Carl von Ossietzky
Universität Oldenburg

Studiengang Diplom-Informatik

DIPLOMARBEIT

Korrigierte Fassung

Semantische Fundierung von CSP-Z

vorgelegt von: Stefan Hallerstede

Betreuender Gutachter: Prof. Dr. E.-R. Olderog

Zweiter Gutachter: Dr. H. Fleischhack

Oldenburg, den 15. Juli 1997

Zusammenfassung

CSP-Z [Fis96] ist eine Sprache zur Spezifikation zustandsbasierter kommunizierender Systeme, welche die Prozeßalgebra CSP [Hoa85] und die Z-Notation [Spi92] zusammenführt. Die Semantik von CSP-Z-Spezifikationen wird durch das Failures/Divergences-Modell von CSP beschrieben, das von Roscoe [Ros88a] angepaßt wurde, um unbeschränkten Nichtdetermismus besser behandeln zu können. Für CSP gibt es eine Verfeinerungstheorie, die auf dem Failures/Divergences-Modell basiert [Hoa85, Ros88a]. Diese Theorie läßt sich in einfacher Weise auf CSP-Z übertragen. Ziel dieser Arbeit ist es die existierende Verfeinerungstheorie von Z für CSP-Z nutzbar zu machen.

Mit Z können Datentypen prädikativ beschrieben werden, die nicht oder nur mit großem Aufwand zu implementieren sind. Doch bieten diese Datentypen den Vorteil, daß sie zu leichter verständlichen Spezifikationen führen, da bei starker Abstraktion viele für die Funktionalität des zu entwickelnden Programms unwesentliche Details wegfallen. Zu einem späteren Zeitpunkt werden jedoch besser implementierbare Datentypen benötigt.

Es ist in Z möglich, einen (abstrakten) Datentypen durch einen anderen (konkreten) Datentypen zu ersetzen, so daß Programme, die bei Verwendung des abstrakten Datentypen korrekt waren, es auch bei Verwendung des konkreten Datentypen bleiben. Dieses Verfahren heißt *Datenverfeinerung*. Als Beweismethode zum Nachweis einer Datenverfeinerung dient die *Simulation* [WD96].

Bis zu einem gewissen Grade läßt sich die Datenverfeinerungstheorie auch im Zusammenhang mit unsichtbaren Operationen verwenden, die durch Hiding entstehen. Es werden einige Regeln entwickelt, die auf den Regeln zur Datenverfeinerung in Z basieren und benutzt werden können, um Simulationsbeziehungen zwischen CSP-Z-Spezifikationen zu beweisen.

Die Anwendung der entwickelten Regeln wird anhand kleinerer Fallstudien veranschaulicht.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Ein verteiltes System besteht aus einer Anzahl verschiedener paralleler Prozesse, die untereinander kommunizieren. Für die Spezifikation eines verteilten Systems wurden unterschiedliche Formalismen vorgeschlagen, mit denen jeweils verschiedene Aspekte des verteilten Systems modelliert werden können. CSP und Z sind zwei bekannte Vertreter dieser Formalismen.

Die Z-Notation [BN92, Spi92, WD96] ist geeignet, sequentielle Systeme zu spezifizieren, die aus einem Zustandsraum und Operationen auf diesem Zustandsraum bestehen. Die Prozeßalgebra CSP [Hoa85, Ros88a, Ros88b] wurde entwickelt, um dynamische Aspekte eines Systems in einfacher Weise modellieren zu können. Ein CSP-System besteht aus einer Sammlung von Prozessen, die sich bei bestimmten Ereignissen synchronisieren. Eine kurze Beschreibung der Sprachen CSP und Z findet sich in den Abschnitten 2.1 und 2.2 des nächsten Kapitels. Beide Sprachen sind ungeeignet alle Aspekte eines kommunizierenden Systems zu modellieren.

Komplexe Datenstrukturen, die zur Beschreibung von Prozessen selbst notwendig sein können, sind in CSP nur aufwendig darstellbar. Solche Datenstruturen können besser in Z spezifiziert werden. Andererseits bietet Z nicht die Möglichkeit dynamische Operationsfolgen oder Parallelität zu spezifizieren. Das kann wiederum besser in CSP geschehen.

Die Spezifikationssprache CSP-Z [Fis96] integriert die beiden Sprachen CSP und Z. Sie erbt das semantische Modell und einige Operatoren von CSP, wie zB. Hiding ($\setminus$) und Parallelkomposition ($\|$), und das Typkonzept und den Strukturierungsmechanismus von Z. Eine CSP-Z-Spezifikation besteht aus einem CSP-Prozeß und einer Z-Spezifikation. Die Idee dahinter ist, daß der Z-Teil einer CSP-Z-Spezifikation im wesentlichen den Zustandsraum und Zustandsübergänge der Spezifikation beschreibt, während der CSP-Teil die dynamischen Eigenschaften der Spezifikation modelliert.

Eine CSP-Z-Spezifikation kommuniziert mit ihrer Umgebung über eine Menge K von Kanälen. Die Werte v_k, die über einen Kanal $k \in K$ kommuniziert werden können, werden durch einen Z-Typen beschrieben. Mehrere

parallele CSP-Z-Spezifikationen synchronisieren sich bei bestimmten *Kommunikationsereignissen*. Ein Kommunikationsereignis (k, v_k) besteht aus einem Kanal $k \in K$ und einem Kommunikationswert $v_k \in type_k$ und bedeutet, daß der Wert v_k über den Kanal k kommuniziert wird. Die Menge K und die zugehörige Typinformation bilden das *Interface* der Spezifikation.

Jedem $k \in K$ ist im Z-Teil einer Spezifikation ein Z-Operationsschema `com_`k zugeordnet, das bei einem Kommunikationsereignis (k, v_k) einen Zustandswechsel der Spezifikation bewirkt. Befindet sich die Spezifikation in einem Zustand s, so kann eine Kommunikation (k, v_k) nicht stattfinden, wenn die Operation `com_`k keinen Folgezustand für den Zustand s und den Kommunikationswert v berechnen kann. Andernfalls kann die Kommunikation (k, v_k) stattfinden. Der CSP-Z-Teil einer CSP-Spezifikation beschreibt mögliche Reihenfolgen von Kommunikationsereignissen durch einen algebraischen Ausdruck über der Menge K der Kanäle der Spezifikation in direkter Weise. CSP-Teil und Z-Teil einer Spezifikation sind durch Parallelkomposition miteinander verknüpft. Sie sind bezüglich aller durch ihr Interface beschreibbare Kommunikationsereignisse synchronisiert, das heißt ein Kommunikationsereignis einer CSP-Z-Spezifikation kann nur dann eintreten, wenn keiner der beiden Teile es verweigert.

1.1 Beispiel

Anhand eines Beispiels wird in diesem Abschnitt die Sprache CSP-Z erläutert und eine Anwendung des in dieser Arbeit entwickelten Verfeinerungskalküls vorgeführt. Die hier gezeigten CSP-Z-Spezifikationen bestehen nur aus einem Z-Teil. Der CSP-Teil ist leer und hat keine Auswirkungen auf das Verhalten der Spezifikationen.

Abbildung 1.1 zeigt die CSP-Z-Spezifikation eines Puffers `Buffer`. Der Puffer kann (höchstens) zwei ganze Zahlen über den Kanal *in* einlesen und speichern. Die gespeicherten Zahlen können in der Reihenfolge, in der sie eingelesen wurden, über den Kanal *out* wieder ausgelesen werden, wobei sie aus dem Puffer gelöscht werden. Jedem Kanal der Spezifikation ist genau ein Z-Operationsschema `com_`*in* bzw. `com_`*out* zugeordnet. Die Komponenten *@in* bzw. *@out* der beiden Operationsschemata bezeichnen die kommunizierten Werte. Die spezifizierten Operationen werden ausgeführt, wenn über den entsprechenden Kanal *in* oder *out* eine Kommunikation stattfindet. Beide Operationsschemata verändern den Zustand `State` des Puffers, der zwei über *in* kommunizierte Werte in einer Sequenz speichern kann. Bevor die erste Kommunikation stattfindet, befindet sich der Puffer in einem durch das Schema `Init_State` beschriebenen Zustand.

Nach seiner Initialisierung ist der Puffer leer ($buff = \langle\,\rangle$). Die Operation `com_`*out* kann nicht ausgeführt werden, denn dazu müßte $buff \neq \langle\,\rangle$ gelten, und der Kommunikationswert *@out* müßte als letztes Element in der Se-

spec Buffer

channel $in, out : \mathbb{Z}$

State

$buff : seq\mathbb{Z}$

$\#buff \leq 2$

In der $\mathbb{Z}$-Sequenz *buff* können bis zu zwei ganze Zahlen gespeichert werden.

Init_State

State′

$buff' = \langle\,\rangle$

Nach der Initialisierung ist *buff* leer.

com_ *in*

ΔState

$@in : \mathbb{Z}$

$buff' = \langle @in \rangle \frown buff$

Der kommunizierte Wert *@in* wird am Anfang der Sequenz *buff* eingefügt.

com_ *out*

ΔState

$@out : \mathbb{Z}$

$buff = buff' \frown \langle @out \rangle$

Der kommunizierte Wert *@out* wird vom Ende der Sequenz *buff* entnommen.

end_spec Buffer

Abbildung 1.1: Spezifikation eines Puffers Buffer in CSP-Z

quenz *buff* gespeichert sein. Über den Kanal *in* kann jedoch ein beliebiger Wert $v : \mathbb{Z}$ eingelesen werden, da die Operation com_ *in* immer ausgeführt werden kann, wenn die Länge von *buff* kleiner oder gleich eins ist. Nach einer Kommunikation über den Kanal *in* kann eine weitere Kommunikation über den Kanal *in* stattfinden oder eine Kommunikation (out, v), wenn v der in *buff* gespeicherte Wert ist. Nach zwei aufeinanderfolgenden Kommunikationen über den Kanal *in* kann die Operation com_ *in* nicht mehr ausgeführt werden, da sonst die Zustandsinvariante $\#buff \leq 2$ verletzt wäre. Es können aber zwei aufeinanderfolgende Kommunikation über den Kanal *out* stattfinden.

Die vier Schemata State, Init_State, com_ *in* und com_ *out* lassen sich auch als Spezifikation eines Z-Datentypen auffassen[1]. In Z ist eine Daten-

[1] In dieser Arbeit wird der Begriff "Datentyp" verwendet wie üblicherweise der Begriff "abstrakter Datentyp". Die Bezeichnungen "abstrakt" und "konkret" werden benutzt, um den Fortgang einer Verfeinerung zu beschreiben. Soll ein Verfeinerungsschritt durchgeführt werden, so wird die Ausgangsspezifikation "abstrakt" genannt und die verfeinerte Spezifikation "konkret".

spec OneBuffer

channel $in, out : \mathbb{Z}$

State

$value : \mathbb{P}\,\mathbb{Z}$

$\#value \leq 1$

In der Menge $value$ kann höchstens eine ganze Zahl gespeichert werden.

Init_State

$\texttt{State}'$

$value' = \varnothing$

Am Anfang ist kein Element in $value$ gespeichert.

com_ *in*

$\Delta\texttt{State}$

$@in : \mathbb{Z}$

$value = \varnothing$

$@in \in value'$

Wenn in $value$ keine Zahl gespeichert ist, kann eine beliebige Zahl $@in$ gespeichert werden.

com_ *out*

$\Delta\texttt{State}$

$@out : \mathbb{Z}$

$@out \in value$

$value' = \varnothing$

Ist eine Zahl $@out$ in $value$ gespeichert, so kann sie ausgelesen werden. Danach ist $value$ leer.

end_spec OneBuffer

Abbildung 1.2: Spezifikation eines Puffers, der eine Zahl speichern kann

verfeinerungstheorie bekannt, die es erlaubt einen Z-Datentypen durch einen anderen determistischeren zu ersetzen. Diese Theorie läßt sich eingeschränkt auf CSP-Z übertragen, so daß auch dort Datenverfeinerung zur Verfügung steht.

Die Spezifikation `Buffer` läßt sich durch die Parallelkomposition von zwei Puffern datenverfeinern, die jeweils eine Zahl speichern können. Abbildung 1.2 zeigt die Spezifikation eines Puffers `OneBuffer`, der höchstens eine ganze Zahl speichern kann.

Die Spezifikation `ParBuffer` stellt die Parallelkomposition zweier Puffer dar, die jeweils eine Zahl speichern können und über den gemeinsamen Kanal *mid* kommunizieren.

$$\texttt{ParBuffer} \equiv \left(\begin{array}{ll} & \texttt{OneBuffer}[valA/value][mid/out] \\ \| & \texttt{OneBuffer}[valB/value][mid/in] \end{array} \right)$$

Mit einer Beweisregel für die parallele Dekomposition wie in [But92, Rö94] ist möglich zu beweisen, daß `ParBuffer` äquivalent zu der Spezifikation

spec TwoBuffer

channel $in, mid, out : \mathbb{Z}$;

State

$valA : \mathbb{P}\,\mathbb{Z}$
$valB : \mathbb{P}\,\mathbb{Z}$

$\#valA \leq 1 \land \#valB \leq 1$

Init_State

State′

$valA = \varnothing \land valB = \varnothing$

com_ *in*

ΔState
$@in : \mathbb{Z}$

$valA = \varnothing$
$@in \in valA'$
$valB' = valB$

com_ *out*

ΔState
$@out : \mathbb{Z}$

$@out \in valB$
$valB' = \varnothing$
$valA' = valA$

com_ *mid*

ΔState
$@mid : \mathbb{Z}$

$@mid \in valA \land valB = \varnothing$
$@mid \in valB' \land valA' = \varnothing$

end_spec TwoBuffer

Abbildung 1.3: Spezifikation TwoBuffer

TwoBuffer in Abbildung 1.3 ist.

ParBuffer ≡ TwoBuffer

In der Spezifikation TwoBuffer kann nach einer Kommunikation über einen der Kanäle *in* oder *out* höchstens eine Kommunikation über den Kanal *mid* stattfinden. Direkt nach der Initialisierung kann keine Kommunikation über den Kanal *mid* erfolgen. Die Anzahl direkt aufeinanderfolgender Kommunikationen über den *mid* ist durch den Wert von $\#valA$ nach oben beschränkt. Es ist klar, daß Verstecken des Kanals *mid* vor der Umgebung von TwoBuffer nicht zu unendlich vielen aufeinanderfolgenden unsichtbaren Kommunikationen über den Kanal *mid* führen kann. Das heißt TwoBuffer $\setminus \{mid\}$ ist divergenzfrei. Es läßt sich nun folgende Beziehung

beweisen:

`Buffer` wird durch `TwoBuffer` $\setminus \{mid\}$ verfeinert.

Diese Beziehung gilt, wenn `Buffer` das Verhalten von `TwoBuffer` *simulieren* kann. Dazu müssen die Zustände der beiden Spezifikationen miteinander in Beziehung gesetzt werden, so daß die Spezifikation `Buffer` an jeder Kommunikation teilnehmen kann, die für `TwoBuffer` möglich ist. Und falls `TwoBuffer` eine Kommunikation ablehnen kann, dann muß auch `Buffer` diese Kommunikation ablehnen können. Das folgende Schema *Simu* stellt eine solche Simulationsbeziehung her.

