korrektheit dieser Schritte systematisch und unter Verwendung mathematischer Beweismethoden durchgeführt werden können. Es stehen eine Anzahl aufeinander abgestimmter Formalismen, unterschiedliche Richtlinien für die systematische Durchführung der Systementwicklung sowie ein ausgereiftes Verfeinerungskonzept zur Verfügung. Generell werden verteilte Systeme in diesem Entwicklungsrahmen als Netzwerke von Komponenten, die asynchron über unbeschränkte, gerichtete Kanäle kommunizieren, modelliert.
Die Entwicklung eines verteilten Systems ist gemäß FOCUS in drei charakteristische Phasen aufgegliedert. In der **Anforderungsphase** wird eine erste Formalisierung der informellen Beschreibung entwickelt, die dann den weiteren Phasen als Grundlage dient. In diesem Stadium werden Spezifikationen entweder als Trace- oder als funktionale "Black Box"-Spezifikationen formuliert. Der Übergang zwischen diesen Paradigmen ist formal fundiert und korrektheitserhaltend.

Die **Designphase** umfaßt den wesentlichen Teil der Entwicklung zu einem *verteilten* System; der strukturelle, interne Aufbau des Systems wird entwickelt, und die Verfeinerung in den gewünschten Detaillierungsgrad wird vorgenommen. Die Spezifikationen basieren hier auf einer einheitlichen denotationellen Semantik, in der Spezifikationen durch Mengen stromverarbeitender Funktionen modelliert werden. Jede dieser Funktionen beschreibt ein mögliches, zulässiges Verhalten. Ein verteiltes System wird durch ein Netzwerk von Basisagenten modelliert; als Basisagenten werden im weiteren also solche Komponenten verstanden, die im Sinne der Modellierung durch Netzwerke intern nicht weiter strukturiert sind. Zur Spezifikation eines Netzwerkes stehen spezielle Operatoren für die Komposition stromverarbeitender Funktionen zur Verfügung. Funktional spezifizierte Agenten können auf diese Weise sequentiell, parallel oder mittels Rückkopplung zu einem Netzwerk verknüpft werden. Zur Erstellung der Spezifikationen in der Designphase stehen Paradigmen, wie z.B. funktionale und relationale Spezifikationen, und verschiedene Spezifikationsstile, wie z.B. Assumption/Commitment- und Gleichungsspezifikationen zur Verfügung. Aufgrund der charakteristischen Eigenschaften dieser Varianten können sie zur Lösung spezieller Probleme passend eingesetzt werden. Die Schnittstelle jedes Agenten wird durch eine endliche Anzahl von Ein- und Ausgabe-Kanälen, sowie die Festlegung der Typen der Nachrichten, die über diese Kanäle ausgetauscht werden, definiert. Die Kommunikationsgeschichte der Kanäle wird mathematisch durch endliche oder unendliche Nachrichtenströme modelliert. Für den Datentyp der Ströme ist eine Anzahl von Operatoren wie beispielsweise der leere Strom, Präfizes, das erste Element oder der Rest eines Stromes oder einige Verknüpfungsoperatoren, definiert.

In der **Implementierungsphase** wird die in der Designphase entwickelte Spezifikation in eine der zur Verfügung stehenden Zielsprachen überführt. Hierfür wird die funktionale Spezifikation in eine speziell für die gewünschte Zielsprache charakterisierte funktionale Spezifikation – diese Spezifikationen umfassen sowohl spezielle syntaktische Anforderungen als auch auf die Zielsprache zugeschnittene semantische Eigenschaften – gebracht und anschließend unter Ausnutzung vorgegebener Überführungsregeln in die Syntax der entsprechenden Zielsprache umgesetzt.

Bei der formalen Entwicklung verteilter Systeme wird der gewünschte Detaillierungsgrad durch schrittweise Verfeinerung des Systems bzw. der einzelnen Agenten erreicht. FOCUS bietet hierfür ein mächtiges, kompositionales Verfeinerungskonzept und Verfeinerungskalküle an (z.B. [Bro93]). Die drei wesentlichen Verfeinerungskonzepte werden im Folgenden kurz erläutert. Das Ziel der **Verhaltens-Verfeinerung** besteht in der Elimination von Unterspezifikationen. Dies wird beispielsweise zur Spezifikation fehlertoleranten Verhaltens benötigt. Auf der semantischen Ebene wird diese Verfeinerung durch Mengeninklusion modelliert. Mit der **Schnittstellen-Verfeinerung** wird die Schnittstelle eines Agenten verfeinert, indem die Anzahl der Kanäle oder auch die Typen der Kanäle verändert werden. Das Konzept der **strukturellen Verfeinerung** erlaubt es, den strukturellen Aufbau des verteilten Systems dadurch zu entwickeln, daß Komponenten zu Netzwerken von Komponenten verfeinert werden.

Mit FOCUS steht dem Anwender somit ein mathematisch fundierter und konzeptuell abgerundeter Rahmen zur Verfügung, der vollständige Systementwicklungen ausgehend von einer abstrakten Anforderungsspezifikation bis hin zur Implementierung ermöglicht.

Dessen ungeachtet sind Spezifikationen, die in dem hier kurz vorgestellten Sinne erstellt

wurden, aufgrund der speziellen Notationen und einer daraus resultierenden möglicherweise, vor allem mathematisch, komplexen Darstellung für Nicht-Spezialisten oft schwer lesbar und somit auch nicht immer sofort nachvollziehbar. Aus diesem Grund erscheint es wichtig, zur Steigerung der Praktikabilität und folglich auch der Akzeptanz formaler Methoden – in diesem Fall FOCUS – weitere bereits bekannte Formalismen, Techniken und eine festgelegte Syntax für die Erstellung der formalen Spezifikation anzubieten. Hierbei sollte jedoch gewährleistet sein, daß diese "neuen" Formalismen adäquat sind und zielgerichtet in der entsprechenden Methodik eingesetzt werden können. In diesem Sinne werden sich die zukünftigen Arbeiten mit Erweiterungen von FOCUS befassen, wobei sich das Arbeitsprogramm verstärkt auf die pragmatische Umsetzung und Anwendung der Konzepte konzentriert. So sollen in der Praxis eingesetzte Spezifikationstechniken wie beispielsweise Tabellen, Diagramme und State-Charts unter Beibehaltung der für FOCUS festgelegten semantischen Basis in die Methodik einbezogen werden. Weiterhin sollen einzelne Entwicklungsschritte schematisiert und Leitlinien entwickelt werden, um so die Übergänge zwischen den Entwicklungsphasen und den Formalismen, die durchzuführenden Beweise sowie die Spezifikationen spezifischer und häufig wiederkehrender Problemstellungen zunehmend zu erleichtern. Mit der hier vorliegenden Arbeit wird die formale Syntax für eine logische Kernsprache von FOCUS festgelegt, die die funktionale Spezifikation von Agenten mit den zugehörigen Operatoren umfaßt. Da mit der hier vorgestellten Syntax vor allem die Beschreibung strukturierter, verteilter Systeme unterstützt werden soll, beschränkt sich die Festlegung der formalen Syntax auf die Umsetzung der für diese Entwicklungsphasen charakteristischen Modellierung von Agenten bzw. Systemen durch Mengen stromverarbeitender Funktionen. Mit der hier definierten Syntax wird somit dem Anwender eine leicht handhabbare und gut verständliche Beschreibungssprache an die Hand gegeben. Gleichzeitig wird auch durch ihre Anbindung an *Isabelle* und HOLCF für die Zukunft maschinenunterstütztes Beweisen mit FOCUS möglich.

2.2 *Isabelle* und HOLCF

Mit *Isabelle* steht ein generischer interaktiver Theorembeweiser zur Unterstützung des formalen Beweisprozesses in unterschiedlichen Objektlogiken wie z.B. HOLCF zur Verfügung ([Pau94a], [Pau94b]). *Isabelle* basiert dabei auf vier wesentlichen Eigenschaften:

- Der Beschreibung der Syntax der Objektlogik basierend auf dem getypten Lambda-Kalkül
- Dem Meta-Kalkül zur Beschreibung der Schlußregeln der Objektlogik
- Der Kombination von Regeln mittels Unifikation und Resolution
- Taktiken zur interaktiven bzw. teilautomatisierten Anwendung von Schlußregeln

Mittels der Beschreibung der Syntax der Objektlogik wird der syntaktische Aufbau von Ausdrücken in der jeweiligen Objektlogik definiert. Dabei kommt wesentlich das Typenkonzept von *Isabelle* zum Tragen, das allen Ausdrücken einen entsprechenden Typ zuordnet. Damit besteht die Definition der Syntax der Objektlogik zum einen aus der Einführung

von geeigneten Typen (z.B. Termen und Formeln im Falle der Definition der Prädikatenlogik erster Stufe) für die entsprechenden syntaktischen Kategorien der Objektlogik. Zum anderen werden für alle Symbole der Logik (z.B. die logischen Konnektoren und Quantoren im Falle der Prädikatenlogik erster Stufe) entsprechende Konstanten durch die Zuordnung von n-stelligen gecurriedten Funktionstypen zu einer n-stelligen Operation der Objektlogik eingeführt.
Ist die syntaktische Struktur einer Objektlogik festgelegt, so kann der semantische Gehalt dieser Objektlogik durch die Angabe entsprechender Axiome definiert werden. Dies geschieht durch die Erweiterung der vordefinierten Metalogik von *Isabelle*, die im wesentlichen eine intuitionistische Logik höherer Stufe ist. Die Konnektoren der Metaebene sind dabei die *Implikation*, die *universelle Quantifizierung*, sowie die *Gleichheit*. Um den Regeln der Objektlogik eine semantische Definition zuzuordnen, wird hier die Verbindung zwischen den Regeln der Meta- und der Objektlogik hergestellt (z.B. wird die Implikation der Prädikatenlogik erster Stufe auf die Implikation der Metalogik zurückgeführt).
Da sich die definierte Objektlogik auf die Metalogik abstützt, wäre es notwendig, die entsprechenden Beweise in der Metalogik auszuführen, um formale Beweise in der Objektlogik zu erhalten. Da dies jedoch zu aufwendigen Beweisen führt, werden statt dessen aus der Metalogik abgeleitete Regeln verwendet. Hier eignet sich besonders die Resolution für den Fall des Rückwärtsschließens. Als zentraler Mechanismus kommt dabei die Unifikation zum Einsatz, der die Instantiierung von Objektregeln für die Anwendung im Beweisprozeß ermöglicht.
Isabelle erlaubt die Herleitung eines Theorems durch die schrittweise Bearbeitung mittels Beweiszuständen. Durch die Anwendung einer Taktik wird der aktuelle Zustand, der i.a. aus mehreren Unterzielen besteht, in einen entsprechenden neuen Zustand übergeführt. Durch die Kombination solcher Taktiken können auf die Objektlogik zugeschnittene, komplexere automatische Beweisprozeduren definiert werden.
Neben den "klassischen" Logiken, wie z.B. Prädikatenlogik erster Stufe (FOL) oder höherer Stufe (HOL), der Logik berechenbarer Funktionen (LCF) oder der Zemelo-Fraenkel-Mengentheorie (ZFL), steht auch eine Logik höherer Stufe berechenbarer Funktionen ("Higher Order Logic of Computable Functions", HOLCF) zur Verfügung (siehe [Reg94]). Diese stellt eine konservative Erweiterung der Logik höherer Stufe um die Konzepte der Theorie berechenbarer Funktionen dar. Neben der Einführung der Funktionen als Konzept ist darüberhinaus insbesondere eine Erweiterung der Logik höherer Stufe um die Konzepte der Bereichstheorie gegeben. Damit ist HOLCF eine geeignete Basis für die Formalisierung von FOCUS, da die wesentlichen Konzepte

- Logik höherer Stufe
- Formalisierung des Strombereichs
- Formalisierung stetiger Funktionen

angeboten werden. Da HOLCF die Definition von Datentypen mittels Bereichskonstruktoren und -gleichungen erlaubt, stehen neben den Strömen auch noch eine Vielzahl weiterer Datentypen zur Verfügung, die die Anbindung von Spezifikationstechniken außerhalb des Bereichs der reaktiven verteilten System erlauben (siehe z.B. [Slo95]).