Simu

`Buffer.State`
`TwoBuffer.State`

$$
\begin{aligned}
&\#buff = 0 \Leftrightarrow valA = \varnothing \wedge valB = \varnothing \\
&\#buff = 1 \Leftrightarrow \\
&\qquad (valA = \{buff(1)\} \wedge valB = \varnothing) \\
&\quad \vee (valA = \varnothing \wedge valB = \{buff(1)\}) \\
&\#buff = 2 \Leftrightarrow valA = \{buff(1)\} \wedge valB = \{buff(2)\}
\end{aligned}
$$

Zum Nachweis der Verfeinerung von `Buffer` durch `TwoBuffer` $\setminus \{mid\}$ genügt es die folgenden vier Bedungungen zu zeigen.

(1) `Buffer` kann jeden Anfangszustand von `TwoBuffer` simulieren.

(2) Jede Kommunikation, die von `TwoBuffer` in einem *stabilen* Zustand abgelehnt werden kann, kann auch in einem durch *Simu* beschriebenen Zustand von `Buffer` abgelehnt werden. Ein Zustand heißt stabil, wenn in ihm die lokale Kommunikation *mid* nicht bereit ist an einer Kommunikation teilzunehmen.

(3) `Buffer` kann an jeder Kommunikation teilnehmen, an der `TwoBuffer` teilnehmen kann, so daß die Simulationsbeziehung erhalten bleibt.

(4) Nimmt `TwoBuffer` an einer Kommunikation über den unsichtbaren Kanal *mid* teil, dann verändert sich der Zustand von `Buffer` unter der Simulationsbeziehung nicht.

`Buffer` kann die Spezifikation `TwoBuffer` mittels der Simulationsbeziehung *Simu* simulieren, wie aus der folgenden Argumentation zu ersehen ist.

(1) Nach der Initialisierung von `Buffer` ist $\#buff = 0$. Die Initialisierung von `TwoBuffer` liefert $valA = \varnothing$ und $valB = \varnothing$. Damit ist die Simulationsbeziehung *Simu* anfangs hergestellt.

(2a) Befinden sich beide Spezifikationen in entsprechenden Zuständen und kann `Buffer` an einer Kommunikation über den Kanal *in* teilnehmen,

dann gilt $\#buff \leq 1$. Falls $\#buff = 0$ ist, dann folgt $valA = \varnothing$ und `TwoBuffer` kann an derselben Kommunikation teilnehmen. Falls $\#buff = 1$ gilt, so kann `TwoBuffer` entweder an derselben Kommunikation teilnehmen oder eine Kommunikation über den unsichtbaren Kanal *mid* kann auftreten. Umgekehrt gilt also, daß falls sich `TwoBuffer` in einem stabilen Zustand befindet und eine Kommunikation ablehnt, dann kann auch `Buffer` diese Kommunikation ablehnen. Da $\texttt{TwoBuffer} \setminus \{mid\}$ divergenzfrei, folgt daraus, daß die Spezifikation `Buffer` jede Kommunikation, die `TwoBuffer` ablehnen kann, ebenfalls ablehnen kann.

(2b) Analog zu (2a) läßt sich auch für den Kanal *out* zeigen, daß jede von `TwoBuffer` abgelehnte Kommunikation auch von `Buffer` abgelehnt werden kann.

(3a) Kann `TwoBuffer` an einer Kommunikation über den Kanal *in* teilnehmen, dann gilt $valA = \varnothing$, und mit der Beziehung *Simu* folgt $\#buff \leq 1$. Da der Kommunikationswert von `Buffer` an der Stelle *buff* 1 gespeichert wird, ist die Simulationsbeziehung nach der Ausführung von `com_` *in* wieder hergestellt.

(3b) Kann `TwoBuffer` an einer Kommunikation über den Kanal *out* teilnehmen, dann ist $valB = \{v\}$ für ein $v : \mathbb{Z}$ und mit der Simulationsbeziehung folgt, daß v als letztes Element in *buff* gespeichert ist. Also kann `Buffer` an derselben Kommunikation teilnehmen. Nach Ausführung von `com_` *out* ist die Simulationsbeziehung wieder hegestellt, da das letzte Element der Sequenz entfernt wurde und *valB* leer ist.

(4) Kann `TwoBuffer` an einer Kommunikation über den Kanal *mid* teilnehmen, so gilt vorher und hinterher $\#buff = 1$. Es ist sofort zu sehen, daß der Zustand von `Buffer` von der Operation `com_` *mid* unberührt und die Simulationsbeziehung erhalten bleibt.

1.2 Überblick

Am Anfang von Kapitel 2 werden die Sprachen CSP, Z und CSP-Z kurz dargestellt. Der Schwerpunkt dieses Kapitels ist Z-Datenverfeinerung. Die Ergebnisse dieses Kapitels werden in Kapitel 3 benutzt, um Datenverfeinerung in CSP-Z zu definieren.

Kapitel 3 enthält zwei grundlegende Regeln zur Datenverfeinerung in CSP-Z sowie den Nachweis für ihre Korrektheit. Weiterhin wird gezeigt, daß beide Regeln zusammen bezüglich einer eingeschränkten Menge von CSP-Z-Spezifikationen vollständig sind. Zum Beweis des Vollständigkeitsresultats werden Spezifikationen benötigt, die keinen CSP-Prozeß enthalten. Eine Fallstudie in Kapitel 4 erläutert, wie die Regeln angewendet werden, um eine Verfeinerung in CSP-Z zu beweisen.

In Kapitel 5 wird die Verfeinerungstheorie erweitert, so daß auch lokale Kanäle in die Datenverfeinerung einbezogen werden können. Das folgende

Kapitel 6 veranschaulicht die Anwendung einiger Regeln aus Kapitel 5.

1.3 Notationsvereinbarungen

Datenverfeinerung einer Spezifikation A durch eine Spezifikation C wird bewiesen, indem eine Simulation *sim* von C durch A angegeben wird, die gewisse Eigenschaften (E1),...,(Ek) erfüllt, die in einer Regel aufgeführt werden. Simulationen werden durch Z-Schemata beschrieben. Alle CSP-Z -Verfeinerungsregeln in dieser Arbeit sind von folgender Form:

Definition 1.1 (Simulationsmethode "Regelname")
sim heißt Regelname-Schema, wenn folgende Bedingungen erfüllt sind:

(E1)...(Ek) □

Als Simulationsmethode kommt Vorwärts- oder Rückwärtssimulation in Frage. Die beiden Methoden werden in Kapitel 2 erläutert. Der Regelname ist eine der Zeichenfolgen der linken Spalte der Tabelle unten. Die letzte Spalte zeigt, auf welcher Seite die entsprechende Regel zu finden ist.

1.4 Abkürzungen

FS	**F**orwards **S**imulation	44
BS	**B**ackwards **S**imulation	44
LIFS	**L**ocal channel **I**ntroduction by **F**orwards **S**imulation	69
LIBS	**L**ocal channel **I**ntroduction by **B**ackwards **S**imulation	69
WLIFS	**W**eak **L**ocal channel **I**ntroduction by **F**orwards **S**imulation	66
WLIBS	**W**eak **L**ocal channel **I**ntroduction by **B**ackwards **S**imulation	67
LEFS	**L**ocal channel **E**limination by **F**orwards **S**imulation	79
LEBS	**L**ocal channel **E**limination by **B**ackwards **S**imulation	81

Kapitel 2

Einführung

Die Prozeßalgebra CSP [Hoa85] dient der Beschreibung dynamischer Aspekte von kommunizierenden parallelen Systemen. In CSP wird ein System durch eine Anzahl paralleler Prozesse dargestellt, die mit ihrer Umgebung über Kanäle kommunizieren und sich bei bestimmten Kommunikationsereignissen mit ihr synchronisieren. Kanäle transportieren Werte oder dienen allein der Synchronisation von Prozessen. Im letzteren Fall werden die Kanäle selbst auch als Ereignisse bezeichnet.

Mit Z [Spi92] können zustandsbasierte sequentielle Systeme modelliert werden. Eine Z-Spezifikation beschreibt in prädikativer Form den Zustandsraum eines Systems und Operationen auf dem Zustandsraum. Alle Variablen, Konstanten und andere Objekte einer Z-Spezifikation haben einen eindeutig bestimmten Typ, der die möglichen Werte dieser Objekte angibt.

CSP-Z [Fis96] kombiniert die beiden Sprachen CSP und Z und dient der Spezifikation paralleler Systeme, deren Komponenten einen Zustand haben, der für alle anderen Komponenten unsichtbar ist. CSP-Z-Spezifikationen kommunizieren mit ihrer Umgebung über CSP-artige Kanäle, denen ein Z-Typ zugeordnet ist, der die möglichen Kommunikationswerte bestimmt. Der CSP-Teil einer CSP-Z-Spezifikation beschreibt mögliche Kommunikationsfolgen, während der Z-Teil Zustandsänderungen beschreibt, die bei jeder Kommunikation stattfinden.

Abschnitt 2.1 enthält eine kurze Einführung in CSP und Abschnitt 2.2 eine kurze Einführung in Z. Anschließend wird in Abschnitt 2.3 die Sprache CSP-Z beschrieben.In Abschnitt 2.2 werden Z-Datentypen definiert, für die ein Mechanismus bekannt ist, der es erlaubt einen Datentypen durch einen anderen zu ersetzen: die Datenverfeinerung. In Abschnitt 2.4 wird die Theorie der Datenverfeinerung in Z dargelegt, deren Ergebnisse als Grundlage der Datenverfeinerungstheorie in CSP-Z dienen.

2.1 Die Sprache CSP

Die Umgebung eines CSP-Prozesses kann Ereignisse beobachten, an denen der Prozeß teilnimmt. Demnach läßt sich das beobachtbare Verhalten eines Prozesses durch die zeitliche Reihenfolge der Ereignisse, an denen der Prozeß teilnehmen kann, beschreiben. CSP besitzt algebraische Eigenschaften, die es erlauben Prozesse zu manipulieren ohne ihr Verhalten zu ändern. Sind zum Beispiel P und Q zwei CSP-Prozesse, so gilt

$$P \parallel Q \equiv Q \parallel P. \tag{2.1}$$

Der Operator $\parallel$ bezeichnet die Parallelkomposition. Äquivalenz (2.1) bedeutet also, daß die Parallelkomposition von zwei Prozessen kommutativ ist.

Der folgende Prozeß modelliert einen Puffer, der eine ganze Zahl speichern kann. Die Zahlen werden über den Kanal *in* eingelesen und anschließend über den Kanal *out* wieder ausgegeben.

$$\begin{aligned} &\alpha \mathit{Buffer} = \{in.v \mid v \in \mathbb{Z}\} \cup \{out.v \mid v \in \mathbb{Z}\} \\ &\mathit{Buffer} = \Box_{x \in \mathbb{Z}}\ in.x \rightarrow out.x \rightarrow \mathit{Buffer} \end{aligned}$$

Jedem Prozeß P ist ein *Alphabet* αP zugeordnet, das die Ereignisse enthält, an denen der Prozeß möglicherweise teilnehmen kann. Die Ereignisse haben die Gestalt $c.v$, wobei c einen Kanal (zB. *in* oder *out*) und v einen Kommunikationswert (zB. 7 oder eine andere ganze Zahl) bezeichnet, der über den Kanal weitergereicht wird. Der Präfixoperator ($\rightarrow$) beschreibt die Sequenzierung von Ereignissen. Der Prozeß $in.v \rightarrow out.v \rightarrow \mathit{Buffer}$ kann am Ereignis $in.v$ teilnehmen und verhält sich dann wie $out.v \rightarrow \mathit{Buffer}$. Der Operator $\Box$ bezeichnet die externe Wahl. Ein Prozeß $(a \rightarrow P) \Box (b \rightarrow Q)$ läßt seiner Umgebung die Wahl, ob er sich nach einem Ereignis a oder b verhalten soll wie P oder wie Q, wenn $a \neq b$. Gilt $a = b$, dann hat die Umgebung keine Kontrolle darüber, ob sich der Prozeß nach einem Ereignis a wie P oder wie Q verhält[1]. Auch $\Box$ ist kommutativ, weshalb die indizierte Schreibweise ($\Box_{x \in \mathbb{Z}}$) wohldefiniert ist.

Nimmt *Buffer* nach einem Ereignis $in.v$ an einem weiteren Ereignis $out.v$ teil, so verhält er sich danach wieder wie *Buffer*. Der Pufferprozeß ist also rekursiv. Er kann auch unter Verwendung des Rekursionsoperators (μ) in folgender Weise geschrieben werden:

$$\mathit{MuBuffer} = \mu\, \mathit{Buffer} \bullet \Box_{x \in \mathbb{Z}}\ in.x \rightarrow out.x \rightarrow \mathit{Buffer}.$$

Der Prozeß *MuBuffer* ist der kleinste Fixpunkt der Gleichung $\mathit{Buffer} = \Box_{x \in \mathbb{Z}}\ in.x \rightarrow out.x \rightarrow \mathit{Buffer}$. Das Alphabet von *MuBuffer* ist gleich dem

[1] Der CSP-Operator $\sqcap$ bezeichnet die interne Wahl. Der Prozeß $P \sqcap Q$ wählt intern einen der beiden Prozesse und verhält sich dann wie P oder wie Q. Die Umgebung hat also keinen Einfluß darauf, welcher der beiden Prozesse P oder Q ausgewählt wird.

Alphabet von *Buffer*. Der rekursive Prozeß *MuBuffer* läßt sich "entfalten". Er ist äquivalent zu

$$\Box_{x \in \mathbb{Z}} \; in.x \rightarrow out.x \rightarrow MuBuffer$$

Ein wichtiges Konzept von CSP ist die Kommunikation. Der Prozeß *Client* kommuniziert den Wert 7 über die Kanäle *in* und *out*.

$$\alpha Client = \{in.7, out.7\}$$
$$Client = in.7 \rightarrow out.7 \rightarrow Stop$$

Es gilt:

$$(Client \parallel MuBuffer) \equiv Client$$

Der (Basis-) Prozeß *Stop* verweigert die Teilnahme an jedem Ereignis. Die Parallelkomposition von *Client* und *Buffer* bewirkt, daß die gemeinsamen Ereignisse beider Prozesse ($\alpha Client \cap \alpha MuBuffer$) synchronisiert werden. Die anderen Ereignisse können unabhängig voneinander auftreten[2].

Die Äquivalenz $(Client \parallel MuBuffer) \equiv Client$ läßt sich folgendermaßen beweisen (Die zum Beweis benötigten algebraischen Gesetze für endliche Alphabete finden sich in [Hoa85] [3]):

$$\begin{aligned}
&Client \parallel MuBuffer \\
&= \; (in.7 \rightarrow out.7 \rightarrow Stop) \parallel (\mu\, Buffer \bullet \Box_{x \in \mathbb{Z}} \; in.x \rightarrow out.x \rightarrow Buffer) \\
&\equiv \; (in.7 \rightarrow out.7 \rightarrow Stop) \parallel (\Box_{x \in \mathbb{Z}} \; in.x \rightarrow out.x \rightarrow MuBuffer) \\
&\equiv \; in.7 \rightarrow out.7 \rightarrow (Stop \parallel MuBuffer) \\
&\equiv \; in.7 \rightarrow out.7 \rightarrow Stop
\end{aligned}$$

Werden Prozesse parallel komponiert, dann sollen häufig die synchronisierten Ereignisse des Gesamtprozesses für die Umgebung unsichtbar sein. Das wird durch Anwendung des Hiding-Operators ($\setminus$) auf den Gesamtprozeß erreicht. Bei dem Prozeß $P \setminus H$, mit $H \subseteq \alpha P$, sind die Ereignisse aus H nicht sichtbar. Die sichtbaren Ereignisse treten in derselben temporalen Reihenfolge auf wie bei P. Der folgende Prozeß *MidBuffer* kann maximal zwei Zahlen speichern, und sie in der gespeicherten Reihenfolge wieder ausgeben:

$$MidBuffer = \left(\begin{array}{cc} & MuBuffer[mid/out] \\ \parallel & MuBuffer[mid/in] \end{array} \right) \setminus \{mid.v \mid v \in \mathbb{Z}\}$$

[2] CSP hat einen speziellen Interleaving-Operator (|||), der bewirkt, daß auch die gemeinsamen Ereignisse der Prozesse nicht synchronisiert werden.

[3] In [Ros88a] wird untersucht, wie CSP sich im Zusammenspiel mit unendlichen Alphabeten verhält. Dort wird behauptet, daß die algebraischen Gesetze aus [Hoa85] bei Verwendung unendlicher Alphabete größtenteils erhalten bleiben.

Der Ausdruck $MuBuffer[mid/out]$ bewirkt, daß der Kanal *out* des Prozesses *MuBuffer* in *mid* umbenannt wird. Die Ereignisse $mid.v$ sind nicht im Alphabet von *MidBuffer* enthalten. Sie sind vor allen möglichen Umgebungen von *MidBuffer* versteckt. Die Umbenennung $[mid/out]$ kann auch als eine Funktion f von Kanalnamen in Kanalnamen geschrieben werden: $f(in) = in$, $f(out) = mid$. Dann ist $f(MuBuffer)$=$MuBuffer[mid/out]$.

2.1.1 Semantik

Die Standard-Semantik von CSP ist das Failures/Divergences-Modell. In dieser Arbeit wird eine von Roscoe [Ros88a] für unendliche Alphabete angepaßte Variante des Modells benutzt. Darin wird ein Prozeß durch sein *Alphabet* $\mathcal{A}$, seine *Failures* $\mathcal{F}$ und seine *Divergenzen* $\mathcal{D}$ beschrieben.

Das Alphabet ist eine Menge von Ereignissen, an denen der Prozeß teilnehmen kann. Die Divergenzen sind endliche Ereignissequenzen tr, nach denen ein Prozeß unbegrenzt viele unsichtbare Ereignisse durchlaufen kann. Die Failures eines Prozesses sind Paare der Form (tr, X), wobei $tr \in \operatorname{seq} \mathcal{A}$ eine endliche Sequenz von Ereignissen und $X \subseteq \mathcal{A}$ eine Menge von *Refusals* ist. Ist $(tr, X) \in \mathcal{F}$, so kann der Prozeß nach der Teilnahme an der Ereignissequenz tr die Teilnahme an einem weiteren Ereignis $a \in X$ verweigern. Die folgende Failure-Menge $\mathcal{F}_{Client}$ modelliert einen Prozeß mit Alphabet $\mathcal{A} = \{in.7, out.7\}$, der zuerst an dem Ereignis $in.7$ und danach an dem Ereignis $out.7$ teilnehmen kann. Anschließend verweigert er jede weitere Kommunikation.

$$\begin{aligned}\mathcal{F}_{Client} = \{\ & (\langle\rangle, \varnothing),\ (\langle\rangle, \{out.7\}),\ (\langle in.7\rangle, \varnothing),\ (\langle in.7\rangle, \{in.7\}),\\ & (\langle in.7\rangle \frown \langle out.7\rangle, \varnothing),\ (\langle in.7\rangle \frown \langle out.7\rangle, \{in.7\}),\ ^{4}\\ & (\langle in.7\rangle \frown \langle out.7\rangle, \{out.7\}),\\ & (\langle in.7\rangle \frown \langle out.7\rangle, \{in.7, out.7\})\}\end{aligned}$$

Gemäß [Ros88a] hat die Semantik von CSP die in Definition 2.1 genannten Eigenschaften.

Definition 2.1 *Ein* Prozeßmodell *ist ein Tupel* $(\mathcal{A}, \mathcal{F}, \mathcal{D})$*, das folgende Eigenschaften erfüllt:*

$$(\langle\rangle, \varnothing) \in \mathcal{F} \qquad \text{(F1)}$$

$$(s \frown t, \varnothing) \in \mathcal{F} \Rightarrow (s, \varnothing) \in \mathcal{F} \qquad \text{(F2)}$$

$$(t, X) \in \mathcal{F} \wedge Y \subseteq X \Rightarrow (t, Y) \in \mathcal{F} \qquad \text{(F3)}$$

$$(t, X) \in \mathcal{F} \wedge \forall\, a \in Y \bullet (t \frown \langle a\rangle, \varnothing) \notin \mathcal{F} \Rightarrow (t, X \cup Y) \in \mathcal{F} \qquad \text{(F4)}$$

$$s \in \mathcal{D} \wedge t \in \operatorname{seq} \mathcal{A} \Rightarrow s \frown t \in \mathcal{D} \qquad \text{(D1)}$$

$$s \in \mathcal{D} \wedge X \subseteq \mathcal{A} \Rightarrow (s, X) \in \mathcal{F} \qquad \text{(D2)}$$

□

[4] Der Ausdruck $s \frown t$ für zwei Sequenzen s und t bewirkt ihre Konkatenation.

(F1) stellt sicher, daß die Failures-Menge $\mathcal{F}$ nicht-leer ist. Aussage (F2) besagt, daß die Menge der *Traces* ($= \operatorname{dom}\mathcal{F}$) Präfix-abgeschlossen ist. Nach (F3) sind die Refusals abgeschlossen bezüglich Teilmengenbildung. Aussage (F4) bedeutet, daß (unendliche) Mengen von Ereignissen, die nach einer Trace nicht stattfinden können, in einer Refusal-Menge zu dieser Trace enthalten sind.

(D1) bedeutet, daß die Divergenzen eine Suffix-abgeschlossene Menge bilden. Die letzte Aussage (D2) fordert, daß nach einer divergenten Trace kein Ereignis stattfinden kann. (D1) und (D2) zusammen implizieren den chaotischen Abschluß von $\mathcal{F}$ für alle Divergenzen.

Die Semantik eines Prozesses P wird in folgender Form notiert:

$$[\![P]\!] = (\alpha P, \mathcal{F}[\![P]\!], \mathcal{D}[\![P]\!])$$

Die Semantik des Prozesses *Client* auf Seite 15 ist zB.:

$$[\![Client]\!] = (\{in.7, out.7\}, \mathcal{F}_{Client}, \varnothing)$$

Zu allen CSP-Operatoren gibt es entsprechende Funktionen über dem semantischen Bereich (zB. $[\![P \setminus H]\!] = [\![P]\!] \setminus H$), die auch zur Rechtfertigung der algebraischen Gesetze verwendet werden. Äquivalenz von CSP-Prozessen P und Q ist definiert durch $P \equiv Q \Leftrightarrow [\![P]\!] = [\![Q]\!]$.

2.1.2 Verfeinerung

Um zu zeigen daß ein Prozeß P eine Implementation eines Prozesses Q ist, können die algebraischen Gesetze von CSP benutzt werden. Auf Seite 15 wurde diese Methode zum Nachweis von $(Client \parallel MuBuffer) \equiv Client$ angewandt. Die andere schwächere Methode heißt *Verfeinerung*.

Definition 2.2 (CSP-Verfeinerung) *Prozeß Q* verfeinert *Prozeß P, in Zeichen $P \sqsubseteq Q$, wenn $\alpha P = \alpha Q$ und*

$$\mathcal{F}[\![Q]\!] \subseteq \mathcal{F}[\![P]\!] \wedge \mathcal{D}[\![Q]\!] \subseteq \mathcal{D}[\![P]\!].$$

□

Wird ein Prozeß P durch einen Prozeß Q verfeinert, so ist Q also weniger divergent als P und deterministischer als P. Gelten $P \sqsubseteq Q$ und $Q \sqsubseteq P$, dann sind die Prozesse äquivalent, dh. $P \equiv Q$.

2.2 Die Sprache Z

Z [Spi92, WD96] basiert auf der Prädikatenlogik erster Stufe und der axiomatischen Mengenlehre. Die Sprache besitzt eine syntaktische Komponente

zur Strukturierung von Spezifikationen, die *Schema* genannt wird. Ein Schema ist eine benannte Sammlung von Variablen, die mittels eines Prädikats miteinander in Beziehung gesetzt werden. Schemata können mit Kommentaren versehen werden, um informell ihre Bedeutung zu erklären. Ein Schema kann einen Zustandsraum oder auch eine Operation auf einem Zustandsraum beschreiben. Das folgende Schema beschreibt den Zustand eines Puffers, der höchstens drei Elemente speichern kann.

BufferState

[Variablendeklarationen[5]]

$buffer : \operatorname{seq} \mathbb{Z}$

[Prädikat, das die Variablen zueinander in Beziehung setzt
oder ihre Verwendung einschränkt]

$\#buffer \leq 3$

Die Menge $\operatorname{seq} \mathbb{Z}$ der endlichen Sequenzen von $\mathbb{Z}$-Werten ist der Typ der Variable *buffer*. Die Variable kann nur Werte aus dieser Menge annehmen. Eine kurze Erläuterung des Z-Typsystems findet sich in Unterabschnitt 2.2.1.

Eine Operation, um dem Puffer ein Element hinzuzufügen, kann durch das Schema

BufferInsert

$\Delta BufferState$

$item? : \mathbb{Z}$

$head\ buffer' = item?$

$tail\ buffer' = buffer$

beschrieben werden.

Die Deklaration $\Delta BufferState$ zeigt an, daß *BufferInsert* eine Zustandsänderung bewirkt. Sie führt die Variablen *BufferState* und *BufferState′* ein. Die erste bezeichnet den Zustand vor der Änderung und die zweite den Zustand nach der Änderung. Die Variable $item? : \mathbb{Z}$ ist die Eingabe der Operation. Eingabevariablen enden nach Konvention mit einem Fragezeichen. Das Prädikat $head\ buffer' = item \land tail\ buffer' = buffer$[6] stellt eine Beziehung zwischen den Variablen her[7]. Es bedeutet, daß *BufferInsert* die Eingabe *item*? vorn in die Sequenz *buffer* einfügt. In der folgenden Operation *BufferRemove*, die das letzte Element aus dem Puffer liest und es daraus löscht, wird eine Ausgabevariable *item*! deklariert. Ausgabevariablen enden per Konvention mit einem Ausrufezeichen.

[5] Einzelne Variablen eines Schemas werden auch *Komponenten* genannt.

[6] Zwischen den Zeilen bestehen implizite Konjunktionen.

[7] Die Funktionen *head* und *tail* sind Bestandteil des Mathematical Toolkit von Z. Im Anhang befinden sich kurze Beschreibungen der in dieser Arbeit benötigten Definitionen des Mathematical Toolkit.

BufferRemove

$\Delta BufferState$
$item! : \mathbb{Z}$

$last\ buffer = item!$
$front\ buffer = buffer'$

Eine Operation muß nicht für jeden Zustand eine Zustandsänderung bewirken können. Zum Beispiel kann nur ein Element in den Puffer eingefügt werden, wenn er nicht voll ist. Mit dem Operator pre kann die Vorbedingung einer Operation berechnet werden. Sie zeigt an für welche Zustände (und Eingaben) die Operation eine Zustandsänderung definiert. Die Vorbedingung von *BufferInsert* ist

$$\begin{aligned} &\text{pre}\, BufferInsert \\ &\quad \widehat{=} \exists\, BufferState' \bullet BufferInsert \\ &\quad \equiv [\, buffer : \text{seq}\, \mathbb{Z} \mid \#buffer < 3 \,]\,. \end{aligned}$$

Das Schema pre *BufferInsert* ist durch den Ausdruck auf der rechten Seite von $\widehat{=}$ definiert. Die letzte Zeile zeigt eine alternative Schreibweise für Z-Schemata.

Bei der Berechnung der Vorbedingung von *BufferRemove* muß auch sichergestellt werden, daß ein Ausgabewert *item*! existiert.

$$\begin{aligned} &\text{pre}\, BufferRemove \\ &\quad \widehat{=} \exists\, BufferState';\ item! : \mathbb{Z} \bullet BufferRemove \\ &\quad \equiv [\, buffer : \text{seq}\, \mathbb{Z} \mid \#buffer > 0 \,]\,. \end{aligned}$$

Ist eine Z-Spezifikation von einer speziellen Form, so wird sie als Z-Datentyp bezeichnet. Z-Datentypen sind von den mengentheoretischen Typen zu unterscheiden, die Bestandteil von Z sind. Im nächsten Unterabschnitt werden die mengentheoretischen Typen kurz erläutert und im darauf folgenden die Z-Datentypen, welche in dieser Arbeit von besonderer Bedeutung sind. Im letzten Unterabschnitt werden Notationen eingeführt, die später benötigt werden.

2.2.1 Das Z-Typkonzept

Jedes Objekt in einer Z-Spezifikation hat einen Typ. Ein Typ ist eine maximale Menge innerhalb der Spezifikation. Wird die Menge der natürlichen Zahlen durch $\mathbb{N} == \{x : \mathbb{Z} \mid x > 0\}$ definiert, dann hat ein Objekt $n : \mathbb{N}$ den Typen $\mathbb{Z}$.

Typen, deren Elemente keine interne Struktur besitzen, heißen *Basistypen*. Zum Beispiel führt die Deklaration

$$[Datum]$$

den Basistypen *Datum* ein, dessen Elemente $x : Datum$ atomar sind. Die Menge $\mathbb{Z}$ der ganzen Zahlen ist der einzige vordefinierte Basistyp in Z. Komplexere Typen lassen sich durch Verwendung der drei Typkonstruktoren von Z bilden: Potenzmengentypen ($\mathbb{P}\,\tau$), Produkttypen (das cartesische Produkt endlich vieler Typen: $\tau_1 \times \tau_2 \times \ldots \times \tau_n$), Schematypen ($[\, Name_1 : \tau_1;\ Name_2 : \tau_2;\ \ldots Name_n : \tau_n \,]$), wobei τ,τ_1,τ_2,...,τ_n Typen sind.

2.2.2 Z-Datentypen

Ein Z-Datentyp *Dat* besteht aus einem Zustandsschema *State*, einem Initialisierungsschema *Init* und einigen Operationsschemata $Op_1, \ldots, Op_n$. Ein Operationsschema *Op* definiert in prädikativer Form mögliche Zustandsübergänge $\Delta State$ und Ausgaben $o!$ in Abhängigkeit vom gegenwärtigen Zustand *State* und den Eingaben $i?$. Das Initialisierungsschema *Init* definiert die möglichen Anfangszustände des Z-Datentypen.