3 Grundlagen der Realisierung

Zur Realisierung von FOCUS mittels HOLCF kann auf eine Theorie zurückgegriffen werden, die die Vorzüge der Spezifikation mittels Konstrukten höherer Stufe mit den Konzepten der berechenbaren Funktionen vereint. Zur Realisierung wird daher eine Erweiterung der HOLCF-Theorie zu einer Focus-Theorie vorgenommen. Diese stellt einen allgemeinen Kern zur Spezifikation und Verifikation von einfachen stromverarbeitenden Funktionen zur Verfügung. In [Spi94] wird das Vorgehen für die Entwicklung der Syntax von Focus festgelegt. Entsprechend wird in der vorliegenden Arbeit die logische Kernsprache ANDL definiert; aufbauend auf diesem Kern sollen dann in späteren Arbeiten geeignete Erweiterungen, wie zum Beispiel zur Darstellung von

- gezeiteten stromverarbeitenden Funktionen
- zustandsbasierten stromverarbeitenden Funktionen

entwickelt werden.
Aufgrund der Tatsache, daß der mathematische Rahmen, in dem die verteilten Systeme mit FOCUS modelliert werden – also die stromverarbeitenden Funktionen und die in [Stø94] aufgeführten Operatoren – in dem bisher genutzten Umfang beibehalten bleiben soll, ergibt sich der in diesem Bericht gewählte Weg zur Definition der Sprache ANDL. Die mit HOLCF zur Verfügung stehenden Konzepte der Logik höherer Stufe und berechenbarer Funktionen liefern die logische Basis zur Umsetzung der FOCUS-Modellierung. Damit müssen in diesem Kapitel die Grundlagen der Realisierung von ANDL darin bestehen, FOCUS und die HOLCF-Konzepte aneinander anzupassen und die stromverarbeitenden Funktionen und FOCUS-Operatoren im Rahmen der HOLCF-Konzepte (insbesondere als Operationen, Funktionen oder Relationen) zu definieren bzw. zu konstruieren.

3.1 Typisierung

In der bisherigen Darstellung von FOCUS wurde ein ungetypter Kalkül höherer Stufe verwendet. Zur Beschreibung von Funktionen und Relationen wurde daher massiv von der Mengennotation Gebrauch gemacht.
Während mit *Isabelle* ([Pau94a]) ein Beweisunterstützungswerkzeug mit getypter Metalogik vorliegt, wurde FOCUS bisher im allgemeinen ohne explizite Typisierung verwendet. Die Typisierung von *Isabelle* überträgt sich naturgemäß auch auf die verwendeten Objektlogiken. Dies bedeutet, daß bei einer Implementierung von FOCUS mittels *Isabelle* von der ungetypten Beschreibung von Objekten abgegangen werden muß.
Ströme werden in der Focus-Theorie durch den Bereich der HOLCF-Ströme dargestellt. Diese entsprechen einem polymorphen Datentyp, der *partielle* Ströme oder Ströme unendlicher Länge beschreibt. Unter partiellen Strömen sind dabei solche Ströme zu verstehen, die durch das $\bot$-Element des zugehörigen Bereichs abgeschlossen werden.
In der bisherigen Version von Focus sind Ströme als die Vereinigung endlicher und unendlicher Spuren (Listen) definiert. Da *Isabelle* eine typisierte Logik verwendet, würde eine direkte Übersetzung dieser Sichtweise in HOLCF zur Folge haben, daß die Mengen M^*, M^∞ und M^ω durch drei unterschiedliche Typen dargestellt werden. Dies würde bedeuten,

daß endliche und unendliche Ströme keine Elemente aus M^ω wären. Die Alternative zu dieser Art der Umsetzung, die im vorliegenden Ansatz verwendet wird, besteht nun darin, den Typ der M^ω-Ströme einzuführen, und die Mengen der endlichen und unendlichen Strömen durch Teilmengenbildung auszuzeichnen.

3.2 Funktionen, Operationen und Relationen

In HOLCF stellen Operationen, Relationen und Funktionen unterschiedliche Typen dar. Unter Funktionen sind hierbei Funktionen im Sinne von *Isabelle-HOL* (vgl. [Pau94a]) zu verstehen; Relationen sind totale bool-wertige Funktionen; Operationen stellen einen eigenen HOLCF-Typ dar, und entsprechen den stetigen Funktionen.

Operationen: Den Operationen entsprechen in der `Focus`-Theorie die stetigen Funktionen, die aus den in diesem Abschnitt beschriebenen Grundoperationen unter Verwendung der Λ-Abstraktion aufgebaut sind. Als entsprechender Typ-Konstruktor wird der HOLCF-Konstruktor "`->`" verwendet. Dementsprechend wird eine stromverarbeitende Funktion f, die Ströme natürlicher Zahlen auf ebensolche abbildet, mittels der Operation

```
f:: dnat stream -> dnat stream
```

dargestellt.

Funktionen: Den Funktionen entsprechen in der hier verwendeten Terminologie, wie oben bereits angesprochen, die *HOL*-Objekte der Funktionen. Als Typkonstruktor wird in der `Focus`-Theorie entsprechend der *HOL*-Typkonstruktor "`=>`" für Abbildungen verwendet. Zur Konstruktion dieser Funktionen stehen die entsprechenden *HOL*-Konstrukte zur Verfügung. Eine allgemeine Abbildung von Strömen natürlicher Zahlen wird hier, im Gegensatz zur obigen Operation, mittels

```
F:: dnat stream => dnat stream
```

beschrieben.

Relationen: Relationen sind, wie oben angesprochen, bool-wertige Funktionen; sie verwenden daher den *HOL*-Typ `bool` und werden entsprechend den Funktionen konstruiert. Eine zweistellige Relation P auf Strömen natürlicher Zahlen wird daher durch die boolwertige Funktion

```
P:: dnat stream => dnat stream => bool
```

dargestellt.

3.3 Karthesisches Produkt

Stromverarbeitende Funktionen operieren auf Tupeln von Strömen. Zur Realisierung in *Isabelle* muß dazu ein geeigneter Tupel-Typ in der entsprechenden Logik zur Verfügung

gestellt werden. HOLCF führt dazu den karthesisches Produkttyp ein. Der zugehörige Typkonstruktor wird durch den HOLCF-Konstruktor "*" dargestellt. Damit läßt sich beispielsweise der Typ einer Funktion f, die Paare von Strömen natürlicher Zahlen auf ebensolche Paare abbildet, definieren als:

```
f:: dnat stream * dnat stream -> dnat stream * dnat stream
```

Auf dem Typ des karthesischen Produkts werden folgende Operationen zur Verfügung gestellt:

(x_1, x_2): Die Bildung des stetigen karthesischen Produkts zweier Elemente wird durch die HOLCF-Operation `cpair‘x1‘x2` dargestellt. Darüberhinaus kann als alternative Notation die Klammerschreibweise mittels `<x1,x2>` verwendet werden;[1] diese bindet nach rechts.[2]

$fst(x)$: Die Abbildung eines karthesischen Produkts auf die erste Komponente wird durch die HOLCF-Operation `cfst‘x` ("carthesian first") dargestellt.

$snd(x)$: Die Abbildung eines karthesischen Produkts auf die zweite Komponente wird durch die HOLCF-Operation `csnd‘x` ("carthesian second") dargestellt.

4 Definition der FOCUS-Operatoren

Im Folgenden werden den in der Liste [Stø94] aufgeführten Operatoren der FOCUS-Notation die entsprechenden Operationen, Funktionen und Relationen der `Focus`-Theorie gegenübergestellt. Dabei werden drei unterschiedlichen Gruppen von Operatoren unterschieden:

Die Basisoperatoren: Mit diesem Begriff werden solche Operatoren bezeichnet, die direkt auf HOLCF-Operationen des Datentyps der Ströme abgebildet werden können.

Zusammengesetzte Operatoren: Darunter sind solche Operatoren zu verstehen, die unter Verwendung von Basisoperationen und Operationen anderer Grundtypen durch Abstraktion und Funktionsanwendung definiert werden können.

Nichtstetige Operatoren: Operatoren, die keine berechenbaren Funktionen darstellen, bedürfen einer besonderen Behandlung, da sie nicht wie die zusammengesetzten Operatoren aufgebaut werden können; sie werden daher in einer eigenen Gruppe zusammengefaßt.

4.1 Die Basisoperatoren

Basisoperatoren finden sich, mit anderer Bezeichnung, direkt im Datentyp der Ströme wieder. Es ist daher keine Definition dieser Operatoren nötig. Statt dessen wird hier nur die entsprechende HOLCF-Darstellungsweise dieser Operatoren wiedergegeben:

[1] Für die nichtstetige karthesisches Tupelbildung wird die Darstellungsform `(x1,x2)` benutzt; diese findet jedoch i.a im Rahmen von FOCUS keine Anwendung.

[2] Damit entspricht beispielsweise `<x,y,z>` der Darstellung `cpair‘x‘cpair‘y‘z`.

$\langle\rangle$: Der leere Strom wird durch das $\perp$-Element des Bereichs der partiellen Ströme dargestellt. Die *Isabelle*-Notation dafür ist `UU`.

$ft(s)$: Die stetige Funktion, die das erste Element eines Stromes liefert, wird durch die HOLCF-Operation `shd‘s` ("stream head") dargestellt.

$rt(s)$: Die stetige Funktion, die den Rest eines Stromes nach Abtrennen des ersten Elementes liefert, wird durch die HOLCF-Operation `stl‘s` ("stream tail") dargestellt.

$x\&xs$: Die stetige Funktion, die das Voranstellen eines Elementes vor einen Strom beschreibt, wird durch die HOLCF-Operation `scons‘x‘xs` ("stream constructor") dargestellt. Hierfür exisitiert alternativ dazu die Infixschreibweise `x&&xs`.

$s \sqsubseteq t$: Die Präfixrelation wird durch die "Schwächer"-Ordnung `s << t` des Bereichs der Ströme dargestellt.

4.2 Zusammengesetzte Operatoren

Für die FOCUS-Operatoren $A©t$ ("Filter") und $\#t$ ("Länge") sind im Datentyp der Ströme von HOLCF keine entsprechenden Operationen definiert. Aus diesem Grund müssen sie durch Abstraktion und Funktionsanwendung aus den Basisoperationen und Operationen anderer Datentypen, wie z.B. den natürlichen Zahlen, aufgebaut werden. Daher wird neben der Beschreibung dieser Operatoren auch ihre Realisierung in HOLCF-Syntax angegeben:

$A©t$: Die stetige Funktion, die das "Herausfiltern" von Elementen einer Menge A aus einem Strom beschreibt, wird durch den FOCUS-Operator $A©t$ dargestellt. Die vorliegende Realisierung als HOLCF-Operation `filter‘P‘t` verwendet eine allgemeinere Darstellung des Filters, wobei das erste Argument nicht eine Menge, sondern eine berechenbare, wahrheitswertige Funktion ist. Eine Implementierung von endlichen Mengen ist dann beispielsweise als Instantiierung mittels strikter Listen möglich. Die Darstellung in HOLCF-Syntax lautet:[3]

```
filter    :: ('a -> tr) -> 'a stream -> 'a stream

filter == fix‘(LAM h. LAM P s. stream_when‘(
                LAM x xs. If P‘x
                          then x&&h‘P‘xs
                          else h‘P‘xs
                          fi)‘s)
```

$\#t$: Die Funktion, die die Länge eines Stromes bestimmt, wird durch die HOLCF-Operation `len‘t` dargestellt. Um eine möglichst durchgängige Einbettung in HOLCF zu erhalten, wird ein neuer `lnat` Datentyp für geordnete natürliche Zahlen mit einem ausgezeichneten ∞-Element eingeführt. Die Darstellung in HOLCF-Syntax lautet:

[3] Die in der Darstellung verwendete Operation `stream_when` operiert auf einer Operation vom Typ `‘a -> ‘a stream -> ‘a stream` sowie einem Objekt vom Typ `‘a stream`; sie liefert $\perp$, falls das Objekt selbst $\perp$ ist, anderenfalls die Anwendung der Operation auf das erste Element sowie den Rest des Objekt.

```
len       :: 'a stream -> lnat

len == fix'(LAM h. LAM s. stream_when'(
             LAM x xs. lsucc'(h'xs))'s)
```

Die beiden in [Stø94] aufgeführten Operatoren $rt_j(s)$ und $s.j$ stellen ebenfalls keine Basisoperatoren dar. Sie können in entsprechender Weise wie die Filter- und Längenfunktionen definiert werden; auf die ausführliche Darstellung in HOLCF-Syntax soll hier jedoch verzichtet werden:

$rt_j(s)$: Die stetige Funktion, die die wiederholte Anwendung der Funktion $rt(s)$ beschreibt, kann direkt durch Anwendung des Iterationsfunktionals `iterate` auf die Funktion `stl` dargestellt werden:

$s.j$: Die stetige Funktion, die das j-te Element eines Stromes liefert, kann auf folgende Weise definiert werden:

$$s.j = ft(rt_j(s))$$

Die HOLCF-Darstellung ergibt sich direkt aus den bisher definierten Operatoren.