Init

$State'$

---

p_{init}

Op$_k$

$\Delta State$
$i_1^k? : type_{i_1^k}$
$\vdots$
$i_m^k? : type_{i_m^k}$
$o_1^k! : type_{o_1^k}$
$\vdots$
$o_n^k! : type_{o_n^k}$

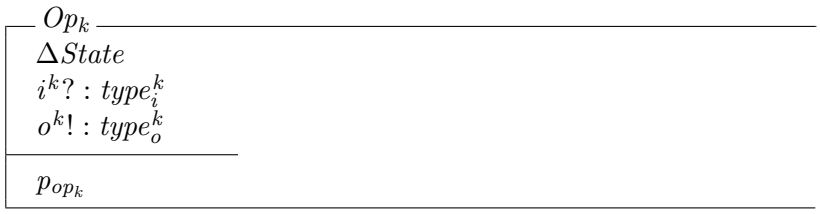
---

p_{op_k}

Das Schema Op_k läßt sich vereinfachen, wenn die Eingaben und Ausgaben zu Vektoren zusammengefaßt werden. Der Typ der Vektoren ist das cartesische Produkt der Eingaben bzw. Ausgaben.

$$type_i^k == type_{i_1^k} \times \ldots \times type_{i_m^k}$$
$$type_o^k == type_{o_1^k} \times \ldots \times type_{o_n^k}$$

Damit läßt sich Op_k mit jeweils einer Eingabe und einer Ausgabe schreiben.



Definition 2.3 (Initialisierungstheorem) *Ist Init das Initialisierungsschema eines* Z*-Datentypen Dat mit Zustandsschema State und gilt*

$$\exists\, State' \bullet Init,$$

dann erfüllt Dat das Initialisierungstheorem. □

2.2.3 Notationen

Das Mathematical Toolkit von Z enthält eine Vielzahl vordefinierter Funktionen und Relationen, und einige zusammengesetzte Typkonstruktoren. Die in dieser Arbeit benötigten Definitionen des Mathematical Toolkit werden in Anhang A informell aufgeführt. Formale Definitionen finden sich in [Spi92, BN92]. Benötigte Notationen, die nicht Bestandteil des Mathematical Toolkit sind, werden an dieser Stelle definiert. Diese Notationen werden hauptsächlich in Abschnitt 2.2 verwendet, um zu einer kompakteren Darstellung zu gelangen.

Die Identische Forsetzung $\rho[K]$ einer Relation $\rho : M \leftrightarrow N$ enthält alle Elemente $(m, k) \mapsto (n, k)$ für die $k \in K$ und $m\rho n$ gilt. Das heißt in der ersten Komponente verhält sich $\rho[K]$ wie ρ und in der zweiten wie $\mathrm{id}[K] == \{k : K \bullet k \mapsto k\}$, der Identität auf K.

Definition 2.4 (Identische Fortsetzung) *Ist $\rho : M \leftrightarrow N$ eine Relation, so ist die* identische Fortsetzung $\rho[K]$ *von ρ auf eine Menge K durch*

$$\rho[K] == \{m : M;\ n : N;\ k : K \mid m\rho n \bullet (m, k) \mapsto (n, k)\}$$

definiert. □

Analog zur Wertsubstitution für Prädikate läßt sich eine solche Substitution auch für Relationen definieren.

Definition 2.5 (Relationale Substitution) *Seien $R_1, \ldots, R_n$, $n \in \mathbb{N}$, Mengen, und $X == R_1 \times \ldots \times R_n$ das cartesische Produkt.*

Ist $\rho \subseteq X$ eine Menge mit Typ $\mathbb{P} X$, so läßt sich ρ als Prädikat

$$\Big(\rho^P(V_1, \ldots, V_n)\Big) \Leftrightarrow \Big((V_1, \ldots, V_n) \in \rho\Big)$$

mit den freien Variablen $V_1, \ldots, V_n$ *auffassen. Für Werte* $r_{i_1} : R_{i_1}, \ldots, r_{i_k} : R_{i_k}$ *mit* $\{i_1, i_2, \ldots, i_k \mid i_a < i_b \Rightarrow a < b\} \subseteq \{1 \,..\, n\}$ *ist die relationale Substitution definiert durch*

$$\rho[r_{i_1}/V_{i_1}, \ldots, r_{i_k}/V_{i_k}] == \\ \{(r_{j_1}, \ldots, r_{j_h}) : \phi \mid \rho^P[r_{i_1}/V_{i_1}, \ldots, r_{i_k}/V_{i_k}](r_{j_1}, \ldots, r_{j_h})\},$$

wobei $\phi = R_{j_1} \times R_{j_2} \times \ldots \times R_{j_h}$ *mit* $\{j_1, j_2, \ldots, j_h\} = (1 \,..\, n) \setminus \{i_1, i_2, \ldots, i_k\}$ *und* $j_a < j_b \Rightarrow a < b$. □

Sind die Namen der Mengen $R_{i_1}, \ldots, R_{i_k}$ untereinander verschieden und auch verschieden von den Namen der restlichen Mengen, dann ist es zweckmäßig die Namen $R_{i_1}, \ldots, R_{i_k}$ als Variablennamen zu verwenden anstatt $V_{i_1}, \ldots, V_{i_k}$.

2.3 Die Sprache CSP-Z

Eine CSP-Z-Spezifikation setzt sich aus einer Z-Spezifikation, die syntaktisch mit einem Z-Datentypen übereinstimmt, und einem CSP-Prozeß zusammen. Die CSP-Z-Spezifikation *OldClock* beschreibt eine alte Uhr, die abwechselnd an den Ereignissen *tick*.v (mit $v \in 1 \,..\, 24$) und *clunk* teilnimmt.

`spec OldClock`

`channel` *tick* : $1 \,..\, 24$;
`channel` *clunk* : `signal`;
`main` = *tick* → *clunk* → `main`;

State

$time : 1 \,..\, 24$

Init_State

State′

$time' = 12$

com_ *tick*

ΔState
$@tick : 1 \,..\, 24$

$@tick = time$
$time' = (time \bmod 24) + 1$

com_ *clunk*

ΞState [8]

`end_spec OldClock`

In CSP-Z wird das Interface (oder Alphabet) einer Spezifikation deklariert. Dazu dient das reservierte Wort `channel`. Bei der Deklaration eines

[8] ΞState spezifiziert eine Operation, die keine Zustandsänderung bewirkt.

Kanals wird diesem ein Typ zugeordnet, der bestimmt welche Werte über den Kanal kommuniziert werden können. Der Z-Basistyp `signal` wird in CSP-Z verwendet, um Kanäle zu modellieren, die keine Werte transportieren, sondern allein der Synchronisation dienen.

$$[\texttt{signal}]$$

$$\mid \#\texttt{signal} = 1$$

Über den Kanal *tick* können die Uhrzeiten $1 \ldots 24$ kommuniziert werden. Der Kanal *clunk* modelliert den Lärm, den die Uhr während ihres Betriebes macht.

Jede CSP-Z-Spezifikation enthält ein Schema `State`, das den Zustandsraum beschreibt, und ein Schema `Init_State`, das die möglichen Anfangszustände angibt. Zu jedem CSP-Z-Kanal c gibt es ein Operationsschema `com_` c, das ausgeführt wird, wenn eine Kommunikation über den Kanal stattfindet. Ist ein Kanal c nicht vom Typ `signal`, dann hat das zugehörige Operationsschema `com_` c eine Komponente $@c$, die den kommunizierten Wert beschreibt. Der CSP-Teil der Spezifikation enthält einen Prozeß mit dem Namen `main`. Sein Alphabet ist die Menge der Kanalnamen der CSP-Z-Spezifikation, in der er enthalten ist. Die Namen `State`, `Init_State`, `com_` c und `main` sind reservierte Schlüsselwörter. Namen einer Spezifikation können im Z-Stil durch die Punktnotation referenziert werden; zB. bezeichnet der Name `OldClock.com_` *tick* das Operationsschema `com_` *tick* der Spezifikation `OldClock`.

Eine Kommunikation kann nur stattfinden, wenn der Zustand des CSP-Prozesses `main` es zuläßt und die Vorbedingung der dem Kanal zugeordneten Operation erfüllt ist. Die Kommunikation kann abgelehnt werden, wenn die Vorbedingung der Operation nicht erfüllt ist, oder der Prozeß `main` sie ablehnt.

Nach der Initialisierung der Uhr *OldClock* ist *time* gleich 12. Die Uhr kann nur mittags in Betrieb genommen werden. Die beiden Operationen `com_` *tick* und `com_` *clunk* könnten immer ausgeführt werden, da ihre Vorbedingungen beide true sind. Die erste Kommunikation der Uhr kann aber nur *tick*.12 sein, da im CSP-Prozeß `main` spezifiziert ist, daß die Kommunikationen *tick* und *clunk* nur abwechselnd, beginnend mit *tick*, stattfinden können. Die weiteren Kommunikationen der Uhr sind also *clunk*, dann *tick*.13, dann *clunk*, dann *tick*.14, dann ....

2.3.1 Syntax

Eine CSP-Z-Spezifikation besteht aus einer Interface-Deklaration, einer Deklaration lokaler Kanäle und den CSP- und Z-Teilen.

```
spec SpecName
    Interface
    LocalChannels
    CSP-Process
    Z-Schemas
end_spec SpecName
```

Das Interface einer Spezifikation enthält Kanaldeklarationen der Form

`channel` *ChannelName* : Z-*Type*.

Lokale Kanäle werden ähnlich zu Interface-Kanälen deklariert und beschreiben interne Kommunikationen bzw. Operationen der Spezifikation.

`local_channel` *ChannelName* : Z-*Type*

Die folgenden CSP-Operatoren können zur Kombination von CSP-Z-Spezifikationen verwendet werden:

Spec	=	$\texttt{Spec}_1 \,\Box\, \texttt{Spec}_2$	externe Wahl
	|	$\texttt{Spec}_1 \sqcap \texttt{Spec}_2$	interne Wahl
	|	$\texttt{Spec}_1 \,\|[\, K \,]|\, \texttt{Spec}_2$	Parallelkomposition über K
	|	$\texttt{Spec}_1 \parallel \texttt{Spec}_2$	Parallelkomposition
	|	$\texttt{Spec}_1 \,\|||\, \texttt{Spec}_2$	Interleaving
	|	$\texttt{Spec}_1 \setminus H$	Hiding
	|	$f(\texttt{Spec}_1)$	Umbenennung

Dabei ist f eine Funktion von Kanalnamen in Kanalnamen. Zur Anwendung der binären Operatoren müssen die Interfaces beider Spezifikationen Typkompatibel sein. Es gibt keinen Kanal mit zwei verschiedenen Typen. Zur Anwendung der Operatoren $\Box$ und $\sqcap$ müssen die Interfaces beider Spezifikationen identisch sein. Der Operator $|[\, K \,]|$ bezeichnet die Parallelkomposition zweier Spezifikationen, die über dem Alphabet K synchronisieren. Die Parallelkomposition $\texttt{Spec}_1 \parallel \texttt{Spec}_2$ ist eine Abkürzung für $\texttt{Spec}_1 \,|[\, I_{12} \,]|\, \texttt{Spec}_2$, wobei I_{12} den Durchschnitt beider Interfaces der Spezifikationen $\texttt{Spec}_1$ und $\texttt{Spec}_2$ bezeichne. Und der Interleaving-Operator $\texttt{Spec}_1 \,|||\, \texttt{Spec}_2$ ist eine Abkürzung für $\texttt{Spec}_1 \,|[\, \varnothing \,]|\, \texttt{Spec}_2$

Die Anwendung des Hiding-Operators führt lokale Kanäle in eine Spezifikation, die keine lokalen Kanäle enthält, folgendermaßen ein:

$$\begin{aligned} &\texttt{spec channel } c : \mathit{type}_c;\ \texttt{P};\ \texttt{Z end\_spec} \setminus \{c.v \mid v : \mathit{type}_v\} \\ \equiv\ &\texttt{spec local\_channel } c : \mathit{type}_c;\ \texttt{P};\ \texttt{Z end\_spec} \end{aligned}$$

Als gleichwertige Kurzschreibweise wird auch

$$\texttt{spec channel } c : \mathit{type}_c;\ \texttt{P};\ \texttt{Z end\_spec} \setminus \{c\}$$

verwendet. Lokale Kanäle und damit verbundene Probleme werden in Kapitel 5 ausführlich behandelt.

Die Menge aller syntaktisch korrekten und typkonsistenten CSP-Z-Spezifikationen, deren Z-Teil das Initialisierungstheorem (Definition 2.3) erfüllt, wird mit *SPEC* bezeichnet. Typkonsistent bedeutet, daß die Spezifikation keine nicht-auswertbaren Ausdrücke enthält. Ist zB. $s : \mathrm{seq}\,\mathbb{Z}$ eine Sequenz von ganzen Zahlen und $i : \mathbb{Z}$ eine ganze Zahl, so ist der Ausdruck $i = s$ nicht typkonsistent. Das Initialisierungstheorem stellt sicher, daß die Spezifikation einen Anfangszustand hat.

2.3.2 Semantik

Das semantische Modell von CSP-Z ist das Failures/Divergences-Modell (Definition 2.1), wobei das Interface einer CSP-Z-Spezifikation die Rolle des Alphabets eines CSP-Prozesses übernimmt.

$$[\![\texttt{Spec}]\!] = (\mathcal{I}[\![\texttt{Spec}]\!], \mathcal{F}[\![\texttt{Spec}]\!], \mathcal{D}[\![\texttt{Spec}]\!])$$

Die semantische Darstellung des Interfaces enthält die Kanalnamen und die zugehörigen Typinformationen. Die Menge der Kanalnamen eines Interfaces $\mathcal{I}$ wird mit $Chans(\mathcal{I})$ bezeichnet und der Typ eines Kanals $c \in Chans(\mathcal{I})$ mit $type_{\mathcal{I}}(c)$. Die Definition der Semantik der einzelnen CSP-Z-Operatoren ist im Anhang zu finden. Die Semantik von CSP läßt sich in einfacher Weise nach CSP-Z übertragen.

Definition 2.6 *Ist* $\texttt{Spec} = \texttt{spec I; P end\_spec}$ *mit Interface* $\mathcal{I}$ *und* P *ein* CSP*-Prozeß der Form* $\texttt{main} = P_0$*, dann ist*

$$[\![\texttt{Spec}]\!] = (\mathcal{I}, \mathcal{F}, \mathcal{D})$$

wobei

$$\mathcal{F} == \{(tr, R) : seqComm(\mathcal{I}) \times Comm(\mathcal{I}) \mid (tr \uparrow Chans(\mathcal{I}), R \uparrow Chans(\mathcal{I})) \in \mathcal{F}[\![P]\!]\}$$

$$\mathcal{D} == \{tr : seqComm(\mathcal{I}) \mid tr \uparrow Chans(\mathcal{I}) \in \mathcal{D}[\![P]\!]\}$$

(Die beiden semantischen Abbildungen $\mathcal{F}[\![P]\!]$ *und* $\mathcal{D}[\![P]\!]$ *sind die in Abschnitt 2.1 für die Sprache* CSP *definierten.)* □

Der Restriktions-Operator $X \uparrow \{c\}$ bewirkt, daß alle Kommunikationen (c, v) entfernt werden, welche die Trace X oder die Menge X enthält, auf die der Operator angewandt wird.

Bezeichne $SPEC(\mathcal{I})$ die Menge aller CSP-Z-Spezifikationen mit Interface $\mathcal{I}$. Die möglichen Kommunikationen eines Interfaces $\mathcal{I}$ lassen sich mit der Funktion *Comm* ermitteln.

$$Comm(\mathcal{I}) == \{(c, v) \mid c \in Chans(\mathcal{I}) \wedge v \in type_{\mathcal{I}}(c)\}$$

Sei im folgenden $\mathtt{S} = \mathtt{spec\,I;\ Z\,end\_spec}$ eine CSP-Z-Spezifikation aus $SPEC(\mathcal{I})$ ohne CSP-Teil und ohne lokale Kanäle.