4.3 Nichtstetige Operatoren

Die Methodik FOCUS ist auf die Entwicklung stetiger stromverarbeitender Funktionen ausgerichtet. Daher bietet es sich für eine durchgängige Entwicklung an, die Basisoperatoren der `Focus`-Theorie durch stetige Operationen zu realisieren. Jedoch sind nicht alle Operatoren aus [Stø94] stetige Operatoren. Insbesondere gilt dies für den Konkatenationsoperator, der nichtmonoton ist:

$$\begin{aligned} \langle\rangle \circ \langle b\rangle &= \langle b\rangle \\ \langle a\rangle \circ \langle b\rangle &= a\&\langle b\rangle \\ (\langle\rangle, \langle b\rangle) &\sqsubseteq (\langle a\rangle, \langle b\rangle) \\ \langle b\rangle &\not\sqsubseteq a\&\langle b\rangle \end{aligned}$$

Diese Operatoren müssen auf allgemeine Funktionen im Sinne von HOL abgebildet werden. Auch ohne die Fundierung durch die Verwendung einer HOLCF-Operation sollte hier jedoch eine möglichst sichere Vorgehensweise gewählt werden. Eine Formalisierung in HOLCF kann im Fall des Konkatenationsoperators durch die Verwendung des Hilbertoperators ϵ ausreichend sicher gestaltet werden:

$$\begin{aligned} s_1 \circ s_2 = \epsilon s.\ & (\#s_1 = \infty \wedge s = s_1) \vee \\ & (s_1 \sqsubseteq s \wedge rt_{\#s_1}(s) = s_2) \end{aligned}$$

Die entsprechende HOLCF-Darstellung lautet:

```
sconc  :: 'a stream => 'a stream => 'a stream

sconc = % s1 s2. @ s. (len's1 = infinity & s = s1) |
                      (s1 << s &
                       rt'(len's1)'s = s2)
```

5 Ein Beispiel

Anhand eines einfachen Beispiels soll die Anwendung der oben definierten Syntax der Kernsprache demonstriert werden. Der Agent "fork" besitzt, wie in Abbildung 1 gezeigt, zwei Eingabekanäle x und y, sowie zwei Ausgabekanäle v und w, auf denen natürliche Zahlen empfangen und gesendet werden. Empfängt der Agent auf Kanal x die Nachricht "0", so wird die nächste, auf Kanal y anstehende Nachricht über Kanal v ausgegeben; anderenfalls wird die über Kanal y empfangene Nachricht über Kanal w ausgegeben.

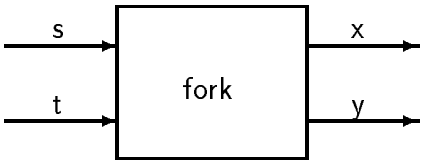

Abbildung 1: Der fork-Agent

FOCUS erlaubt die Beschreibung dieses Agenten auf unterschiedliche Arten. Eine Möglichkeit besteht darin, eine programmiersprachlich orientierte Aufschreibung zu verwenden (vgl. z.B. [Ded92]). In der bisher benutzten FOCUS-Notation (z.B. [Bro94], [Ded92]) lautet eine mögliche Spezifikation dieses Agenten:[4]

$$\begin{aligned}\mathrm{fork}(s,t) = \ & \textbf{if } \mathrm{ft}\,.s = 0 \\ & \textbf{then } (\mathrm{ft}\,.t \,\&\, \mathrm{fst}\,.\,\mathrm{fork}(\mathrm{rt}\,.s, \mathrm{rt}\,.t), \mathrm{snd}\,.\,\mathrm{fork}(\mathrm{rt}\,.s, \mathrm{rt}\,.t)) \\ & \textbf{else } (\mathrm{fst}\,.\,\mathrm{fork}(\mathrm{rt}\,.s, \mathrm{rt}\,.t), \mathrm{ft}\,.t \,\&\, \mathrm{snd}\,.\,\mathrm{fork}(\mathrm{rt}\,.s, \mathrm{rt}\,.t))\end{aligned}$$

Diese Spezifikation läßt sich folgendermaßen direkt in die hier definierte Syntax übersetzen:

```
fork    ::(dnat stream * dnat stream) -> (dnat stream * dnat stream)

fork =
  fix'(LAM f.LAM z.
      If deq‘shd‘cfst‘z‘dzero
      then cpair‘scons‘shd‘csnd‘z
                      ‘cfst‘f‘cpair‘stl‘cfst‘z‘stl‘csnd‘z
                ‘csnd‘f‘cpair‘stl‘cfst‘z‘stl‘csnd‘z
      else cpair‘cfst‘f‘cpair‘stl‘cfst‘z‘stl‘csnd‘z
                ‘scons‘shd‘csnd‘z
                      ‘csnd‘f‘cpair‘stl‘cfst‘z‘stl‘csnd‘z
      fi)
```

Die entsprechende Darstellung unter Verwendung des Infixoperators zur Bildung des karthesischen Produkts und des Stromkonstruktors lautet dabei:

[4] Während in den Abschnitten 3.3 und 4.1 die Funktionanwendung mittels Klammerschreibweise geschrieben wurde, wird hier bei Basisoperatoren zur besseren Lesbarkeit die Applikation durch den "."-Operator verwendet.

```
fork =
  fix‘(LAM f.LAM z.
      If shd‘cfst‘z === dzero
      then <shd‘csnd‘z&&cfst‘(f‘<stl‘(cfst‘z),stl’(csnd‘z)>),
            csnd‘(f‘<stl‘(cfst‘z),stl‘(csnd‘z)>)>
      else <cfst‘(f‘<stl‘(cfst‘z),stl‘(csnd‘z)>),
           shd‘csnd‘z&&csnd‘(f<stl‘(cfst‘z),stl‘(csnd‘z)>)>
      fi)
```

Wird schließlich noch das `let`-Konstrukt zur Verdeutlichung der Darstellung verwendet, so ergibt sich

```
fork =
  fix‘(LAM f.LAM z.
      let s = cfst‘z, t = csnd‘z in
      If shd‘s === dzero
      then <shd‘t&&cfst‘(f‘<stl‘s,stl‘t>),
            csnd‘(f‘<stl‘s,stl‘t>)>
      else <cfst‘(f‘<stl‘s,stl‘t>),
           shd‘t&&csnd‘(f<stl‘s,stl‘t>)>
      fi)
```

Eine zweite Möglichkeit zur Spezifikation des Agenten “fork” besteht darin, die Menge der möglichen Implementierungen prädikativ zu charakterisieren. Die bisher bekannte FOCUS-Notation liefert beispielsweise das folgende rekursive Prädikat:

$$
\begin{aligned}
\mathrm{Pfork}(f) = \;& (f(\langle\rangle, t) = (\langle\rangle, \langle\rangle)) \wedge \\
& (f(s, \langle\rangle) = (\langle\rangle, \langle\rangle)) \wedge \\
& (\forall a, b, s1, s2 : \exists h : \\
& \quad \mathrm{Pfork}(h) \wedge \\
& \quad (a = 0 \Rightarrow f(a \& s1, b \& s2) = (b \& \mathrm{fst}\,.h(s1, s2), \mathrm{snd}\,.h(s1, s2))) \wedge \\
& \quad (a \neq 0 \Rightarrow f(a \& s1, b \& s2) = (\mathrm{fst}\,.h(s1, s2), b \& \mathrm{snd}\,.h(s1, s2))))
\end{aligned}
$$

Dieses kann analog direkt in ein HOLCF-Prädikat übersetzt werden:

```
Pfork    ::((dnat stream * dnat stream) ->
                (dnat stream * dnat stream)) => bool

Pfork(f)   =
      ((! t . f’cpair’UU’t = cpair’UU’UU) &
       (! s . f’cpair’s’UU = cpair’UU’UU) &
       (! a b s1 s2 . (? h .
         Pfork(h) &
         ((a = dzero) -->
          f’cpair’scons’a’s1’scons’b’s2 =
                  cpair’scons’b’cfst’h’cpair’s1’s2
```

```
                'csnd'h'cpair's1's2) &
        ((a ~= dzero) -->
         f'cpair'scons'a's1'scons'b's2 =
                cpair'cfst'h'cpair's1's2
                'scons'b'csnd'h'cpair's1's2))))
```

Auch hier kann naürlich die Infixnotation verwendet werden:

```
Pfork(f)   =
      ((! t . f<UU,t> = <UU,UU>) &
       (! s . f<s,UU> = <UU,UU>) &
       (! a b s1 s2 . (? h .
         Pfork(h) &
         ((a = dzero) -->
          f'<a&&s1,b&&s2> =
              <b&&cfst'h'<s1,s2>,csnd'h'<s1,s2>) &
         ((a ~= dzero) -->
          f'<a&&s1,b&&s2> =
              <cfst'h'<s1,s2>,b&&csnd'h'<s1,s2>))))
```

6 Die FOCUS-Syntax

Mit der Definition der Basisoperatoren in Abschnitt 4 sind die notwendigen Voraussetzungen zur Beschreibung des Verhaltens von Basisagenten mittels einer an *HOLCF* und *Isabelle* orientierten Syntax gegeben. Diese damit geschaffene Beschreibungssprache läßt jedoch gerade hinsichtlich Strukturierung und Lesbarkeit noch vielfach zu wünschen übrig. Gerade die damit geschaffene und an den Beispielen demonstrierte Vielfalt von verschiedenen Darstellungsmöglichkeiten trägt durch ihre Uneinheitlichkeit nicht unwesentlich zu einer geringeren Akzeptanz der formalen Methodik FOCUS bei mathematisch weniger geschulten Anwendern bei. Durch die Festschreibung eines leicht verständlichen, festen syntaktischen Rahmens für Spezifikationen soll im Folgenden versucht werden, diese Hemmschwelle bei der Anwendung abzubauen.
Die vorgestellte Syntax soll zur Beschreibung von sogenannten Basisagenten verteilter Systeme dienen. Darunter sind solche Komponenten verteilter Systeme zu verstehen, die selbst nicht als verteiltes Systems beschrieben werden sollen. Daher stellt der hier beschriebene Anteil der FOCUS-Syntax keine der in [BDD+93] aufgeführten Operationen zur sequentiellen, parallelen oder rückkoppelnden Komposition von Systemkomponenten zur Verfügung. Ein Basisagent ist mittels Ein- und Ausgabekanälen mit seiner Umgebung verbunden. Über diese gerichteten Kanäle kann eine Komponente Nachrichten eines bestimmten Typs von der Umgebung empfangen bzw. an diese schicken. Eine Beschreibung eines Agenten setzt sich daher aus zwei Teilen zusammen, nämlich aus

- der Beschreibung der Verbindungsstruktur von Agent und Umgebung durch Angabe der Ein- und Ausgabekanäle sowie der Nachrichten, die über diese ausgetauscht werden können, sowie aus

- der Beschreibung des Verhaltens des Agenten durch Festlegung der Beziehung zwischen empfangenen und verschickten Nachrichten.

Diese Zweiteilung der Darstellung eines Agenten wird durch die FOCUS-Syntax unterstützt, und zwar durch eine entsprechende Aufspaltung der Spezifikation in

- die Beschreibung der syntaktischen Schnittstelle und
- die Verhaltensbeschreibung des Agenten.

Die beiden nachfolgenden Abschnitte werden sich mit diesen beiden Teilen der Spezifikation beschäftigen; insbesondere werden bei der Erläuterung der Verhaltensbeschreibung zwei unterschiedlich orientierte Spezifikationsstile eingeführt, nämlich

- der deskriptiv orientierte *relationale* Stil, sowie
- der implementierungsorientiertere *funktionale* Stil.

Zur Verdeutlichung wird jeder Abschnitt mit Beispielen der oben eingeführten `fork`-Funktion illustriert.

6.1 Schnittstellenbeschreibung

Wie oben besprochen, besteht die Spezifikation eines Basisagenten aus der Beschreibung der Schnittstelle und des Verhaltens des Agenten. Das Ziel der Schnittstellenbescheibung liegt darin, die Verbindungen zwischen Agent und Umgebung zu charakterisieren.

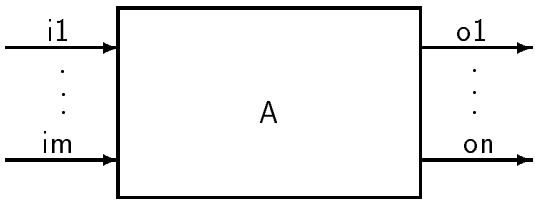

Abbildung 2: Die Schnittstelle eines Basisagenten

Entsprechend der obigen Darstellung besteht daher die Beschreibung der syntaktischen Schnittstelle eines Agenten aus

- einer Benennung des Agenten (Festlegung eines Identifikators),
- einer Beschreibung der Eingabekanäle und
- einer Beschreibung der Ausgabekanäle.

Die Beschreibung eines einzelnen Kanals wiederum setzt sich aus

- einem Kanalidentifikator, sowie

```
< agent-spec >    =   agent < ident >
                       input channel < channel-list >
                       output channel < channel-list >
                      is < body-spec >
                      end < ident >
< channel-list >  =  < empty >|< ident > : < ident > ,  < channel-list >
< empty >         =
```

Abbildung 3: Syntax der Schnittstellenbeschreibung

- dem Kanaltyp bzw. Nachrichtentyp

zusammen. Insgesamt ergibt sich damit für die Beschreibung eines Basisagenten die in Abbildung 3 als Ausschnitt einer BNF-Grammatik dargestellte Struktur.
Um die Beschreibung der syntaktischen Schnittstelle zu verdeutlichen, soll der in Abbildung 2 dargestellte Agent entsprechend der obigen Darstellung charakterisiert werden. Die zugehörige syntaktische Beschreibung findet sich in Abbildung 4. Spezifikationen, die in der oben beschriebenen Syntax formuliert werden, genügen dem in Abbildung 4 dargestellten Schema. Dabei ist `A` der Bezeichner des Agenten, `i1` bis `im` bzw. `o1` bis `on` sind die

```
agent A
 input  channel i1: I1, ... , im: Im
 output channel o1: O1, ... , on: On
is R(i1,...,im,o1,...,on)
end A
```

Abbildung 4: Beschreibung der syntaktischen Schnittstelle

Bezeichner der Ein- bzw. Ausgabekanäle, `I1` bis `Im` bzw. `O1` bis On sind die Bezeichner der Nachrichtentypen. `R(i1,...,im,o1,...,on)` steht hier für ein beliebiges Prädikat, das die Bezeichner für die Ein- und Ausgabekanäle als freie Variablen besitzt.
Auf die in der Beschreibung verwendeten Nachrichtentypen soll hier nicht eigens eingegangen werden. Die Spezifikation abstrakter Datentypen stellt ein gesondertes Gebiet mit eigenen Untersuchungen dar (siehe z.B. [BFG+93]), dessen Fragestellungen außerhalb der Spezifikation verteilter Systeme behandelt werden. Hier soll nur darauf verwiesen werden, daß durch die Abstützung der FOCUS-Syntax auf einen allgemeinen Theorembeweiser die Anbindung abstrakter Datentypen unterstützt wird. Aktuelle Arbeiten zu diesem Thema finden sich zum Beispiel in [vO95] oder [Slo95].
Insgesamt steht mit der hier vorgestellten Darstellung von Basisagenten eine leicht verständliche Beschreibung der syntaktischen Schnittstelle von Systemkomponenten zur Verfügung. Insbesondere kann so eine graphische Darstellung der Schnittstelle bzw. der Struktur direkt in die Syntax überführt werden. Ziel der Festlegung der FOCUS-Syntax ist jedoch letztendlich auch die Beweisunterstützung von FOCUS durch die auf HOLCF basierende

Focus-Theorie mittels des Theorembeweisers *Isabelle*. Dazu ist eine Umsetzung der syntaktischen Schnittstellenbeschreibung in eine entsprechende *Isabelle*-Formalisierung notwendig. Damit diese Umsetzung in einer werkzeuggestützten Systementwicklung sinnvoll eingesetzt werden kann, sollte sie darüberhinaus schematisierbar sein, um später automatisch zu erfolgen.
Da in FOCUS Agenten jeweils durch Mengen stromverarbeitender Funktionen dargestellt werden, muß die Umsetzung der Schnittstellenbeschreibung darin bestehen, einen Rahmen zur Charakterisierung von stromverarbeitenden Funktionen auf der Basis der Verhaltensbeschreibung zu generieren. Wegen der Verwendung eines höherstufigen Prädikatenkalküls wird dies hier durch ein Prädikat über stromverarbeitenden Funktionen dargestellt.
Für die in Abbildung 4 angeführte Schnittstellenspezifikation lautet die entsprechende Isabelle-Notation

```
is_A::   ((I1 stream * ... * Im stream) -> (O1 stream * ... * On stream))
                      => Bool

is_A(A) =
  ! i1 ... im o1 ... on .
     (A'<i1,...,im> = <o1,...,on> --> R(i1,...,im,o1,...,on))
```

Entsprechend des Typenkonzepts von Isabelle besteht die Formalisierung aus zwei Teilen, nämlich der Typisierung des charakterisierenden Prädikats sowie der zugehörigen Axiomatisierung. Durch die Beschreibung der syntaktischen Schnittstelle wird also auf der formalen Ebene ein Prädikat bereitgestellt, das Funktionen mit den entsprechenden Ein- und Ausgabekanälen charakterisiert. Diese Funktionen bilden dabei entsprechend der in FOCUS verwendeten Modellierung Tupel von Strömen über den Typen der Eingabe - im Beispiel `(I1 stream * ... * Im stream)` - auf Tupel von Strömen über den Typen der Ausgabekanäle - im Beispiel `(O1 stream * ... * On stream)` - ab.
Anschließend wird die Bedeutung des Prädikats " `is_A_def`" durch das entsprechende Axiom festgelegt. Dieses Prädikatenschema schreibt dabei lediglich fest, daß jede stromverarbeitende Funktion, die eine Instanz des Agenten `A` darstellt, die im Rumpf der Agentenbeschreibung festgelegten Eigenschaften erfüllen muß. Auf diese Beschreibung der semantischen Eigenschaften des Agenten soll im Folgenden näher eingegangen werden. Hier entspricht `R` wieder dem Rumpf der Agentenspezifikation.

6.2 Verhaltensbeschreibung

Wie oben bereits besprochen, besteht die Spezifikation eines Agenten neben der Charakterisierung der syntaktischen Schnittstelle aus der Beschreibung der semantischen Eigenschaften, also der Beziehung zwischen Ein- und Ausgaben des Agenten. Im obigen Abschnitt wurde auf die Verhaltensbeschreibung eines Agenten nicht weiter Bezug genommen. Zur Vervollständigung der Agentenbeschreibung soll daher im Folgenden besprochen werden, wie dessen semantische Charakterisierung im Rahmen der Syntax festgelegt wird.
Die Trennung der hier vorgestellten Notation zwischen syntaktischer und semantischer Beschreibung eines Agenten erlaubt bei einer einfachen Aufschreibung der syntaktischen

Schnittstelle eines Agenten die Verwendung der vollen Ausdrucksmöglichkeit der Focus-Theorie zur Charakterisierung der semantischen Eigenschaften. Diese erlaubt es, ganz unterschiedliche Spezifikationsstile zur Beschreibung des Agentenverhaltens zu verwenden. Um dies zu verdeutlichen, sollen hier zwei Stile vom jeweils entgegengesetzten Ende des Spektrums vorgestellt werden, nämlich

- der sehr abstrakte, eher deskriptiv orientierte, *relationale* Stil, sowie
- der konkretere, eher an Implementierungen orientierte, *funktionale* Stil.

Beide Stile sollen dabei jeweils anhand des oben eingeführten Beispiels verdeutlicht werden.

6.2.1 Relationale Beschreibung

Wie zu Beginn des Abschnitts erläutert, stellt der syntaktische Rahmen im wesentlichen lediglich die Typisierungsinformation des charakterisierenden Parädikats sowie ein Prädikatenschema zur Axiomatisierung desselben bereit. Dies läßt dem Anwender wesentlichen Freiraum zur Spezifikation des Agentenrumpfes. So ist es zum Beispiel möglich, eine besonders abstrakte Technik zur Verhaltenbeschreibung einzusetzen, nämlich die Technik der *relationalen Beschreibung*. Hierbei wird das Verhalten des Agenten lediglich durch die Betrachtung der jeweiligen Ein- und Ausgaben auf den entsprechenden Kanälen beschrieben. Daher stellt diese Form der Verhaltensbeschreibung ein Prädikat mit den Kanalbezeichnern als freien Variablen dar. Da hier nicht unmittelbar das Verhalten einer Funktion beschrieben wird, sondern lediglich die Beziehung (Relation) zwischen den auf den Kanälen übertragenen Nachrichten, wird diese Vorgehensweise als "relationaler Stil" bezeichnet.

6.2.2 Funktionale Beschreibung

Im Gegensatz zur relationalen Verhaltensbeschreibung verwendet die *funktionale* Beschreibung neben den Bezeichnern für die Kanäle auch den Bezeichner für den Agenten selbst, um dessen Verhalten zu beschreiben. Damit ist zusätzlich möglich, das Agentenverhalten induktiv zu notieren. So können Agenten in einer Weise beschrieben werden, die bereits eine konkrete Implementierung nahe legen. Dies schließt unter anderem mit ein, daß die Verhaltenbeschreibung in einer konstruktiven Form vorliegt, also eine direkte Implementierung darstellt.

6.2.3 Ein Beispiel

Anhand des in Abschnitt 5 eingeführten Beispiels soll die Verwendung der Syntax zur Definition von Basisagenten in relationaler und funktionaler Ausprägung demonstriert werden. Eine mögliche relationale Agentenbeschreibung könnte z.B. in der Form

```
agent fork (* Syntaktische Schnittstelle *}
 input  channel s: dnat t: dnat
 output channel x: dnat y: dnat
is (* Relationale Beschreibung des Verhaltens *)
  Fork(s,t,x,y) where
```

```
    Fork(s,t,x,y) = (((s = UU | t = UU) & (x = UU & y = UU)) |
                     (EX a b ss tt xx yy . (
                        Fork(ss,tt,xx,yy) & (
                           (a = dzero & s = a&&ss & t = b&&tt &
                                        x = b&&xx & y = yy) |
                           (a ~= dzero & s = a&&ss & t = b&&tt &
                                         x = xx & y = b&&yy)))))
end fork
```

notiert werden. Eine konstruktive funktionale Beschreibung des `fork`-Agenten könnte hingegen folgende Gestalt haben:

```
agent fork (* Syntaktische Schnittstelle *}
 input  channel s: dnat t: dnat
 output channel x: dnat y: dnat
is (* Konstruktive Beschreibung des Verhaltens *)
  <x,y> = fix'(LAM f. LAM s t.
              if(ft'x === dzero,
                 <shd't & cfst'fork'<stl's,stl't>,
                          csnd'fork'<stl's,stl't>>,
                 <        cfst'fork'<stl's,stl't>,
                  shd't & csnd'fork'<stl's,stl't>>))'<s,t>
end fork
```

Für die funktionale Beschreibung muß – wie im Beispiel ersichtlich – mittels der Fixpunktkonstruktion eine Anpassung an den relationalen Stil erfolgen. Um diese explizite Konstruktion zu vermeiden, wäre eine syntaktische Unterscheidung der beiden Stilarten – wie zum Beispiel durch die Schlüsselwörter `relation` und `function` – sinnvoll. Die Umsetzung der funktionalen Beschreibung in HOLCF liefert dann einen Rahmen, der diese Fixpunktkonstruktion enthält. Damit muß in der Spezifikation nur der eigentliche Rumpf der funktionalen Agentenbeschreibung angegeben werden.