Ist $(c, v) \in Comm(\mathcal{I})$ und $\mathtt{S.com\_}\, c \mathrel{\widehat=} [\, \Delta\mathtt{S.State};\ @c : type_{\mathcal{I}}(c) \mid p_c \,]$ die zugehörige Operation in S, dann läßt sich die Variable $@c$ durch den Kommunikationswert v substituieren.

$$\mathtt{S.com\_}\,(c, v) \mathrel{\widehat=} (\mathtt{S.com\_}\, c[v/@c]) \setminus (v)$$

Mit dieser Schreibweise lassen sich nun Traces und Operationsfolgen miteinander in Beziehung setzen. Sei $tr : \mathrm{seq}\, Comm(\mathcal{I})$.

$$\mathtt{S}_{\langle\rangle} \mathrel{\widehat=} \mathtt{S.Init\_State}$$
$$\mathtt{S}_{tr \frown \langle (c,v) \rangle} \mathrel{\widehat=} \mathtt{S}_{tr} \fatsemi \mathtt{S.com\_}\,(c, v)$$

Die Sequentielle Komposition der Operationsschemata $\mathtt{S}_{tr}$ liefert ein Schema der Form:

$\mathtt{State}'$
p_{tr}

Es wird sich als zweckmäßig erweisen, die Zustandskomponente $\mathtt{State}'$ in State umzubenennen.

$$\mathtt{S}^{\star}_{tr} \mathrel{\widehat=} (\mathtt{S}_{tr})[\mathtt{State}/\mathtt{State}']$$

Um die Semantik der Spezifikation S zu beschreiben, müssen die Refusals, die Failures und die Divergenzen von S angegeben werden. Die Divergenzen sind die leere Menge. Die Refusals einer Spezifikation werden in Abhängigkeit vom gegenwärtigen Zustand definiert. Das ist genau der Zustand, den $\mathtt{S}^{\star}_{tr}$ liefert.

$$Ref_{\mathtt{S}}\, R \mathrel{\widehat=} \forall (c, v) \in R \bullet \neg\, \mathrm{pre}\, \mathtt{S.com\_}\,(c, v), \quad \text{wobei } R \subseteq Comm(\mathcal{I})$$

Die Failures von S lassen sich in der folgenden Form definieren:

$$\mathcal{F}[\![\mathtt{S}]\!] == \{(tr, R) : \mathrm{seq}\, Comm(\mathcal{I}) \times \mathbb{P}\, Comm(\mathcal{I}) \mid \exists\, \mathtt{S.State} \bullet \mathtt{S}^{\star}_{tr} \wedge Ref_{\mathtt{S}}\, R\}$$

Die Traces, an denen S teilnehmen kann werden mit $traces(\mathtt{S})$ bezeichnet.

$$traces(\mathtt{S}) == \{tr : \mathrm{seq}\, Comm(\mathcal{I}) \mid (tr, \varnothing) \in \mathcal{F}[\![\mathtt{S}]\!]\}$$

Die Semantik einer CSP-Z-Spezifikation $\mathtt{S} = \mathtt{spec\,I;\ Z\,end\_spec}$ ist durch $[\![\mathtt{S}]\!] = (\mathcal{I}[\![\mathtt{S}]\!], \mathcal{F}[\![\mathtt{S}]\!], \varnothing)$ definiert. Die Semantik einer CSP-Z-Spezifikation ergibt sich aus der Parallelkomposition ihres CSP-Teils und ihres Z-Teils.

Definition 2.7 *Die Semantik einer Spezifikation* $\mathtt{S} = \mathtt{spec\,I;\ P;\ Z\,end\_spec}$ *aus* $SPEC(\mathcal{I})$ *ist durch die Parallelkomposition*

$$\mathtt{S} \equiv \mathtt{spec\,I;\ P\,end\_spec} \,|[\,\mathcal{I}\,]|\, \mathtt{spec\,I;\ Z\,end\_spec}$$

definiert. □

2.3.3 Verfeinerung

In CSP-Z ist Verfeinerung analog zur Verfeinerung in CSP definiert. Die Verfeinerung des CSP-Teils einer Spezifikation impliziert in offensichtlicher Weise eine Verfeinerung der Spezifikation.

Definition 2.8 *Seien* $M_1 = (\mathcal{I}_1, \mathcal{F}_1, \mathcal{D}_1)$ *und* $M_2 = (\mathcal{I}_2, \mathcal{F}_2, \mathcal{D}_2)$ *zwei* CSP-Z*-Modelle. Dann wird* M_1 *durch* M_2 verfeinert*, in Zeichen* $M_1 \sqsubseteq M_2$*, wenn*

$$\begin{aligned} \mathcal{I}_2 &= \mathcal{I}_1, \\ \mathcal{F}_2 &\subseteq \mathcal{F}_1, \\ \mathcal{D}_2 &\subseteq \mathcal{D}_1. \end{aligned}$$

Gelten $M_1 \sqsubseteq M_2$ *und* $M_1 \sqsubseteq M_2$*, dann wird das als* $M_1 \equiv M_2$ *notiert.* □

2.4 Datenverfeinerung in Z

Datenverfeinerung wird zunächst für Datentypen *dt* definiert, deren Zustandsraum eine Menge *st* und deren Operationen partielle Relationen *do* über dem Zustandsraum sind. Angenommen *at* ist ein solcher Datentyp mit dem Zustandsraum *st* und einer Operation *ao*. Eine vereinfachte Verfeinerung von *at* durch einen Datentypen *ct* mit demselben Zustandsraum *st* und einer Operation *co* könnte einfach durch die Inklusion $co \subseteq ao$ definiert werden. Dann würde aber $co = \varnothing$ jede Operation *ao* verfeinern.

Die Menge $(\operatorname{dom} co) \setminus st$ bezeichnet die Zustände für die *co* undefiniert ist. Es wird hier die in Z übliche Sichtweise übernommen, daß eine Anwendung von *co* auf einen Zustand außerhalb von $\operatorname{dom} co$ zu chaotischem Verhalten führt. Gilt also $co \subseteq ao$, so kann sich *co* chaotischer Verhalten als *ao*. Diese Verfeinerungstheorie würde also die totale Korrektheit einer Operation nicht erhalten. Deshalb werden in der betrachteten Verfeinerungstheorie nur totale Operationen betrachtet; und die partiellen Operationen *do* werden in totale Operationen $do^\bullet$ eingebettet, die chaotisches Verhalten durch den speziellen Zustand $\perp$ modellieren. Die Verfeinerung von *at* durch *ct* wird durch $co^\bullet \subseteq ao^\bullet$ definiert. Die so enstandene Verfeinerungstheorie erhält die totale Korrektheit von Operationen.

Es werden zwei Simulationsmethoden entwickelt, mit deren Hilfe eine Datenverfeinerung bewiesen werden kann: Vorwärtssimulation und Rückwärtssimulation. Das wesentliche Merkmal der Vorwärtssimulation ist, rückblickend das Verhalten des zu simulierenden Datentyps nachzuvollziehen, während bei der Rückwärtssimulation zukünftiges Verhalten vorhergesagt wird. Näheres dazu findet sich in [AL91, Mor88, GM89].

Die Semantik eines Z-Datentypen *Abs* wird später durch einen Datentypen $\mathcal{M}^{\mathsf{Z}}[\![Abs]\!] = at$ definiert. Der Datentyp *at* ist von der im letzten Absatz beschriebenen Form. Die Datenverfeinerung von *Abs* durch einen

Z-Datentypen *Con* mit Semantik $\mathcal{M}^{\mathsf{Z}}[\![Con]\!] = ct$ kann bewiesen werden, indem gezeigt wird, daß *at* durch *ct* verfeinert wird. Dieser Umweg über die Semantik läßt sich jedoch vermeiden, wenn eine Verfeinerungsbeziehung für Z-Datentypen, *Con verfeinert Abs*, existiert, so daß gilt:

Con verfeinert Abs $\Rightarrow$ *at* wird durch *ct* verfeinert.

Die *verfeinert*-Beziehung wird mittels eines Repräsentationsschemas *Rep* hergestellt, dessen Semantik $\mathcal{R}^{\mathsf{Z}}[\![Rep]\!]$ eine Vorwärtssimulation oder eine Rückwärtssimulation enthalten kann.

Die Darstellungen dieses Abschnitts stammen hauptsächlich aus [WD96]. In Unterabschnitt 2.4.1 wird die Verfeinerungstheorie für die Datentypen definiert, die die Semantik der Z-Datentypen beschreiben. In Unterabschnitt 2.4.2 wird diese Verfeinerungstheorie auf Z-Datentypen übertragen. Das Resultat sind zwei Kalkülregeln, die zum Beweis einer Datenverfeinerung verwendet werden können. Am Ende des Unterabschnitts 2.4.1 werden Vorbereitungen zum Beweis der Korrektheit der beiden Kalkülregeln getroffen. Im letzten Unterabschnitt dieses Abschnitts wird noch eine weitere Möglichkeit diskutiert Datenverfeinerung zu definieren.

2.4.1 Der semantische Bereich

Ein Datentyp besteht aus einer Initialisierung, einer Menge von Operationen, einer Finalisierung und einer Menge von Zuständen, in denen sich eine Variable dieses Datentyps im Laufe der Zeit befinden kann. Es werden die beiden folgenden Annahmen über Datentypen gemacht:

- Der Zustand einer Variablen eines Datentyps kann nur durch ihre Initialisierung, Finalisierung und ihre Operationen verändert werden.
- Initialisierung, Finalisierung, und Operationen eines Datentyps sind nur über dem eigenen Zustandsraum definiert. (Die Ausführung einer Operation hat keine Nebeneffekte.)

Ein *Datentyp dt* ist ein Tupel $dt = (ds, di, df, \{do_1, \ldots, do_n\})$, $n \in \mathbb{N}$, wobei

1. ds eine *Menge von Zuständen* ist,
2. $di : Void \leftrightarrow ds$ eine *Initialisierung* ist,
3. $df : ds \rightarrow Void$ eine *Finalisierung* ist,
4. *Void* ein Basistyp mit genau einem Element *void* ist,
5. $do_k : ds \times inp_k \leftrightarrow ds \times outp_k$ die *Operationen* des Datentyps sind.

Die Menge inp_k ist eine Menge von Eingabewerten und $outp_k$ eine Menge von Ausgabewerten der Operation do_k. Operationen sind partielle oder totale Relationen. Durch dom do_k sind Vorbedingungen für die Anwendung der Operationen gegeben. Bevor eine Operation eines Datentyps dt angewandt werden kann, muß dt initialisiert werden. Wird er nicht mehr benötigt, so wird er finalisiert. Die Finalisierung ist ein technisches Hilfsmittel, das eine kompakte Definition der Datenverfeinerung ermöglicht. Der Basistyp *Void* beschreibt den Zustand, in dem keine Operation ausgeführt werden kann. Jeder Datentyp befindet sich vor seiner Initialisierung und nach seiner Finalisierung im Zustand *void*. Ist $di = \varnothing$, dann ist es nicht möglich dt zu initialisieren und eine der Operationen anzuwenden.

Definition 2.9 *Ein Datentyp* $dt = (ds, di, df, d)$ *heißt* initialisierbar*, wenn* $di \neq \varnothing$. □

Eine Operation do eines Datentyps ist eine Relation von Zuständen und Eingaben in Folgezustände und Ausgaben. Ein Paar bestehend aus Zustand und Eingabe sei kurz als Eingabezustand bezeichnet und ein Paar bestehend aus Zustand und Ausgabe als Ausgabezustand. Der Vorbereich einer Operation gibt an zu welchen Eingabezuständen die Operation Ausgabezustände berechnet. Anstatt do wird nachfolgend die totalisierte Operation $do^\bullet$ mit $do \subseteq do^\bullet$ betrachtet. Die Operation $do^\bullet$ modelliert das chaotische Verhalten von do, wenn sie auf einen Eingabezustand angewandt wird für den do keinen Zustandsübergang enthält. Wird eine totalisierte Operation $do^\bullet$ auf einen Eingabezustand st angewandt, der nicht in dom do enthalten ist, dann ordnet $do^\bullet$ dem Eingabezustand st nichtdeterministisch einen beliebigen Ausgabezustand $x \in ds$ oder $\perp$ zu. Ist st in dom do enthalten, dann verhält sich $do^\bullet$ genauso wie do. Der spezielle Zustand $\perp$ dient zur Kennzeichnung undefinierter Zustände.

Ist M eine Menge, so ist $M^\perp == M \cup \{\perp\}$ die um den undefinierten Zustand erweiterte Menge. Sei $do_k : ds \times inp_k \leftrightarrow ds \times outp_k$ eine Operation eines Datentypen $dt = (ds, di, df, \{do_1, \ldots, do_n\})$, dann ist die Totalisierung von do_k, di und df definiert durch

$$do_k^\bullet == do_k \cup (\overline{\text{dom } do_k}^\perp \times (ds \times outp_k)^\perp),$$
$$di^\bullet == di \cup (\overline{\text{dom } di}^\perp \times ds^\perp),$$
$$df^\bullet == df \cup (\overline{\text{dom } df}^\perp \times Void^\perp).$$

Bezeichne weiterhin $dt^\bullet$ den Datentypen, der entsteht, indem alle Operationen, die Initialisierung und die Finalisierung des Datentypen $dt = (ds, di, df, \{do_1, \ldots, do_n\})$ totalisiert werden, dh.

$$dt^\bullet = (ds, di^\bullet, df^\bullet, \{do_1^\bullet, \ldots, do_n^\bullet\}).$$

Die Menge aller Kombinationen von Eingaben und Ausgaben der Operationen $do_k : ds \times inp_k \leftrightarrow ds \times outp_k$ eines Datentyps $dt = (ds, di, df, \{do_1, \ldots, do_n\})$ wird durch die Menge *InOut* beschrieben:

$$InOut == \{\exists\, k : 1 \,..\, n;\ v : inp_k;\ w : outp_k \bullet (k, v, w)\}.$$

Definition 2.10 *Eine Sequenz* $\mathbf{P} \in seq\, InOut$ *heißt* Programm. *Ein* Programm $\mathbf{P}$ über einem Datentypen dt – $\mathbf{P}(dt)$ – *ist die relationale Komposition der durch das Programm* $\mathbf{P} = \langle (k_1, v_1, w_1), (k_2, v_2, w_2), \ldots, (k_m, v_m, w_m)\rangle$ *gegebenen Sequenz von Operationen von* dt

$$\mathbf{P}(dt) = do_{k_1}[v_1/inp_{k_1}, w_1/outp_{k_1}] \fatsemi \ldots \fatsemi do_{k_m}[v_m/inp_{k_m}, w_m/outp_{k_m}]$$

□

Ein Programm $\mathbf{P}$ ist eine Sequenz S von Elementen von *InOut*. In dem Programm $\mathbf{P}(dt)$ wird jedes Element (k, v, w) der Sequenz S durch die Operation

$$do_k[v/inp_k, w/outp_k]$$

ersetzt und dann die relationale Komposition dieser Operationen in der von S vorgegebenen Reihenfolge durchgeführt.