7 Agentennetzwerke

In den vorangegangenen Kapiteln wurden die Grundlagen zur Aufschreibung von funktionalen Spezifikationen in *Isabelle*/HOLCF-Syntax gegeben und eine einfache Syntax zur Definition von Basisagenten festgelegt. Damit ist es nun möglich, die "kleinsten" Bausteine der mit FOCUS beschreibbaren Systeme in einem syntaktisch fest vorgegebenen Rahmen zu spezifizieren.
Um jedoch die für die Methodik FOCUS wesentlichen *verteilten* Systeme angeben zu können, ist es ebenso notwendig, auch hierfür einen syntaktischen Rahmen anzugeben. Wie schon in Kapitel 2.1 kurz erläutert, werden verteilte Systeme in FOCUS als Netzwerke von miteinander kommunizierenden Agenten modelliert. Die einzelnen Agenten innerhalb eines Netzwerkes sind über Kanäle miteinander verbunden; sie können unabhängig voneinander arbeiten (*parallel*), direkt hintereinandergeschaltet sein (*sequentiell*) oder auch eigene Ausgaben erneut wiederverarbeiten (*Rückkopplung*). In der bereits zu Beginn des Berichts

genannten Modellierung mittels stromverarbeitender Funktionen werden diese Kompositionsarten direkt über explizit definierte Operatoren angegeben. Wesentlich ist jedoch auch bei dieser Formalisierung die korrekte Zusammenschaltung der einzelnen Agenten über ihre Verbindungskanäle.
Als Mittel zur Veranschaulichung hat sich die z.B. in [BDD+93] verwendete graphische Darstellung bewährt. Die einzelnen Agenten werden durch Kästen, die mit einem Bezeichner für den Agenten versehen sind, angegeben. Die Kanäle werden durch gerichtete und mit den Bezeichnern der Kanäle markierte Kanten zwischen diesen Kästen bzw. zwischen dem Netzwerk und seiner Umgebung veranschaulicht. Ausgehende Kanten gelten somit für den durch den jeweiligen Kasten repräsentierten Agenten als Ausgabe- und eingehende Kanten dementsprechend als Eingabekanäle. Kanten für die explizit kein Agent als Ausgangs- bzw. Zielpunkt angegeben ist, definieren für das entsprechende Netzwerk die Schnittstelle des gesamten Netzwerks zu seiner Umgebung. Mit Hilfe dieser graphischen Darstellung lassen sich so die Schnittstellen der einzelnen Agenten und die Verbindungsstruktur des gesamten Netzwerks direkt ablesen.
Ein weiterer wesentlicher Punkt im Rahmen der so beschriebenen FOCUS-Sichtweise von verteilten Systemen besteht darin, daß die, wie oben beschrieben als Boxen dargestellten, Agenten nicht notwendigerweise Basisagenten sein müssen. In Entwicklungsschritten hin zu einer detaillierteren und filigraneren Beschreibung der Systeme ist es in FOCUS vorgesehen, erst in mehreren Abstraktionsstufen den endgültigen strukturellen Aufbau eines Systems zu entwickeln. Das geforderte Verhalten eines Agenten ergibt sich dann aus dem korrekten Zusammenschalten der Teilkomponenten bei Netzwerken bzw. aus dem Prädikat, das die Beziehung zwischen empfangenen und verschickten Nachrichten direkt festlegt, bei den Basisagenten (wie im vorhergehenden Kapitel beschrieben).
Die hier definierte Syntax zur Spezifikation von FOCUS-Netzwerken soll möglichst eng an diese Vorgehens- und Sichtweise angelehnt werden. Insbesondere soll es möglich sein, ein durch die oben beschriebene graphische Darstellung veranschaulichtes Agentennetzwerk möglichst schematisch in eine syntaktische Beschreibung umzusetzen. Zur Definition der Syntax werden die BNF-Form und die Umsetzung in *Isabelle*/HOLCF angegeben und die Verwendung der Syntax an einem Beispiel veranschaulicht.

7.1 Netzwerksyntax

Ein Netzwerk von Agenten stellt in FOCUS dank der kompositionalen Semantik lediglich eine besondere Form eines Agenten dar. Darunter ist zu verstehen, daß in FOCUS auf semantischer Ebene nicht unterschieden werden muß, ob ein Agent als Basisagent oder als Agentennetzwerk definiert ist. Diese einheitliche Sichtweise spiegelt sich auch in der hier definierten Beschreibungssprache für Agentennetzwerke wider.
In [Ded92] findet sich ein erster Ansatz zur Definition einer Beschreibungssprache für Agenten und Agentennetzwerke, der jedoch eher konstruktiv ausgelegt ist. Die hier definierte Syntax der Netzwerkbeschreibungen ist in Teilen an diesen Ansatz angelehnt.
Wie auch bereits die Beschreibung von Basisagenten, besteht die Beschreibung eines Agentennetzwerks infolge der oben erwähnten einheitlichen Sichtweise aus zwei Teilen, nämlich

- der Beschreibung der Verbindungsstruktur zwischen der Schnittstelle des Agentennetzwerks und der Umgebung durch Angabe der Ein- und Ausgabekanäle sowie der

Nachrichten, die über diese ausgetauscht werden können, sowie

- der Beschreibung des Verhaltens des Agentennetzwerk durch Festlegung der Beziehung zwischen empfangenen und verschickten Nachrichten an seiner Schnittstelle.

Bei Basisagenten wird die Verhaltensbeschreibung durch relational deskriptiven oder mehr funktional konstruktiven Prädikate unter Verwendung der FOCUS-Basisoperatoren angegeben. Da sich das Verhalten von Agentennetzwerken aus dem Verhalten der einzelnen Agenten sowie der Netzwerkstruktur ableiten läßt, werden nun Agentennetzwerke durch die Angabe

- der Agenten, auf die sich die Beschreibung des Agentennetzwerkes abstützt, sowie
- der Verbindungsstruktur dieser Agenten untereinander und zur Schnittstelle der Agentennetzwerkes

beschrieben. Damit besteht die Beschreibung eines Agentennetzwerkes aus

- der Schnittstellenbeschreibung, sowie
- der Netzwerkstrukturbeschreibung.

In bisherigen Arbeiten zu FOCUS wurden mehrere unterschiedliche Möglichkeiten zur formalen Darstellung von Netzwerkstrukturen vorgestellt. Ein wesentliches Ziel des hier vorgestellten Ansatzes besteht darin, eine möglichst intuitive und leicht verständliche Beschreibungssprache zu definieren. Zur Netzwerkbeschreibung wurde daher die folgende einfach aufgebaute Darstellungsform gewählt: die Beschreibung setzt sich aus einer Menge von Agenten/Kanal-Beschreibungen zusammen, die wiederum als Gleichungen der Form

$$< \text{Ausgabekanäle} >=< \text{Agent} >< \text{Eingabekanäle} >$$

geschrieben werden. Damit können unterschiedlichste Verbindungsstrukturen auf einheitliche Art und Weise dargestellt werden.
Insgesamt ergibt sich für die Beschreibung eines Netzwerks von Agenten die Struktur, wie sie in Abbildung 5 als Ausschnitt einer BNF-Grammatik dargestellt ist.

< network-body-spec >	=	`network` < agent-list > `end network`
< agent-list >	=	< agent > | < agent > ; < agent-list >
< agent >	=	< output-channels >=< ident >< input-channels >
< output-channels >	=	< channel-set >
< input-channels >	=	< channel-set >
< channel-set >	=	`<` < ident-list > `>`
< ident-list >	=	< empty > | < ident > , < ident-list >

Abbildung 5: Syntax der Netzwerkbeschreibung

Zusätzlich zu diesem syntaktischen Rahmen muß für die Netzwerkbeschreibungen die Verträglichkeitsanforderung für die Ein- und Ausgabekanäle der Agenten des Netzwerkes gelten. Diese Anforderung stellt die korrekte Koppelung von Ein- und Ausgabekanälen sicher. Sie besteht aus zwei Einzelanforderungen:

Ausgabeverträglichkeit: Jeder Ausgabekanal des gesamten Netzwerks ist genau einem Agenten zugeordnet.

Schnittstellenverträglichkeit: Alle Eingabekanäle der Schnittstellenbeschreibung sind auch ausschließlich Eingabekanäle der einzelnen Agenten; alle Ausgabekanäle der Schnittstellenbeschreibung sind auch ausschließlich Ausgabekanäle der einzelnen Agenten.

Auf syntaktischer Ebene kann die erste Anforderung nachgeprüft werden, indem sichergestellt wird, daß jeder Kanalidentifikator höchstens einmal in der Menge der Ausgabekanäle, also auf der linken Seite der Agenten/Kanalbeschreibungen vorkommt. Entsprechend muß für die zweite Anforderung gelten, daß jeder Eingabekanal der Schnittstellenbeschreibung nur auf der rechten Seite der Agenten/Kanalbeschreibungen vorkommt sowie jeder Ausgabekanal nur auf der linken.

7.2 Umsetzung in HOLCF

Die Beschreibungen eines Agentennetzwerks besteht, wie in 7.1 beschrieben, aus der Darstellung der syntaktischen Schnittstelle des Netzwerks sowie der Verbindungsstruktur der Agenten. Dementsprechend müssen für die Umsetzung eines Netzwerks Umsetzungen für die Schnittstellenbeschreibung und die Verbindungsstruktur angegeben werden.
Ähnlich wie im Fall eines Basisagenten (vgl. Abschnitt 6) wird hierfür einerseits die Typisierung für das charakterisierende Prädikat festgelegt, und andererseits die dazugehörige Axiomatisierung. Während dabei die Typisierung des charakterisierenden Prädikats des Netzwerks entsprechend der eines Basisagenten aufgebaut ist, ist jedoch die Axiomatisierung unterschiedlich.
Entsprechend der BNF-Grammatiken aus den Abbildungen 3 und 5 entspricht die vollständige Beschreibung eines Agentennetzwerks einschließlich Schnittstellenbeschreibung und Vernetzungsstruktur dem Schema:

```
agent A
 input  channel i1: I1, ... ,im: Im
 output channel o1: O1, ... ,on: On
is network
<oc11,...,oc1r> = A1<ic11,...,ic1s>;
                ...
<ock1,...,ockr> = Ak<ick1,...,icks>
end network
end A
```

wobei

- `A1`,...,`Ak` Bezeichner von Agenten sind, und

- `ocl1`,...,`oclr` Platzhalter für Bezeichner von Ausgabekanälen des gesamten Agentennetzwerks oder von internen Kanälen, und
- `icl1`,...,`icls` Platzhalter für Bezeichner von Eingabekanälen des gesamten Agentennetzwerks oder von internen Kanälen.

Als interne Kanäle werden dabei solche Kanäle bezeichnet, die nicht in der Schnittstellenbeschreibung auftreten, also die Komponenten des Netzwerks nur untereinander, und nicht mit der Umgebung des Netzwerks verbinden. Im Folgenden werden `h1`,...,`hl` als Bezeichner der internen Kanäle verwendet.

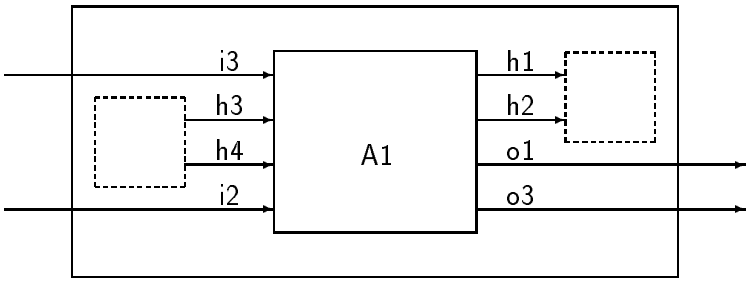

Abbildung 6: Komponente eines Netzwerks

In der konkreten Spezifikation eines Netzwerks werden also die Platzhalter des Schemas ersetzt durch die konkreten Bezeichner für die externen und internen Kanäle. Beispielsweise könnte das Gleichungsschema

```
<oc11,...,oc1r> = A1<ic11,...,ic1s>
```

im Fall eines wie in Abbildung 6 dargestellten Netzwerks die konkrete Gestalt

```
<h1,h2,o1,o3>   = A1<i3,h3,h4,i2>
```

annehmen.
Die HOLCF-Umsetzung der Axiomatisierung des Agentennetzwerks besteht wiederum aus zwei Teilen, nämlich

- der Charakterisierung des Agentennetzwerks (`is_A`) in Abhängigkeit von der Beschreibung der Agenten und der Netzwerkstruktur des Netzwerks, sowie
- der Umsetzung der Netzwerkbeschreibung d.h. der Beschreibung der Netzwerkstruktur (`is_net_A`).

Damit ergibt sich für die Schnittstellenbeschreibung der obigen Netzwerke die folgende HOLCF-Darstellung:

```
is_A::  ((I1 stream * ... * Im stream) -> (O1 stream * ... * On stream)
                    => Bool
is_A(A) =
```

```
  ? A1 ... Ak .
     is_A1(A1) & ... & is_Ak(Ak) &
     ! i1 ... im o1 ... on .
        (A'<i1,...,im> = <o1,...,on) -->
         is_net_A(A1,...,Ak,i1,...,im,o1,...,on))
```

Die Umsetzung der Netzwerkbeschreibung in die entsprechende HOLCF-Darstellung ordnet dem Netzwerk ein Prädikat `is_net_A` zur Charakterisierung von Ein- und Ausgabepaaren in Abhängigkeit von entsprechenden Vertretern der einzelnen Agenten zu, das der Verhaltensbeschreibung bei Basisagenten entspricht. Zusammen mit der HOLCF-Darstellung für die Schnittstellenbeschreibung des Netzwerks ergibt sich die vollständige Charakterisierung des Agentennetzwerks.
Neben der Generierung des entsprechenden Typs der Netzwerkbeschreibung

```
is_net_A: [(I11 stream * ... * Im1 stream) ->
                    (O11 stream * ...* On1 stream),
           ...
           (I1k stream * ... * Imk stream) ->
                    (O1k stream * ...* Onk stream),
           I1 stream,...,Im stream,O1 stream,...,On stream]
                      ==> Bool
```

- wobei `(I1j stream * ... * Imj stream) -> (O1j stream * ... * Onj stream)` dem Typ des i-ten Agenten des Netzwerks entspricht - wird der Beschreibung folgende Bedeutung zugeordnet:

- Für jede Belegung der Ein- und Ausgabekanäle des gesamten Agentennetzwerks gibt es eine Belegung der internen Kanäle, die das gesamte Gleichungssystem erfüllt.
- Jede Belegung der internen Kanäle ist dabei minimal; dies bedeutet, daß keine andere echt kleinere Belegung der internen Kanäle das Gleichungssystem erfüllt.

Die Bedeutung dieser Zuordnung wird in [BDD+92] beschrieben.
Für das obige Agentennetzwerk lautet die entsprechende Umsetzung in HOLCF damit:

```
is_net_A(A1,...,Ak,i1,...,im,o1,...,on) =
     ? h1 ... hl .
         G(A1,...,Ak,i1,...,im,o1,...,on,h1,...,hl) &
         ! oo1 ... oon hh1 ... hhl .
             G(A1,...,Ak,i1,...,im,oo1,...,oon,hh1,...,hhl) --->
                 <h1,...,hl> << <hh1,...,hhl>
```

Dabei repräsentiert `G(A1,...,Ak,i1,...,im,o1,...,on,h1,...,hl)` die Umsetzung für das obige Gleichungssystem zur Beschreibung der Netzwerkstruktur. Dies wird direkt aus dem Netzwerkgleichungssystem durch Konjunktion der einzelnen Gleichungen gewonnen.

```
<oc11,...,oc1r> = A1<ic11,...,ic1s> &
                ...
<ock1,...,ockr> = Ak<ick1,...,icks>
```

Hierbei werden mehrfach auftreten Instanzen eines Agenten durch jeweils unterschiedliche Bezeichner ersetzt, um unabhängige Instantiierungen zu erlauben. In Abschnitt 7.3 wird dies am Agenten Medium deutlich gemacht.

7.3 Beispiel

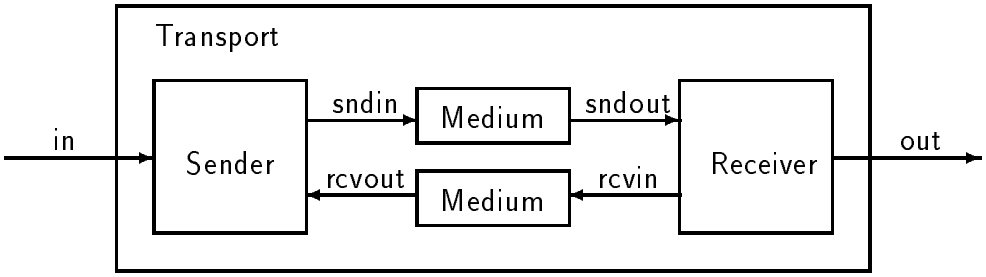

Abbildung 7: Struktur des Transporters

Im folgenden Abschnitt soll anhand des Beispiels des "Alternating Bit Protocol" (siehe z.B. [DW92]) die Verwendung der Beschreibung von Netzwerkagenten deutlich gemacht werden. Dazu wird die in Abbildung 7 beschriebene Struktur des Agenten "Transport" definiert. Dieser Agent besteht aus

- Eingabekanal in,
- Ausgabekanal out,
- und den Unterkomponenten Sender, Receiver sowie zwei Agenten der Art Medium.

Dabei sollen auf allen Kanälen nur Nachrichten vom Typ `Message` übertragen werden. Ohne genauere Kenntnis der Unterkomponenten kann damit die in Abbildung 7 beschriebene Netzwerkstruktur sofort in die in Abbildung 8 gezeigte FOCUS-Darstellung umgesetzt werden. Für die vollständige Beschreibung des Netzwerks müssen nun die entsprechenden FOCUS-Darstellungen der Basiskomponenten Sender, Receiver, sowie Medium angegeben werden, worauf hier verzichtet werden soll.

```
agent Transport
 input  channel in: Message
 output channel out: Message
is network
   <sndin>     = Sender<in,rcvout>;
   <sndout>    = Medium<sndin>;
   <rcvout>    = Medium<rcvin>;
   <out,rcvin> = Receiver<sndout>
   end network
end Transport
```

Abbildung 8: Beschreibung der syntaktischen Schnittstelle

```
is_Transport::   ((Message stream) -> (Message stream))
                    => Bool

is_Transport(Transport) =
  ? Sender Medium1 Medium2 Receiver .
     is_Sender(Sender) & is_Medium(Medium1) &
     is_medium(Medium2) & is_Receiver(Receiver) &
     ! in out .
        (Transport'in = out -->
           is_net_Transport(Sender,Medium1,Medium2,Receiver,in,out))

is_net_Transport: [(Message stream * Message stream -> Message stream),
                   (Message stream -> Message stream),
                   (Message stream -> Message stream),
                   (Message stream -> Message stream * Message stream),
                   Message stream,Message stream]
                      => Bool

is_net_Transport(Sender,Medium1,Medium2,Receiver,in,out) =
     ? sndin rcvout sndout rcvin .
         G(Sender,Medium1,Medium2,Receiver,in,out,
              sndin,rcvout,sndout,rcvin) &
         ! oout hsndin hrcvout hsndout hrcvin .
              G(Sender,Medium1,Medium2,Receiver,in,oout,
                 hsndin,hrcvout,hsndout,hrcvin) -->
                  <sndin,rcvout,sndout,rcvin> <<
                      <hsndin,hrcvout,hsndout,hrcvin>
```

Abbildung 9: HOLCF-Darstellung des “Alternating Bit”

8 Zusammenfassung und Ausblick

Mit dem vorliegenden Bericht wird eine möglichst einfache und leicht handhabbare technische Beschreibungssprache für FOCUS festgelegt. Mit diesen syntaktischen Festlegungen ist es nunmehr möglich, basierend auf einem einheitlichen semantischen Modell – den stromverarbeitenden Funktionen – sowohl Basisagenten als auch verteilte Systeme als Agentennetzwerke in einer syntaktischen Kernsprache zu spezifizieren.
Durch die Anbindung an HOLCF und die Definition der HOLCF-Theorie `Focus` im Rahmen einer konservativen und konstruktiven Erweiterung von HOLCF sind wir nunmehr in der Lage die Möglichkeiten, die mit dieser Logik zur Verfügung stehen, auch für FOCUS-Spezifikationen zu nutzen. Insbesondere durch die Festlegung der schematischen Umsetzung der "neuen" FOCUS-Spezifikationen in *Isabelle*-Syntax ist es möglich, formale Beweise mit Hilfe des Beweisunterstützungswerkzeuges durchzuführen.
Die mit den syntaktischen Festlegungen zu erstellenden Schnittstellenbeschreibungen von Agenten erscheinen uns vor allem deshalb leicht handhabbar und verständlich, weil sie eine direkte Umsetzung der für FOCUS-Spezifikationen oft zur Veranschaulichung benutzten graphischen Darstellung der Agenten und ihrer Ein-/Ausgabekanäle darstellen. Diese schematische Umsetzung ist nicht nur für die Spezifikation der Schnittstellen der Basisagenten gegeben, sondern setzt sich auch in der syntaktischen Festschreibung von Netzwerkspezifikationen fort. Hier wird die durch die graphische Darstellung vorgegebene Struktur des Netzwerks direkt in eine Menge von Gleichungen umgesetzt.
Zur Erstellung der Verhaltensprädikate der Basisagenten im Rumpf einer Spezifikation sind mit diesem Bericht die vorgegebenen Operatoren in *Isabelle*/HOLCF festgelegt, und dem Spezifikateur stehen zunächst zwei unterschiedliche Stile zur Aufschreibung der Verhaltensspezifikation zur Verfügung – der deskriptive relationale Stil und der eher implementierungsnahe funktionale Stil. Im Rahmen dieser Möglichkeiten sollte es für FOCUS-Kenner relativ leicht möglich sein, Verhaltensspezifikationen in der neuen Syntax zu erstellen. Dies wird sicher auch dadurch erleichtert, daß die *Isabelle*-Syntax in einer erst vor kurzem abgeschlossenen Arbeit ([Reg95]) um die gängigen mathematischen Symbole erweitert wurde. Für FOCUS-Neulinge bzw. Anwender, die mit der Erstellung von FOCUS-Spezifikationen noch nicht vertraut sind, bleibt die Erstellung der Verhaltensprädikate auch weiterhin eine nichttriviale Aufgabe. Hierfür bleibt Kreativität und Erfahrung erforderlich, und diese Aufgabe läßt sich nicht generell schematisieren. Der Bericht liefert jedoch auch hierfür Hilfestellung, da zumindest der formale Rahmen für die Aufschreibung festgelegt ist.
Damit ergibt sich, daß funktional spezifizierte FOCUS-Systeme gemäß dem in Kapitel 2.1 beschriebenen Rahmen nunmehr in jeder Entwicklungsstufe entsprechend dem gewünschten Abstraktionsgrad als Agent oder Agenten-Netzwerk mit den syntaktischen Festlegungen von ANDL spezifiziert werden können. Insbesondere bedeutet dies, daß es selbstverständlich nicht erforderlich ist, ein verteiltes System von Anfang an in seinem vollen Detaillierungsgrad zu kennen, damit es in ANDL als Agenten-Netzwerk mit den einzelnen Basisagenten vollständig spezifiziert werden kann. Mit dem vorliegenden Bericht war lediglich beabsichtigt, zunächst die grundlegende Konzeption von FOCUS-Systemen syntaktisch vorzugeben. Daraus ergibt sich, daß es für die Top-Down-Entwicklung von Systemen mit den für FOCUS charakteristischen Verfeinerungskonzepten nötig ist, die entsprechenden Begriffe ebenfalls in den logischen Rahmen einzubetten.

Die bisher geleisteten und in diesem Bericht vorgestellten Arbeiten bieten jedoch auch ein weites Feld für weitere Überlegungen und Erweiterungen der FOCUS-Syntax. Für eine vollständige Umsetzung in HOLCF ist es notwendig, auch die Typisierung der Nachrichten und Daten explizit festzuschreiben. Hier wird [vO95] wertvolle Hilfestellung leisten, da hiermit die Datentyp-Definition in einer erweiterten Version von HOLCF als Paket zur Verfügung steht. Auch die Arbeiten zur Erstellung eines Parsers für die Umsetzung der FOCUS-Spezifikationen in eine Form mit der die Beweisunterstützung durch *Isabelle* genutzt werden kann, sind weit fortgeschritten, so daß der Parser bald zur Verfügung stehen wird. Da FOCUS auch gezeitete Spezifikationen unterstützt und in einem theoretisch ausgereiften Konzept zur Verfügung stellt, muß die Syntax um diese Variante erweitert werden. Weitere Arbeiten werden sich damit beschäftigen, zur Erstellung der Verhaltensspezifikationen im Rahmen der syntaktischen Festlegungen auch zunehmend pragmatische Techniken wie Tabellen, Diagramme und graphische Darstellungen direkt für die Spezifikation des Verhaltens im Rumpf der Basisagenten zuzulassen. Auch weitere Spezifikationsstile wie Assumption/Commitment oder zustandsorientierte Beschreibungen sollen hier ermöglicht werden. Da die Akzeptanz und Handhabbarkeit der Syntax erprobt werden muß, wird es erforderlich sein, Beispiele und Fallstudien zu bearbeiten. Auf diesem Weg könnte es sich auch ergeben, daß die Spezifikationen häufig verwendeter Agenten als Paket für umfangreichere Arbeiten standardmäßig zur Verfügung stehen.