Die *Vervollständigung* eines Programms $\mathbf{P}(dt^\bullet)$ durch Voranstellen der Initialisierung und Anhängen der Finalisierung liefert

$$di^\bullet \fatsemi \mathbf{P}(dt^\bullet) \fatsemi df^\bullet \subseteq \{void \mapsto void, void \mapsto \bot, \bot \mapsto void, \bot \mapsto \bot\}.$$

Definition 2.11 (Datenverfeinerung) *Ein initialisierbarer Datentyp* $ct = (cs, ci, cf, c)$ datenverfeinert *den initialisierbaren Datentypen* $at = (as, ai, af, a)$ *genau dann, wenn für alle Programme* $\mathbf{P}$ *gilt:*

$$ci^\bullet \fatsemi \mathbf{P}(ct^\bullet) \fatsemi cf^\bullet \subseteq ai^\bullet \fatsemi \mathbf{P}(at^\bullet) \fatsemi af^\bullet$$

□

Ein initialisierbarer Datentyp $ct = (cs, ci, cf, \{co_1, \ldots, co_n\})$ datenverfeinert den initialisierbaren Datentypen $at = (as, ai, af, \{ao_1, \ldots, ao_n\})$, wenn alle Berechnungen eines Programms $ci^\bullet \fatsemi \mathbf{P}(ct^\bullet) \fatsemi cf^\bullet$, das den Datentyp $ct^\bullet$ verwendet, auch mit dem Programm $ai^\bullet \fatsemi \mathbf{P}(at^\bullet) \fatsemi af^\bullet$ möglich sind, in dem Datentyp $ct^\bullet$ durch den Datentypen $at^\bullet$ und ersetzt ist. Das Programm $\mathbf{P}(at)$ kann also das Verhalten von $\mathbf{P}(ct)$ simulieren.

Sind $ct = (cs, ci, cf, \{co_1, \ldots, co_n\})$ und $at = (as, ai, af, \{ao_1, \ldots, ao_n\})$ zwei Datentypen, so ergibt sich mit Definition 2.10 die folgende Charakterisierung von Definition 2.11.

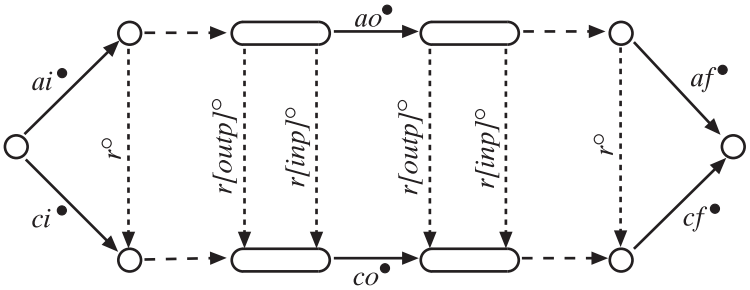


Abbildung 2.1: Vorwärtssimulation

Der Datentyp *ct* datenverfeinert den Datentyp *at* genau dann, wenn

$$\begin{aligned}
&\forall \langle (k_1, v_1, w_1), \ldots, (k_m, v_m, w_m) \rangle : \operatorname{seq} InOut \bullet \\
&\quad ci^\bullet \fatsemi co^\bullet_{k_1}[v_1/inp_{k_1}, w_1/outp_{k_1}] \fatsemi co^\bullet_{k_2}[v_2/inp_{k_2}, w_2/outp_{k_2}] \fatsemi \ldots \\
&\qquad \fatsemi co^\bullet_{k_m}[v_m/inp_{k_m}, w_m/outp_{k_m}] \fatsemi cf^\bullet \\
&\qquad\qquad \subseteq \qquad (2.2) \\
&\quad ai^\bullet \fatsemi ao^\bullet_{k_1}[v_1/inp_{k_1}, w_1/outp_{k_1}] \fatsemi ao^\bullet_{k_2}[v_2/inp_{k_2}, w_2/outp_{k_2}] \fatsemi \ldots \\
&\qquad \fatsemi ao^\bullet_{k_m}[v_m/inp_{k_m}, w_m/outp_{k_m}] \fatsemi af^\bullet
\end{aligned}$$

Die Inklusion (2.2) vergleicht Sequenzen von Operationen ohne die Zwischenzustände zu berücksichtigen. Diese Methode eignet sich nicht, um eine Verfeinerung zu beweisen. Stattdessen werden nun die Zustände beider Datentypen während einer Berechnung mit derselben Sequenz von Operationen, Eingaben und Ausgaben betrachtet.

Die Zustände von *at* und *ct* werden mittels einer Repräsentationsrelation verglichen. Diese Relation kann vom Typ $as \leftrightarrow cs$ oder $cs \leftrightarrow as$ sein. Diese beiden Typen liefern zwei verschiedene Simulationen.

Eine Repräsentationsrelation $r : as \leftrightarrow cs$ wird *Vorwärtssimulation* genannt, wenn

$$ci^\bullet \subseteq ai^\bullet \fatsemi r^\circ \qquad (2.3)$$

$$\forall\, k : 1 \,..\, n \bullet \quad r[inp_k]^\circ \fatsemi co^\bullet_k \subseteq ao^\bullet_k \fatsemi r[outp_k]^\circ \qquad (2.4)$$

$$r^\circ \fatsemi cf^\bullet \subseteq af^\bullet \qquad (2.5)$$

Die *erweiterten* Relationen

$$\begin{aligned}
&r^\circ == r \cup (\{\bot\} \times ds^\bot), \\
&r[inp_k]^\circ == r \cup (\{\bot\} \times (ds \times inp_k)^\bot), \text{ und} \\
&r[outp_k]^\circ == r \cup (\{\bot\} \times (ds \times outp_k)^\bot)
\end{aligned}$$

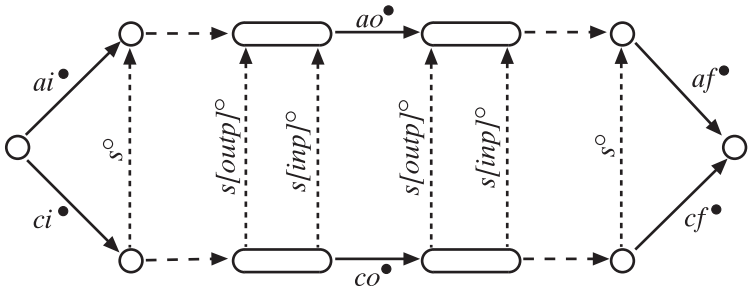


Abbildung 2.2: Rückwärtssimulation

entsprechen der Relation r, sind aber so erweitert, daß sie undefinierte Zustände behandeln wie die totalisierten Operationen. Dadurch wird das chaotische Verhalten des einen Datentypen bei dem anderen sichtbar.

Abbildung 2.1 stellt dar, wie eine Vorwärtssimulation die Zustände des abstrakten Datentypen *at* und des konkreten Datentypen *ct* miteinander in Beziehung setzt. Es wird vorausschauend überprüft, ob jede Ausführung einer Operation des konkreten Datentyps von der entsprechenden Operation des abstrakten Datentyps nachvollzogen werden kann. Befinden sich *at* und *ct* in entsprechenden Zuständen und kann *ct* die Operation co_k ausführen, dann kann *at* die Operation ao_k ausführen und befindet sich hinterher in einem Zustand, der dem von *ct* nach Ausführung von co_k entspricht.

Eine Repräsentationsrelation $s : cs \leftrightarrow as$ heißt *Rückwärtssimulation*, wenn

$$ci^\bullet \fatsemi s^\circ \subseteq ai^\bullet \tag{2.6}$$

$$\forall\, k : 1 \mathinner{..} n \bullet \quad co_k^\bullet \fatsemi s[outp_k]^\circ \subseteq s[inp_k]^\circ \fatsemi ao_k^\bullet \tag{2.7}$$

$$cf^\bullet \subseteq s^\circ \fatsemi af^\bullet \tag{2.8}$$

Abbildung 2.2 zeigt wie abstrakte Datentypen *at* und konkrete Datentypen *ct* durch eine Rückwärtssimulation verglichen werden. Es wird rückblickend überprüft, ob zu jedem Zustand von *ct*, der durch Ausführung einer konkreten Operation erreicht werden konnte, ein entsprechender Zustand von *at* durch Ausführung der entsprechenden abstrakten Operation erreicht werden könnte. Befinden sich *at* und *ct* in einander entsprechenden Zuständen und hat *ct* die Operation co_k ausgeführt, dann konnte auch die Operation ao_k von *at* ausgeführt werden, und *at* befand sich vor Ausführung von ao_k in einem Zustand, der dem von *ct* vor Ausführung von co_k entspricht.

Bemerkung 2.12 *Sind at und ct zwei Datentypen, und gibt es eine Forwärtsimulation $r : as \leftrightarrow cs$ oder eine Rückwärtssimulation $s : cs \leftrightarrow as$,*

dann wird at durch ct datenverfeinert. □

Die Inklusionen (2.3)-(2.8) lassen sich ohne totalisierte Operationen und erweiterte Repräsentationsrelation schreiben. Das führt zu einer Darstellung, die den Korrektheitsbeweis der Kalkülregeln für die Datenverfeinerung von Z-Datentypen vereinfacht. Außerdem stellt sich heraus, daß die Simulationsbedingungen für die Finalisierungen redundant sind.

In den beiden folgenden Charakterisierungen seien $ct = (cs, ci, cf, \{co_1, \ldots, co_n\})$ und $at = (as, ai, af, \{ao_1, \ldots, ao_n\})$ initialisierbare Datentypen.

Charakterisierung 2.13 (Vorwärtssimulation)
Eine Repräsentationsrelation $r : as \leftrightarrow cs$ ist eine Vorwärtssimulation genau dann, wenn

1. $ci \subseteq ai \fatsemi r$
2. $\forall k : 1 \mathinner{..} n \bullet \operatorname{ran}(\operatorname{dom} ao_k \lhd r[inp_k]) \subseteq \operatorname{dom} co_k$
3. $\forall k : 1 \mathinner{..} n \bullet (\operatorname{dom} ao_k \lhd r[inp_k]) \fatsemi co_k \subseteq ao_k \fatsemi r[outp_k]$

Beweis. Zur 1. Behauptung:

$$
\begin{array}{ll}
 & ci^\bullet \subseteq ai^\bullet \fatsemi r^\circ \\
\Leftrightarrow & ci \subseteq ai^\bullet \fatsemi r^\circ \wedge \overline{\operatorname{dom} ci}^{\perp} \times ds^{\perp} \subseteq ai^\bullet \fatsemi r^\circ \\
 & \text{[Definition von } ci^\bullet\text{]} \\
\Leftrightarrow & ci \subseteq ai \fatsemi r \wedge \overline{\operatorname{dom} ci}^{\perp} \times ds^{\perp} \subseteq ai^\bullet \fatsemi r^\circ \\
 & \text{[da } \perp \notin \operatorname{dom} ci\text{, und } at \text{ initialisierbar, also } ai \text{ total]} \\
\Leftrightarrow & ci \subseteq ai \fatsemi r \wedge \{\perp\} \times ds^{\perp} \subseteq ai^\bullet \fatsemi r^\circ \\
 & \text{[da auch } ci \text{ total]} \\
\Leftrightarrow & ci \subseteq ai \fatsemi r \\
 & \text{[da } \perp \in \operatorname{ran} ai^\bullet\text{, und nach Definition von } r^\circ\text{]}
\end{array}
$$

Zur 2. und 3. Behauptung: Zur Vereinfachung werden die Indizes der Operationen weggelassen. Sei $co : cs \times inp \leftrightarrow cs \times outp$ eine Operation von ct und $ao : as \times inp \leftrightarrow as \times outp$ die co entsprechende Operation von at. Sei $p == \operatorname{dom} ao$. Die Menge p enthält die Elemente von $(as \times inp)$, für die ao einen Zustandsübergang enthält.

$$
\begin{array}{ll}
 & r[inp]^\circ \fatsemi co^\bullet \subseteq ao^\bullet \fatsemi r[outp]^\circ \\
\Leftrightarrow & p \lhd (r[inp]^\circ \fatsemi co^\bullet) \subseteq ao \fatsemi r[outp] \\
 & \text{[da } \perp \notin \operatorname{dom} ao\text{]} \\
\Leftrightarrow & (p \lhd r[inp]^\circ) \fatsemi co^\bullet \subseteq ao \fatsemi r[outp]
\end{array}
$$

$$
\begin{array}{ll}
 & \text{[Eigenschaft von } \lhd \text{ und } \fatsemi\text{]} \\
\Leftrightarrow & (p \lhd r[inp]) \fatsemi co^\bullet \subseteq ao \fatsemi r[outp] \\
 & \text{[da } \bot \notin \operatorname{dom} ao\text{]} \\
\Leftrightarrow & (p \lhd r[inp]) \fatsemi (co \cup \overline{\operatorname{dom} co}^\bot \times (cs \times outp)^\bot) \subseteq ao \fatsemi r[outp] \\
 & \text{[Definition der Totalisierung]} \\
\Leftrightarrow & \quad (p \lhd r[inp]) \fatsemi co \subseteq ao \fatsemi r[outp] \qquad (2.9) \\
 & \wedge\ (p \lhd r[inp]) \fatsemi (\overline{\operatorname{dom} co}^\bot \times (cs \times outp)^\bot) \subseteq ao \fatsemi r[outp] \qquad (2.10) \\
 & \text{[Eigenschaft von } \fatsemi, \cup \text{ und } \subseteq\text{]}
\end{array}
$$

Wegen (2.9) ist Behauptung 3 bewiesen. Behauptung 2 ist äquivalent zu (2.10), denn:

$$
\begin{array}{ll}
 & (p \lhd r[inp]) \fatsemi (\overline{\operatorname{dom} co}^\bot \times (cs \times outp)^\bot) \subseteq ao \fatsemi r[outp] \\
\Leftrightarrow & p \lhd (r[inp] \fatsemi (\overline{\operatorname{dom} co}^\bot \times (cs \times outp)^\bot)) = \varnothing \\
 & \text{[da } \bot \notin \operatorname{ran} r[outp]\text{, und Eigenschaft von } \lhd \text{ und } \fatsemi\text{]} \\
\Leftrightarrow & \operatorname{ran}(\operatorname{dom} ao \lhd r[inp]) \subseteq \operatorname{dom} co \\
 & \text{[Definition von } \lhd\text{]}
\end{array}
$$

Die Simulationsbedingung für die Finalisierungen (2.5) ist äquivalent zu true.

$$
\begin{array}{ll}
 & r^\circ \fatsemi cf^\bullet \subseteq af^\bullet \\
\Leftrightarrow & r \fatsemi cf^\bullet \subseteq af \\
 & \text{[da ran } af = \{void\} = \operatorname{ran} cf\text{, und } af \text{ total]} \\
\Leftrightarrow & r \fatsemi cf \subseteq af \\
 & \text{[da } \bot \notin \operatorname{ran} r\text{, und } cf \text{ total]} \\
\Leftrightarrow & \text{true} \\
 & \text{[da } af \text{ total]}
\end{array}
$$

□

Charakterisierung 2.14 (Rückwärtssimulation)
Eine Repräsentationsrelation $s : cs \leftrightarrow as$ ist eine Rückwärtssimulation genau dann, wenn

1. $ci \fatsemi s \subseteq ai$
2. $\forall k : 1 \mathinner{..} n \bullet$
   $(cs \times inp_k) \subseteq \operatorname{dom}(s[inp_k] \rhd (\operatorname{dom} ao_k)) \cup \operatorname{dom}(\operatorname{dom} co_k \lhd s[inp_k])$
3. $\forall k : 1 \mathinner{..} n \bullet \operatorname{dom}(s[inp_k] \rhd (\operatorname{dom} ao_k)) \lhd (co_k \fatsemi s[outp_k]) \subseteq s \fatsemi ao_k$

Beweis. Zur 1. Behauptung:

$$
\begin{array}{ll}
 & ci^\bullet \fatsemi s^\circ \subseteq ai^\bullet \\
\Leftrightarrow & ci \fatsemi s^\circ \subseteq ai^\bullet \wedge (\overline{\mathrm{dom}\, ci}^\perp \times ds^\perp \fatsemi s^\circ) \subseteq ai^\bullet \\
 & \text{[Definition von } ci^\bullet\text{]} \\
\Leftrightarrow & ci \fatsemi s^\circ \subseteq ai^\bullet \\
 & \text{[da } ci \text{ total]} \\
\Leftrightarrow & ci \fatsemi s \subseteq ai \\
 & \text{[da } ai \text{ total, und } \perp \notin \mathrm{ran}\, ci\text{]}
\end{array}
$$

Zum Beweis der 2. und 3. Behauptung: Sei $co : cs \times inp \leftrightarrow cs \times outp$ eine Operation von ct und $ao : as \times inp \leftrightarrow as \times outp$ die co entsprechende Operation von at. Sei $q == \mathrm{dom}(s[inp] \rhd (\mathrm{dom}\, ao))$. Die Menge $q \subseteq (cs \times inp)$ enthält die Zustände von ct, für die ao auf den entsprechenden Zuständen nicht definiert ist.

$$
\begin{array}{ll}
 & co^\bullet \fatsemi s[outp]^\circ \subseteq s[inp]^\circ \fatsemi ao^\bullet \\
\Leftrightarrow & \quad q \lhd (co^\bullet \fatsemi s[outp]^\circ) \subseteq s[inp] \fatsemi ao \\
 & \wedge\ (cs \times inp) \subseteq \mathrm{dom}\, s[inp] \qquad (2.11) \\
 & \text{[da } \mathrm{ran}(q \rhd s[inp]) \subseteq \mathrm{dom}\, ao\text{]} \\
\Leftrightarrow & \quad (q \lhd (co \cup (\overline{\mathrm{dom}\, co}^\perp \times (cs \times outp)^\perp))) \fatsemi s[outp]^\circ \subseteq s[inp] \fatsemi ao \\
 & \wedge\ (cs \times inp) \subseteq \mathrm{dom}\, s[inp] \\
 & \text{[Eigenschaft von } \fatsemi \text{ und } \lhd\text{, und Definition der Totalisierung]} \\
\Leftrightarrow & \quad (q \lhd co) \fatsemi s[outp] \subseteq s[inp] \fatsemi ao \qquad (2.12) \\
 & \wedge\ (q \lhd (\overline{\mathrm{dom}\, co}^\perp \times (cs \times outp)^\perp) \fatsemi s[outp]^\circ \subseteq s[inp] \fatsemi ao \qquad (2.13) \\
 & \wedge\ (cs \times inp) \subseteq \mathrm{dom}\, s[inp] \qquad (2.14)
\end{array}
$$

Wegen (2.12) ist die 3. Behauptung wahr. Die 2. Behauptung ist äquivalent zu (2.13) $\wedge$ (2.14).

$$
\begin{array}{ll}
 & (q \lhd (\overline{\mathrm{dom}\, co}^\perp \times (cs \times outp)^\perp) \fatsemi s[outp]^\circ \subseteq s[inp] \fatsemi ao \\
 & \wedge\ (cs \times inp) \subseteq \mathrm{dom}\, s[inp] \\
\Leftrightarrow & \quad q \lhd ((\overline{\mathrm{dom}\, co}^\perp \times (cs \times outp)^\perp \fatsemi s[outp]^\circ) = \varnothing \\
 & \wedge\ (cs \times inp) \subseteq \mathrm{dom}\, s[inp] \\
 & \text{[da } \perp \notin \mathrm{ran}\, ao\text{, und Eigenschaft von } \lhd \text{ und } \fatsemi\text{]} \\
\Leftrightarrow & \overline{\mathrm{dom}\, co} \subseteq \mathrm{dom}(s[inp] \rhd (\mathrm{dom}\, ao)) \wedge (cs \times inp) \subseteq \mathrm{dom}\, s[inp] \\
 & \text{[Definition von } \rhd\text{]}
\end{array}
$$

$$\begin{array}{ll}
\Leftrightarrow & \overline{\mathrm{dom}\ co} \subseteq \mathrm{dom}(s[inp] \rhd (\mathrm{dom}\ ao)) \wedge (\mathrm{dom}\ co \cup \overline{\mathrm{dom}\ co}) \subseteq \mathrm{dom}\ s[inp] \\
& \left[\mathrm{da}\ (cs \times inp) = \mathrm{dom}\ co \cup \overline{\mathrm{dom}\ co}\right] \\
\Leftrightarrow & \overline{\mathrm{dom}\ co} \subseteq \mathrm{dom}(s[inp] \rhd (\mathrm{dom}\ ao)) \wedge \mathrm{dom}\ co \subseteq \mathrm{dom}(\mathrm{dom}\ co \lhd s[inp]) \\
\Leftrightarrow & (cs \times inp) \subseteq \mathrm{dom}(s[inp] \rhd (\mathrm{dom}\ ao)) \cup \mathrm{dom}(\mathrm{dom}\ co \lhd s[inp]) \\
& \left[\mathrm{da}\ \overline{\mathrm{dom}\ co} \cap \mathrm{dom}(\mathrm{dom}\ co \lhd s[inp]) = \varnothing\right]
\end{array}$$

Die Simulationsbedingung für die Finalisierung (2.8) ist äquivalent zu true.

$$\begin{array}{ll}
& cf^{\bullet} \subseteq s^{\circ} \fatsemi af^{\bullet} \\
\Leftrightarrow & cf \subseteq s \fatsemi af^{\bullet} \\
& [\mathrm{da}\ cs \subseteq \mathrm{dom}\ s \text{ nach (2.11), also } \mathrm{dom}\ cf = \mathrm{dom}\ s, \mathrm{da}\ cf \text{ total}] \\
\Leftrightarrow & cf \subseteq s \fatsemi af \\
& [\mathrm{da}\ af \text{ total}] \\
\Leftrightarrow & \mathrm{true} \\
& [\mathrm{da}\ cs \subseteq \mathrm{dom}\ s, \text{ und } \mathrm{ran}(s \fatsemi af) = \{void\} = \mathrm{ran}\ cf]
\end{array}$$

□

Die Finalisierungen spielen bei den Simulationen keine Rolle. In [HHS87, WD96] ist zu sehen, daß das in einem allgemeineren Kontext nicht mehr der Fall ist.

2.4.2 Z-Datentypen

Die Datenverfeinerungsregeln des vorigen Abschnitts sind auch für Z-Datentypen direkt formulierbar. Dazu wird die Semantik von Z-Datentypen *Dat* betrachtet, die durch einen Datentypen $\mathcal{M}^{\mathsf{Z}}[\![Dat]\!]$ beschrieben wird, und die Simulationen werden in Form eines Z-Schemas angegeben. In Unterabschnitt 2.2.2 wurde gezeigt wie Operationsschemata mit beliebig vielen Eingaben und Ausgaben zu Operationsschemata mit einer Eingabe und einer Ausgabe vereinfacht werden können. Die Semantik von Z-Datentypen *Dat* ist nur für solche *Dat* definiert, deren Operationsschemata die einfache Form mit einer Eingabe und einer Ausgabe haben. Dadurch ist eine direkte Abbildung der Z-Datentypen in Datentypen möglich.

Die Zustände, die ein Z-Datentyp *Dat* annehmen kann, werden durch sein Zustandsschema *State* beschrieben. Die Zustände selbst sind Bindungen $\Theta State$ des Zustandsschemas. Die Zustandskomponente von $\mathcal{M}^{\mathsf{Z}}[\![Dat]\!]$ enthält dementsprechend alle Bindungen $\Theta State$. Die Operationsschemata Op_k von *Dat* mit einer Eingabe und einer Ausgabe besitzen eine offensichtliche Interpretation als relationale Operationen do_k von $\mathcal{M}^{\mathsf{Z}}[\![Dat]\!]$.

Definition 2.15 *Sei Dat ein* Z*-Datentyp mit Zustandsschema State, Initialisierungsschema InitState und Operationsschemata* $Op_1, \ldots, Op_n$ *mit*

Op_k

$\Delta State$
$i^k? : type_i^k$
$o^k! : type_o^k$

---

p_{op_k}

Die Semantik des Z*-Datentypen Dat ist ein Datentyp* $dt = (ds, di, df, \{do_1, \ldots, do_n)$.

$$\mathcal{M}^{\mathsf{Z}}[\![Dat]\!] = dt$$

wobei

$$ds = \mathcal{S}^{\mathsf{Z}}[\![State]\!] == \{State \bullet \Theta State\},$$
$$di = \mathcal{I}^{\mathsf{Z}}[\![InitState]\!] == \{Init' \bullet void \mapsto \Theta State'\},$$
$$df = \mathcal{T}^{\mathsf{Z}}[\![State]\!] == \{State \bullet \Theta State \mapsto void\},$$
$$\forall\, k \in 1 \,..\, n \bullet$$
$$do_k = \mathcal{O}^{\mathsf{Z}}[\![Op_k]\!] == \{Op_k \bullet (\Theta State, i^k?) \mapsto (\Theta State', o^k!)\}$$

□

In den Betrachtungen dieses Unterabschnitts wird davon ausgegangen, daß die Operationsschemata des Z-Datentyps in geeigneter Form numeriert sind. Dann ist die semantische Abbildung auf einen Datentypen eindeutig.

Die Verfeinerungstheorie für Datentypen setzt ihre Initialisierbarkeit voraus. Das folgende Lemma zeigt, daß die Initialisierbarkeit der Semantik $\mathcal{M}^{\mathsf{Z}}[\![Dat]\!]$ eines Datentypen *Dat* mit Zustandsschema *State* und Initialisierungsschema *InitState* gleichwertig mit dem Initialisierungstheorem ist.

Lemma 2.16 $\mathcal{M}^{\mathsf{Z}}[\![Dat]\!]$ *ist initialisierbar genau dann, wenn Dat das Initialierungstheorem erfüllt:* $\exists\, State' \bullet InitState$. □

Seien *Abs* und *Con* zwei Z-Datentypen, die das Initialisierungstheorem erfüllen. Seien *AbsState* und *AbsInit* das Zustandsschema und das Initialierungsschemata von *Abs* und *ConState* und *ConInit* die entsprechenden Schemata von *Con*. Die Anzahl n der Operationsschemata beider Datentypen stimme überein.

$AbsOp_k$

$\Delta AbsState$
$i^k? : type_i^k$
$o^k! : type_o^k$

---

p_{AbsOp_k}

$$
\begin{array}{|l}
\hline
ConOp_k \\
\Delta ConState \\
i^k? : type_i^k \\
o^k! : type_o^k \\
\hline
p_{ConOp_k} \\
\hline
\end{array}
$$

Es ist zu beachten, daß die Typen der Eingaben und Ausgaben von Operationen mit gleichem Index identisch sind. (Datenverfeinerung betrifft nur das Zustandsschema von *Abs* und *Con*.)

Ziel ist nun eine Simulationsbeziehung zwischen den beiden Z-Datentypen *Abs* und *Con* so zu formulieren, daß wenn *Abs* durch *Con* datenverfeinert wird, auch $\mathcal{M}^{\mathsf{Z}}[\![Abs]\!]$ durch $\mathcal{M}^{\mathsf{Z}}[\![Con]\!]$ datenverfeinert wird. Eine Datenverfeinerung von $\mathcal{M}^{\mathsf{Z}}[\![Abs]\!]$ durch $\mathcal{M}^{\mathsf{Z}}[\![Con]\!]$ besteht, wenn es eine Repräsentationsrelation gibt, die eine Vorwärtssimulation oder eine Rückwärtssimulation von $\mathcal{M}^{\mathsf{Z}}[\![Con]\!]$ durch $\mathcal{M}^{\mathsf{Z}}[\![Abs]\!]$ etabliert. Die Repräsentationsrelation vergleicht die Zustände $\mathcal{S}^{\mathsf{Z}}[\![ConState]\!]$ und $\mathcal{S}^{\mathsf{Z}}[\![AbsState]\!]$. Es liegt nahe, ein Schema zu definieren, daß *AbsState* und *ConState* miteinander in Beziehung setzt.

$$
\begin{array}{|l}
\hline
Rep \\
AbsState \\
ConState \\
\hline
p_{Rep} \\
\hline
\end{array}
$$

Das Schema *Rep* heißt *Repräsentationsschema*. Es gibt zwei Möglichkeiten *Rep* als Repräsentationsrelation darzustellen. Die Semantik eines Repräsentationsschemas wird durch beide Relationen definiert.

$$\mathcal{R}^{\mathsf{Z}}[\![Rep]\!] == (r, s) \quad \text{wobei}$$

$$r == \{Rep \bullet \Theta AbsState \mapsto \Theta ConState\}$$
$$s == \{Rep \bullet \Theta ConState \mapsto \Theta AbsState\}$$

Das Schema *Rep* kann eine Vorwärtssimulation r oder eine Rückwärtssimulation s (oder beide) induzieren. Die beiden folgenden Bemerkungen nennen die Bedingungen, die erfüllt sein müssen, damit die Semantik $\mathcal{R}^{\mathsf{Z}}[\![Rep]\!]$ eines Repräsentationsschemas *Rep* eine Simulation ist. In [WD96] wird der Beweis der ersten Bemerkung teilweise ausgeführt.

Bemerkung 2.17 (Vorwärtssimulation) *Sei Rep ein Repräsentationsschema und es gelte*

1. $\forall ConState' \bullet \exists AbsState' \bullet$
   $ConInit \Rightarrow AbsInit \wedge Rep'$

2. Für alle k gilt:
   $\forall\, ConState;\ AbsState;\ i^k? : type_i^k \bullet$
   $pre\, AbsOp_k \land Rep \Rightarrow pre\, ConOp_k$

3. Für alle k gilt:
   $\forall\, \Delta ConState;\ AbsState;\ i^k? : type_i^k;\ o^k! : type_o^k \bullet$
   $pre\, AbsOp_k \land Rep \land ConOp_k \Rightarrow (\exists\, AbsState' \bullet AbsOp_k \land Rep')$

Dann wird Abs durch Con datenverfeinert.

Beweisskizze. Sei $\mathcal{M}^{\mathsf{Z}}[\![Abs]\!] == \{as, ai, af, \{ao_1, \ldots, ao_n\}\}$, $\mathcal{M}^{\mathsf{Z}}[\![Con]\!] == \{cs, ci, cf, \{co_1, \ldots, co_n\}\}$ und *Rep* ein Repräsentationsschema mit $\mathcal{R}^{\mathsf{Z}}[\![Rep]\!] == \{r, s\}$. Mit Charakterisierung 2.13 ergeben sich folgende Äquivalenzen:

2.13.1 $\Leftrightarrow$ 1.
2.13.2 $\Leftrightarrow$ 2.
2.13.3 $\Leftrightarrow$ 3.

□

Informell bedeutet Aussage 2.17.1, daß es zu jedem Anfangszustand von *Con* einen entsprechenden Anfangszustand von *Abs* gibt. Aussage 2.17.2 besagt, daß wenn *Abs* und *Con* sich in einander entsprechenden Zuständen befinden und die Operation $AbsOp_k$ ausgeführt werden kann, dann kann die ihr entsprechende Operation $ConOp_k$ ausgeführt werden. Die Operation $ConOp_k$ ist also bei mehr oder gleichvielen Zuständen wie $AbsOp_k$ ausführbar. Die letzte Aussage 2.17.3 drückt aus, daß, wenn *Abs* und *Con* sich in einander entsprechenden Zuständen befinden; und wenn die Operation $ConOp_k$ und die Operation $AbsOp_k$ ausgeführt werden können, dann kann $AbsOp_k$ einen Zustand berechnen, der dem von $ConOp_k$ berechneten entspricht.

Bemerkung 2.18 (Rückwärtssimulation) *Sei Rep ein Repräsentationsschema und es gelte*

1. $\forall\, ConState';\ AbsState' \bullet$
   $ConInit \land Rep' \Rightarrow AbsInit$

2. *Für alle k gilt:*
   $\forall\, ConState;\ i^k? : type_i^k \bullet \exists\, AbsState \bullet$
   $Rep \land (\text{pre}\, AbsOp_k \Rightarrow \text{pre}\, ConOp_k)$

3. *Für alle k gilt:*
   $\forall\, \Delta ConState;\ AbsState';\ i^k? : type_i^k;\ o^k! : type_o^k \bullet$
   $(\forall\, AbsState \bullet Rep \Rightarrow \text{pre}\, AbsOp_k) \land ConOp_k \land Rep' \Rightarrow$
   $(\exists\, AbsState \bullet Rep \land AbsOp_k)$

Dann wird Abs durch Con datenverfeinert.

Beweisskizze. Sei $\mathcal{M}^{\mathsf{Z}}[\![Abs]\!] == \{as, ai, af, \{ao_1, \ldots, ao_n\}\}$, $\mathcal{M}^{\mathsf{Z}}[\![Con]\!] == \{cs, ci, cf, \{co_1, \ldots, co_n\}\}$ und *Rep* ein Repräsentationsschema mit $\mathcal{R}^{\mathsf{Z}}[\![Rep]\!] == \{r, s\}$. Mit Charakterisierung 2.14 ergeben sich folgende Äquivalenzen:

2.14.1 $\Leftrightarrow$ 1.
2.14.2 $\Leftrightarrow$ 2.
2.14.3 $\Leftrightarrow$ 3.

□

Aussage 2.18.1 besagt, daß wenn *Abs* und *Con* sich in einander entsprechenden Zuständen befinden und *Con* sich in einem Anfangszustand befindet, dann kann sich auch *Abs* in einem Anfangszustand befinden. Die zweite Aussage 2.18.2 fordert, daß es zu jedem Zustand von *Con* einen entsprechenden Zustand von *Abs* gibt; und wenn in diesem Zustand $AbsOp_k$ ausgeführt werden kann, dann kann auch $ConOp_k$ in dem entsprechenden Zustand ausgeführt werden. Die dritte Aussage 2.18.3 besagt, daß wenn *Abs* und *Con* sich in einander entsprechenden Zuständen befinden und eine Operation $ConOp_k$ ausgeführt werden konnte, dann konnte die entsprechende Operation $AbsOp_k$ ausgeführt werden und *Abs* und *Con* befanden sich vor der Ausführung der Operationen in sich entsprechenden Zuständen. Durch die Teilbedingung $\forall\, AbsState \bullet Rep \Rightarrow \text{pre}\, AbsOp_k$ werden nur diejenigen Zustände *ConState* betrachtet, deren sämtliche Entsprechungen *AbsState* eine Ausführung von $AbsOp_k$ ermöglichen.

2.4.3 Anmerkungen

Die in [WD96] angegebene Beweisregel für die Rückwärtssimulation ist nicht korrekt. Dort wird behauptet ein Repräsentationsschema *Rep* etabliert eine Rückwärtssimulation eines Z-Datentyps *Con* durch einen Z-Datentypen *Abs*, wenn die Bedingungen 2.18.1, 2.18.3 und die Bedingung

für alle k :
$\forall\, ConState;\ i^k? : type_i^k \bullet$
$(\forall\, AbsState \bullet Rep \Rightarrow \text{pre}\, AbsOp_k) \Rightarrow \text{pre}\, ConOp_k)$

erfüllt sind. Das Repräsentationsschema

Rep

AbsState *ConState*
false

erfüllt diese Bedingungen für alle Z-Datentypen *Abs* und *Con*, falls pre $ConOp_k \equiv$ true für alle k. Es ist leicht einzusehen, daß diese Regel nicht korrekt sein kann.

In [HHS87] wird Datenverfeinerung von Datentypen für eine größere Klasse von Programmen behandelt. Insbesondere werden dort auch rekursive Programme einbezogen, die die Betrachtung unendlicher Operationsfolgen notwendig machen. Die Regeln der letzten beiden Abschnitte sind nicht zur Behandlung rekursiver Programme tauglich. Gardiner und Morgan zeigen in [GM89] wie Vorwärts- und Rückwärtssimulation zu einer einzigen Simulationsrelation zusammengefaßt werden können. Bei der Verwendung dieser Simulationsrelation werden jedoch die beiden Simulationsmethoden wieder unterschieden.

Die folgende Diskussion zeigt eine alternative Definition der Datenverfeinerung, die für CSP-Z geeignet ist. Die Darstellung der wichtigen Ideen erfolgt nur in Kürze, da sie im weiteren Verlauf dieser Arbeit nicht mehr von Bedeutung ist.

Die in Definition 2.11 eingeführte Datenverfeinerung wird in der folgenden Darstellung als *schwache* Datenverfeinerung referenziert, um sie von der hier definierten *strengen* Datenverfeinerung abzugrenzen. Die strenge Datenverfeinerung wird so definiert, daß eine strenge Verfeinerung eine schwache Verfeinerung impliziert. Der Unterschied zwischen beiden ist, daß bei einer schwachen Verfeinerung mehr Programme $\mathbf{P}(ct)$, die den konkreten Datentypen verwenden, korrekt sein können als Programme $\mathbf{P}(at)$, die den abstrakten Datentypen verwenden. Bei der strengen Datenverfeinerung sind alle Programme $\mathbf{P}(at)$ genau dann korrekt, wenn auch $\mathbf{P}(ct)$ korrekt ist.

Zunächst werden die strenge Totalisierung und die strenge Erweiterung einer Relation benötigt.

Definition 2.19 *Ist $\rho : X \leftrightarrow Y$ eine Relation, dann ist*

$$\rho^{\blacktriangledown} == \rho \cup \overline{dom\,\rho}^{\perp} \times \{\perp\}$$

die strenge Totalisierung von ρ und

$$\rho^{\triangledown} == \rho \cup \{\perp \mapsto \perp\}$$

die strenge Erweiterung von ρ. □

Strenge Datenverfeinerung ist analog zur Datenverfeinerung (Definition 2.11) unter Verwendung der strengen Totalisierung und der strengen Erweiterung definiert.

Definition 2.20 (strenge Datenverfeinerung) *Ein initialisierbarer Datentyp $ct = (cs, ci, cf, c)$* datenverfeinert *den initialisierbaren Datentypen $at = (as, ai, af, c)$ genau dann, wenn für alle Programme $\mathbf{P}$ gilt:*

$$ci^{\blacktriangledown} \fatsemi \mathbf{P}(ct^{\blacktriangledown}) \fatsemi cf^{\blacktriangledown} \subseteq ai^{\blacktriangledown} \fatsemi \mathbf{P}(at^{\blacktriangledown}) \fatsemi af^{\blacktriangledown}$$

□

Der Unterschied zwischen Definition 2.20 und Definition 2.11 besteht darin, daß in Definition 2.20 zusätzlich für alle Programme

$$ci \fatsemi \mathbf{P}(ct) \neq \varnothing \quad \Rightarrow \quad ai \fatsemi \mathbf{P}(at) \neq \varnothing$$

gelten muß. Es kann also nicht mehr Programme über ct geben, die etwas berechnen, als solche Programme über at.

Die Betrachtungen aus Abschnitt 2.4.1 behalten ihre Gültigkeit. Die Charakterisierungen 2.13 und 2.14 sind jedoch etwas stärker. Initialisierung und Finalisierung sind von der neuen Definition nicht betroffen, da sie totale Relationen sind. Sie werden daher nicht nochmals aufgeführt.

Charakterisierung 2.21 (strenge Vorwärtssimulation)

$$\begin{aligned} & r[inp]^{\triangledown} \fatsemi co^{\blacktriangledown} \subseteq ao^{\blacktriangledown} \fatsemi r[outp]^{\triangledown} \\ \Leftrightarrow \quad & dom\, ao \lhd (r[inp]^{\triangledown} \fatsemi co^{\blacktriangledown}) \subseteq ao \fatsemi r[outp] \\ & \wedge\ dom(r[inp] \rhd (dom\, co)) \subseteq dom\, ao \end{aligned} \tag{2.15}$$

Charakterisierung 2.22 (strenge Rückwärtssimulation)

$$\begin{aligned} & co^{\blacktriangledown} \fatsemi s[outp]^{\triangledown} \subseteq s[inp]^{\triangledown} \fatsemi ao^{\blacktriangledown} \\ \Leftrightarrow \quad & dom(s[inp] \rhd (dom\, ao)) \lhd (co^{\blacktriangledown} \fatsemi s[outp]^{\triangledown}) \subseteq s \fatsemi ao \\ & \wedge\ cs \subseteq dom\, s \\ & \wedge\ ran((dom\, co) \lhd r[inp]) \subseteq dom\, ao \end{aligned} \tag{2.16}$$

Die in Abschnitt 2.4.2 definierten semantischen Abbildungen bleiben von den neuen Definitionen unberührt.

Seien *Abs* und *Con* zwei Z-Datentypen mit Operationen *AbsOp* und *ConOp* und *Rep* ein Repräsentationsschema (mit den beiden Komponenten *AbsState* und *ConState*). Soll *Rep* eine Vorwärtsimulation r oder eine Rückwärtssimulation s induzieren, dann muß zusätzlich zu den in Bemerkung 2.17 bzw. Bemerkung 2.18 angegebenen Bedingungen die folgende erfüllt sein:

$$\forall\, ConState;\ AbsState;\ i? : type_i \bullet \mathrm{pre}\, ConOp \wedge Rep \Rightarrow \mathrm{pre}\, AbsOp \tag{2.17}$$

Aussage (2.17) ist äquivalent zu (2.15) und (2.16) und bedeutet, daß Operation *ConOp* nicht für mehr Zustände als Operation *AbsOp* definiert ist.

In Kapitel 3 wird dieses Ergebnis verwendet, um zu passenden Verfeinerungsregeln im Kontext von CSP-Z zu gelangen.

Kapitel 3

Datenverfeinerung in CSP-Z

In Abschnitt 2.4 wurde Datenverfeinerung für Datentypen definiert und Beweisregeln für die Verfeinerung von Z-Datentypen daraus abgeleitet. Diese Beweisregeln sind der Ausgangspunkt für die Formulierung von Datenverfeinerungsregeln für CSP-Z. Die neuen Beweisregeln werden in prädikativer Form angegeben und dann ihre Korrektheit in CSP-Z bewiesen.

Im nächsten Abschnitt werden Vorwärtssimulation und Rückwärtssimulation für CSP-Z definiert. Im darauf folgenden Abschnitt 3.1 wird die Korrektheit beider Regeln bewiesen. In Abschnitt 3.3 wird ein eingeschränktes Vollständigkeitsresultat für die beiden Regeln aus Abschnitt 3.1 bewiesen, das nur für CSP-Z-Spezifikationen gilt, deren CSP-Teil leer ist.

3.1 Beweisregeln

Seien $\mathtt{A} = \mathtt{spec}\,\mathtt{I};\ \mathsf{Z}^{\mathtt{A}}\,\mathtt{end\_spec}$ und $\mathtt{C} = \mathtt{spec}\,\mathtt{I};\ \mathsf{Z}^{\mathtt{C}}\,\mathtt{end\_spec}$ zwei CSP-Z-Spezifikationen mit $\mathtt{A} \sqsubseteq \mathtt{C}$. Die Regeln 2.17 und 2.18 sind nicht geeignet, um nachzuweisen, daß A durch C verfeinert wird. Durch beide Regeln werden der konkreten Spezifikation C mehr Traces ermöglicht als der abstrakten A. Dann ist aber $\mathcal{F}[\![\mathtt{C}]\!] \not\subseteq \mathcal{F}[\![\mathtt{A}]\!]$. Beide Regeln erlauben den Vorbedingungen der konkreten Operationen schwächer zu sein, als die Vorbedingungen der entsprechen abstrakten Operationen. In Abschnitt 2.4.3 wurde die strenge Datenverfeinerung definiert, die nach (2.17) genau diese Abschwächung der Vorbedingungen der abstrakten Operationen verbietet. Die Aussagen 2.17.2 und (2.17) zusammen sind äquivalent zu:

$$\forall\, \mathtt{C.State};\ \mathtt{A.State};\ @c : \mathit{type}_{\mathcal{I}}(c) \bullet \\ \mathit{Rep} \Rightarrow (\mathrm{pre}\,\mathtt{A.com\_}\,c \Leftrightarrow \mathrm{pre}\,\mathtt{C.com\_}\,c) \tag{3.1}$$

Aus 2.18.2 folgt:

$$\forall\, \mathtt{C.State};\ @c : \mathit{type}_{\mathcal{I}}(c) \bullet \\ (\forall\, \mathtt{A.State} \bullet \mathit{Rep} \Rightarrow \mathrm{pre}\,\mathtt{A.com\_}\,c) \Rightarrow \mathrm{pre}\,\mathtt{C.com\_}\,c \tag{3.2}$$

Und (3.3) zusammen mit (2.17) ist äquivalent zu:

$$\forall \mathtt{C.State};\ @c : type_{\mathcal{I}}(c) \bullet \qquad (3.3)$$
$$(\forall \mathtt{A.State} \bullet Rep \Rightarrow \mathrm{pre}\, \mathtt{A.com\_}\ c) \Leftrightarrow \mathrm{pre}\, \mathtt{C.com\_}\ c$$

Die beiden Regeln für Vorwärts- und Rückwärtssimulation unterscheiden sich nur in Bedingung 3 von den Regeln 2.17 und 2.18.

Definition 3.1 (Vorwärtssimulation "FS")
Seien $\mathtt{A} = \mathtt{spec\,I;\ P;\ Z^A\ end\_spec}$ *und* $\mathtt{C} = \mathtt{spec\,I;\ P;\ Z^C\ end\_spec}$ *zwei* CSP-Z*-Spezifikationen. Ein Repräsentationsschema Rep heißt FS-*Schema *von* A *nach* C*, wenn*

1. **(Initialisierung)**
   $\forall \mathtt{C.State'} \bullet \exists \mathtt{A.State'} \bullet$
   $\mathtt{C.Init\_State} \Rightarrow \mathtt{A.Init\_State} \wedge Rep'$

2. **(Definiertheit)**
   Für alle Kanäle $c \in Chans(