8.1 Erweiterung der Syntax

Über die bisher vorgestellte Syntax zur Beschreibung von Agenten hinaus ist eine Parametrisierung der Agenten denkbar. Darunter ist zu verstehen, daß einem Agenten neben den Ein- und Ausgabekanälen Variablen zugeordnet werden, deren Belegung das Verhalten des Agenten beeinflußt. Abbildung 10 zeigt eine solche Variante für den Fall eines Basisagenten.

```
agent A input channel  i1 : I1, ... , im : Im
        output channel o 1: O1, ... , on : On
        par p1 : P1, ... , pl : Pl
is R(p1,...,pl,i1,...,im,o1,...,on)
end A
```

Abbildung 10: Parametrisierter Basisagent

Entsprechend ist auch die Erweiterung von Agentennetzwerkbeschreibungen möglich. Eine Umsetzung in HOLCF ist dabei auf unterschiedliche Weisen denkbar, nämlich als Parametrisierung der entsprechenden stromverarbeitenden Funktionen oder als Parametrisierung des entsprechenden charakterisierenden Prädikats.
In beiden Fällen kann diese Parametrisierung auch zur Bildung von Agenten höherer Stufe eingesetzt werden; darunter sind solche Agenten zu verstehen, deren Parameter selbst stromverarbeitende Funktionen darstellen. Eine mögliche Anwendung ist hierbei zum Beispiel die Bildung von Funktionalen, und damit von Netzwerkoperationen; hierbei wäre unter anderem die n-fache Parallel- oder Hintereinanderschaltung eines Agenten zu erwähnen.

Eine solche Erweiterung muß jedoch sorgfältig in die bisherige Beschreibungssprache eingebettet werden, um die bisher sichergestellte Konservativität und Konstruktivität des Ansatzes sicherzustellen. Insbesondere sind dazu die Kosten und Nutzen einer solchen Erweiterung genau zu untersuchen.

Danksagung

Wir danken Franz Regensbuger für die wiederholte Überarbeitung der HOLCF-Anteile dieser Arbeit. Weiterhin danken wir Manfred Broy, Ursula Hinkel, Frank Leßke, Oscar Slotosch und Peter Paul Spies für ihre Kommentare zur Arbeit.

Literatur

[BDD+92] Manfred Broy, Frank Dederich, Claus Dendorfer, Max Fuchs, Thomas Gritzner und Rainer Weber. The Design of Distributed Systems - An Introduction to FOCUS. TUM-I 9202, Technische Universität München, Januar 1992.

[BDD+93] Manfred Broy, Frank Dederich, Claus Dendorfer, Max Fuchs, Thomas Gritzner und Rainer Weber. The Design of Distributed Systems - An Introduction to FOCUS. TUM-I 9202-2, Technische Universität München, Januar 1993. SFB-Bericht Nr.342/2-2/92 A.

[BFG+93] Manfred Broy, Christian Facchi, Radu Grosu, Rudi Hettler, Heinrich Hußmann, Dieter Nazareth, Franz Regensburger, Oscar Slotosch und Ketil Stølen. The Requirement and Design Specification Language SPECTRUM An Informal Introduction (V 1.0). TUM-I 9311, Technische Universität München, 1993.

[Bro93] Manfred Broy. Interaction Refinement The Easy Way. In M. Broy, Hrsg., *Program Design Calculi. Springer NATO ASI Series, Series F: Computer and System Sciences, Vol. 118*, 1993.

[Bro94] Manfred Broy. Specification and Refinement of a Buffer of Lenght One, 1994. Working material of Marktoberdorf Summer School 1994.

[Ded92] Frank Dederichs. *Transformation verteilter Systeme: Von applikativen zu prozeduralen Darstellungen.* Dissertation, Technische Universität München, 1992.

[DW92] Claus Dendorfer und Rainer Weber. Development and Implementation of a Communication Protocol - An Exercise in FOCUS. TUM-I 9205, Technische Universität München, Marz 1992.

[Pau94a] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, Jgg. 828 of *LNCS*. Springer-Verlag, 1994.

[Pau94b] Lawrence C. Paulson. *The Isabelle Reference Manual.* Computer Labarotary, University of Cambridge, 1994.

[Reg94] Franz Regensburger. *HOLCF: Eine konservative Erweiterung von HOL um LCF.* Dissertation, Technische Universität München, 1994.

[Reg95] Franz Regensburger. *The 8-bit Isabelle* HOLCF *Manual Pages.* Technische Universität München, 1995.

[Slo95] Oscar Slotosch. Implementing the Change of Data Structures with SPECTRUM in the Framework of KORSO Development Graphs. TUM-I 9511, Technische Universität München, 1995.

[Spi94] Katharina Spies. Protokoll zur SFB-A6-Besprechung am 28.10.1994. Unveröffentlichtes Manuskript, 1994.

[Stø94] Ketil Stølen. Preliminary Set of Operators. Unveröffentlichtes Manuskript, 1994.

[vO95] David von Oheim. Datentyp-Spezifikationen in HOLCF. Diplomarbeit, Technische Universität München, 1995.

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

342/1/90 A	Robert Gold, Walter Vogler: Quality Criteria for Partial Order Semantics of Place/Transition-Nets, Januar 1990
342/2/90 A	Reinhard Fößmeier: Die Rolle der Lastverteilung bei der numerischen Parallelprogrammierung, Februar 1990
342/3/90 A	Klaus-Jörn Lange, Peter Rossmanith: Two Results on Unambiguous Circuits, Februar 1990
342/4/90 A	Michael Griebel: Zur Lösung von Finite-Differenzen- und Finite-Element-Gleichungen mittels der Hierarchischen Transformations-Mehrgitter-Methode
342/5/90 A	Reinhold Letz, Johann Schumann, Stephan Bayerl, Wolfgang Bibel: SETHEO: A High-Performance Theorem Prover
342/6/90 A	Johann Schumann, Reinhold Letz: PARTHEO: A High Performance Parallel Theorem Prover
342/7/90 A	Johann Schumann, Norbert Trapp, Martin van der Koelen: SETHEO/PARTHEO Users Manual
342/8/90 A	Christian Suttner, Wolfgang Ertel: Using Connectionist Networks for Guiding the Search of a Theorem Prover
342/9/90 A	Hans-Jörg Beier, Thomas Bemmerl, Arndt Bode, Hubert Ertl, Olav Hansen, Josef Haunerdinger, Paul Hofstetter, Jaroslav Kremenek, Robert Lindhof, Thomas Ludwig, Peter Luksch, Thomas Treml: TOPSYS, Tools for Parallel Systems (Artikelsammlung)
342/10/90 A	Walter Vogler: Bisimulation and Action Refinement
342/11/90 A	Jörg Desel, Javier Esparza: Reachability in Reversible Free- Choice Systems
342/12/90 A	Rob van Glabbeek, Ursula Goltz: Equivalences and Refinement
342/13/90 A	Rob van Glabbeek: The Linear Time - Branching Time Spectrum
342/14/90 A	Johannes Bauer, Thomas Bemmerl, Thomas Treml: Leistungsanalyse von verteilten Beobachtungs- und Bewertungswerkzeugen
342/15/90 A	Peter Rossmanith: The Owner Concept for PRAMs
342/16/90 A	G. Böckle, S. Trosch: A Simulator for VLIW-Architectures
342/17/90 A	P. Slavkovsky, U. Rüde: Schnellere Berechnung klassischer Matrix-Multiplikationen

Reihe A

342/18/90 A Christoph Zenger: SPARSE GRIDS

342/19/90 A Michael Griebel, Michael Schneider, Christoph Zenger: A combination technique for the solution of sparse grid problems

342/20/90 A Michael Griebel: A Parallelizable and Vectorizable Multi- Level-Algorithm on Sparse Grids

342/21/90 A V. Diekert, E. Ochmanski, K. Reinhardt: On confluent semi-commutations-decidability and complexity results

342/22/90 A Manfred Broy, Claus Dendorfer: Functional Modelling of Operating System Structures by Timed Higher Order Stream Processing Functions

342/23/90 A Rob van Glabbeek, Ursula Goltz: A Deadlock-sensitive Congruence for Action Refinement

342/24/90 A Manfred Broy: On the Design and Verification of a Simple Distributed Spanning Tree Algorithm

342/25/90 A Thomas Bemmerl, Arndt Bode, Peter Braun, Olav Hansen, Peter Luksch, Roland Wismüller: TOPSYS - Tools for Parallel Systems (User's Overview and User's Manuals)

342/26/90 A Thomas Bemmerl, Arndt Bode, Thomas Ludwig, Stefan Tritscher: MMK - Multiprocessor Multitasking Kernel (User's Guide and User's Reference Manual)

342/27/90 A Wolfgang Ertel: Random Competition: A Simple, but Efficient Method for Parallelizing Inference Systems

342/28/90 A Rob van Glabbeek, Frits Vaandrager: Modular Specification of Process Algebras

342/29/90 A Rob van Glabbeek, Peter Weijland: Branching Time and Abstraction in Bisimulation Semantics

342/30/90 A Michael Griebel: Parallel Multigrid Methods on Sparse Grids

342/31/90 A Rolf Niedermeier, Peter Rossmanith: Unambiguous Simulations of Auxiliary Pushdown Automata and Circuits

342/32/90 A Inga Niepel, Peter Rossmanith: Uniform Circuits and Exclusive Read PRAMs

342/33/90 A Dr. Hermann Hellwagner: A Survey of Virtually Shared Memory Schemes

342/1/91 A Walter Vogler: Is Partial Order Semantics Necessary for Action Refinement?

342/2/91 A Manfred Broy, Frank Dederichs, Claus Dendorfer, Rainer Weber: Characterizing the Behaviour of Reactive Systems by Trace Sets

342/3/91 A Ulrich Furbach, Christian Suttner, Bertram Fronhöfer: Massively Parallel Inference Systems

Reihe A

342/4/91 A Rudolf Bayer: Non-deterministic Computing, Transactions and Recursive Atomicity

342/5/91 A Robert Gold: Dataflow semantics for Petri nets

342/6/91 A A. Heise; C. Dimitrovici: Transformation und Komposition von P/T-Netzen unter Erhaltung wesentlicher Eigenschaften

342/7/91 A Walter Vogler: Asynchronous Communication of Petri Nets and the Refinement of Transitions

342/8/91 A Walter Vogler: Generalized OM-Bisimulation

342/9/91 A Christoph Zenger, Klaus Hallatschek: Fouriertransformation auf dünnen Gittern mit hierarchischen Basen

342/10/91 A Erwin Loibl, Hans Obermaier, Markus Pawlowski: Towards Parallelism in a Relational Database System

342/11/91 A Michael Werner: Implementierung von Algorithmen zur Kompaktifizierung von Programmen für VLIW-Architekturen

342/12/91 A Reiner Müller: Implementierung von Algorithmen zur Optimierung von Schleifen mit Hilfe von Software-Pipelining Techniken

342/13/91 A Sally Baker, Hans-Jörg Beier, Thomas Bemmerl, Arndt Bode, Hubert Ertl, Udo Graf, Olav Hansen, Josef Haunerdinger, Paul Hofstetter, Rainer Knödlseder, Jaroslav Kremenek, Siegfried Langenbuch, Robert Lindhof, Thomas Ludwig, Peter Luksch, Roy Milner, Bernhard Ries, Thomas Treml: TOPSYS - Tools for Parallel Systems (Artikelsammlung); 2., erweiterte Auflage

342/14/91 A Michael Griebel: The combination technique for the sparse grid solution of PDE's on multiprocessor machines

342/15/91 A Thomas F. Gritzner, Manfred Broy: A Link Between Process Algebras and Abstract Relation Algebras?

342/16/91 A Thomas Bemmerl, Arndt Bode, Peter Braun, Olav Hansen, Thomas Treml, Roland Wismüller: The Design and Implementation of TOPSYS

342/17/91 A Ulrich Furbach: Answers for disjunctive logic programs

342/18/91 A Ulrich Furbach: Splitting as a source of parallelism in disjunctive logic programs

342/19/91 A Gerhard W. Zumbusch: Adaptive parallele Multilevel-Methoden zur Lösung elliptischer Randwertprobleme

342/20/91 A M. Jobmann, J. Schumann: Modelling and Performance Analysis of a Parallel Theorem Prover

342/21/91 A Hans-Joachim Bungartz: An Adaptive Poisson Solver Using Hierarchical Bases and Sparse Grids

342/22/91 A Wolfgang Ertel, Theodor Gemenis, Johann M. Ph. Schumann, Christian B. Suttner, Rainer Weber, Zongyan Qiu: Formalisms and Languages for Specifying Parallel Inference Systems

Reihe A

342/23/91 A Astrid Kiehn: Local and Global Causes

342/24/91 A Johann M.Ph. Schumann: Parallelization of Inference Systems by using an Abstract Machine

342/25/91 A Eike Jessen: Speedup Analysis by Hierarchical Load Decomposition

342/26/91 A Thomas F. Gritzner: A Simple Toy Example of a Distributed System: On the Design of a Connecting Switch

342/27/91 A Thomas Schnekenburger, Andreas Weininger, Michael Friedrich: Introduction to the Parallel and Distributed Programming Language ParMod-C

342/28/91 A Claus Dendorfer: Funktionale Modellierung eines Postsystems

342/29/91 A Michael Griebel: Multilevel algorithms considered as iterative methods on indefinite systems

342/30/91 A W. Reisig: Parallel Composition of Liveness

342/31/91 A Thomas Bemmerl, Christian Kasperbauer, Martin Mairandres, Bernhard Ries: Programming Tools for Distributed Multiprocessor Computing Environments

342/32/91 A Frank Leßke: On constructive specifications of abstract data types using temporal logic

342/1/92 A L. Kanal, C.B. Suttner (Editors): Informal Proceedings of the Workshop on Parallel Processing for AI

342/2/92 A Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: The Design of Distributed Systems - An Introduction to FOCUS

342/2-2/92 A Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: The Design of Distributed Systems - An Introduction to FOCUS - Revised Version (erschienen im Januar 1993)

342/3/92 A Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: Summary of Case Studies in FOCUS - a Design Method for Distributed Systems

342/4/92 A Claus Dendorfer, Rainer Weber: Development and Implementation of a Communication Protocol - An Exercise in FOCUS

342/5/92 A Michael Friedrich: Sprachmittel und Werkzeuge zur Unterstüt- zung paralleler und verteilter Programmierung

342/6/92 A Thomas F. Gritzner: The Action Graph Model as a Link between Abstract Relation Algebras and Process-Algebraic Specifications

342/7/92 A Sergei Gorlatch: Parallel Program Development for a Recursive Numerical Algorithm: a Case Study

342/8/92 A Henning Spruth, Georg Sigl, Frank Johannes: Parallel Algorithms for Slicing Based Final Placement

Reihe A

342/9/92 A Herbert Bauer, Christian Sporrer, Thomas Krodel: On Distributed Logic Simulation Using Time Warp

342/10/92 A H. Bungartz, M. Griebel, U. Rüde: Extrapolation, Combination and Sparse Grid Techniques for Elliptic Boundary Value Problems

342/11/92 A M. Griebel, W. Huber, U. Rüde, T. Störtkuhl: The Combination Technique for Parallel Sparse-Grid-Preconditioning and -Solution of PDEs on Multiprocessor Machines and Workstation Networks

342/12/92 A Rolf Niedermeier, Peter Rossmanith: Optimal Parallel Algorithms for Computing Recursively Defined Functions

342/13/92 A Rainer Weber: Eine Methodik für die formale Anforderungsspezifkation verteilter Systeme

342/14/92 A Michael Griebel: Grid– and point–oriented multilevel algorithms

342/15/92 A M. Griebel, C. Zenger, S. Zimmer: Improved multilevel algorithms for full and sparse grid problems

342/16/92 A J. Desel, D. Gomm, E. Kindler, B. Paech, R. Walter: Bausteine eines kompositionalen Beweiskalküls für netzmodellierte Systeme

342/17/92 A Frank Dederichs: Transformation verteilter Systeme: Von applikativen zu prozeduralen Darstellungen

342/18/92 A Andreas Listl, Markus Pawlowski: Parallel Cache Management of a RDBMS

342/19/92 A Erwin Loibl, Markus Pawlowski, Christian Roth: PART: A Parallel Relational Toolbox as Basis for the Optimization and Interpretation of Parallel Queries

342/20/92 A Jörg Desel, Wolfgang Reisig: The Synthesis Problem of Petri Nets

342/21/92 A Robert Balder, Christoph Zenger: The d-dimensional Helmholtz equation on sparse Grids

342/22/92 A Ilko Michler: Neuronale Netzwerk-Paradigmen zum Erlernen von Heuristiken

342/23/92 A Wolfgang Reisig: Elements of a Temporal Logic. Coping with Concurrency

342/24/92 A T. Störtkuhl, Chr. Zenger, S. Zimmer: An asymptotic solution for the singularity at the angular point of the lid driven cavity

342/25/92 A Ekkart Kindler: Invariants, Compositionality and Substitution

342/26/92 A Thomas Bonk, Ulrich Rüde: Performance Analysis and Optimization of Numerically Intensive Programs

342/1/93 A M. Griebel, V. Thurner: The Efficient Solution of Fluid Dynamics Problems by the Combination Technique

342/2/93 A Ketil Stølen, Frank Dederichs, Rainer Weber: Assumption / Commitment Rules for Networks of Asynchronously Communicating Agents

Reihe A

342/3/93 A Thomas Schnekenburger: A Definition of Efficiency of Parallel Programs in Multi-Tasking Environments

342/4/93 A Hans-Joachim Bungartz, Michael Griebel, Dierk Röschke, Christoph Zenger: A Proof of Convergence for the Combination Technique for the Laplace Equation Using Tools of Symbolic Computation

342/5/93 A Manfred Kunde, Rolf Niedermeier, Peter Rossmanith: Faster Sorting and Routing on Grids with Diagonals

342/6/93 A Michael Griebel, Peter Oswald: Remarks on the Abstract Theory of Additive and Multiplicative Schwarz Algorithms

342/7/93 A Christian Sporrer, Herbert Bauer: Corolla Partitioning for Distributed Logic Simulation of VLSI Circuits

342/8/93 A Herbert Bauer, Christian Sporrer: Reducing Rollback Overhead in Time-Warp Based Distributed Simulation with Optimized Incremental State Saving

342/9/93 A Peter Slavkovsky: The Visibility Problem for Single-Valued Surface (z = f(x,y)): The Analysis and the Parallelization of Algorithms

342/10/93 A Ulrich Rüde: Multilevel, Extrapolation, and Sparse Grid Methods

342/11/93 A Hans Regler, Ulrich Rüde: Layout Optimization with Algebraic Multigrid Methods

342/12/93 A Dieter Barnard, Angelika Mader: Model Checking for the Modal Mu-Calculus using Gauß Elimination

342/13/93 A Christoph Pflaum, Ulrich Rüde: Gauß' Adaptive Relaxation for the Multilevel Solution of Partial Differential Equations on Sparse Grids

342/14/93 A Christoph Pflaum: Convergence of the Combination Technique for the Finite Element Solution of Poisson's Equation

342/15/93 A Michael Luby, Wolfgang Ertel: Optimal Parallelization of Las Vegas Algorithms

342/16/93 A Hans-Joachim Bungartz, Michael Griebel, Dierk Röschke, Christoph Zenger: Pointwise Convergence of the Combination Technique for Laplace's Equation

342/17/93 A Georg Stellner, Matthias Schumann, Stefan Lamberts, Thomas Ludwig, Arndt Bode, Martin Kiehl und Rainer Mehlhorn: Developing Multicomputer Applications on Networks of Workstations Using NXLib

342/18/93 A Max Fuchs, Ketil Stølen: Development of a Distributed Min/Max Component

342/19/93 A Johann K. Obermaier: Recovery and Transaction Management in Write-optimized Database Systems

Reihe A

342/20/93 A Sergej Gorlatch: Deriving Efficient Parallel Programs by Systemating Coarsing Specification Parallelism

342/01/94 A Reiner Hüttl, Michael Schneider: Parallel Adaptive Numerical Simulation

342/02/94 A Henning Spruth, Frank Johannes: Parallel Routing of VLSI Circuits Based on Net Independency

342/03/94 A Henning Spruth, Frank Johannes, Kurt Antreich: PHIroute: A Parallel Hierarchical Sea-of-Gates Router

342/04/94 A Martin Kiehl, Rainer Mehlhorn, Matthias Schumann: Parallel Multiple Shooting for Optimal Control Problems Under NX/2

342/05/94 A Christian Suttner, Christoph Goller, Peter Krauss, Klaus-Jörn Lange, Ludwig Thomas, Thomas Schnekenburger: Heuristic Optimization of Parallel Computations

342/06/94 A Andreas Listl: Using Subpages for Cache Coherency Control in Parallel Database Systems

342/07/94 A Manfred Broy, Ketil Stølen: Specification and Refinement of Finite Dataflow Networks - a Relational Approach

342/08/94 A Katharina Spies: Funktionale Spezifikation eines Kommunikationsprotokolls

342/09/94 A Peter A. Krauss: Applying a New Search Space Partitioning Method to Parallel Test Generation for Sequential Circuits

342/10/94 A Manfred Broy: A Functional Rephrasing of the Assumption/Commitment Specification Style

342/11/94 A Eckhardt Holz, Ketil Stølen: An Attempt to Embed a Restricted Version of SDL as a Target Language in Focus

342/12/94 A Christoph Pflaum: A Multi-Level-Algorithm for the Finite-Element-Solution of General Second Order Elliptic Differential Equations on Adaptive Sparse Grids

342/13/94 A Manfred Broy, Max Fuchs, Thomas F. Gritzner, Bernhard Schätz, Katharina Spies, Ketil Stølen: Summary of Case Studies in FOCUS - a Design Method for Distributed Systems

342/14/94 A Maximilian Fuchs: Technologieabhängigkeit von Spezifikationen digitaler Hardware

342/15/94 A M. Griebel, P. Oswald: Tensor Product Type Subspace Splittings And Multilevel Iterative Methods For Anisotropic Problems

342/16/94 A Gheorghe Ştefănescu: Algebra of Flownomials

342/17/94 A Ketil Stølen: A Refinement Relation Supporting the Transition from Unbounded to Bounded Communication Buffers

342/18/94 A Michael Griebel, Tilman Neuhoeffer: A Domain-Oriented Multilevel Algorithm-Implementation and Parallelization

Reihe A

342/19/94 A Michael Griebel, Walter Huber: Turbulence Simulation on Sparse Grids Using the Combination Method

342/20/94 A Johann Schumann: Using the Theorem Prover SETHEO for verifying the development of a Communication Protocol in FOCUS - A Case Study -

342/01/95 A Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids

342/02/95 A Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law

342/03/95 A Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space

342/04/95 A Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes

342/05/95 A Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems

342/06/95 A Maximilian Fuchs: Formal Design of a Model-N Counter

342/07/95 A Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology

342/08/95 A Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies

342/09/95 A Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness

342/10/95 A Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design

342/11/95 A Thomas Schnekenburger: The ALDY Load Distribution System

342/12/95 A Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm

342/13/95 A Stephan Melzer, Javier Esparza: Checking System Properties via Integer Programming

342/14/95 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Point-to-Point Dataflow Networks

342/15/95 A Andrei Kovalyov, Javier Esparza: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs

342/16/95 A Bernhard Schätz, Katharina Spies: Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B Wolfgang Reisig: Petri Nets and Algebraic Specifications

342/2/90 B Jörg Desel: On Abstraction of Nets

342/3/90 B Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems

342/4/90 B Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme

342/1/91 B Barbara Paech1: Concurrency as a Modality

342/2/91 B Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox -Anwenderbeschreibung

342/3/91 B Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen

342/4/91 B Werner Pohlmann: A Limitation of Distributed Simulation Methods

342/5/91 B Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis

342/6/91 B Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme

342/7/91 B W. Reisig: Concurrent Temporal Logic

342/1/92 B Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support

Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support

342/2/92 B Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen

342/1/93 B Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung

342/2/93 B Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick

342/1/94 B Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS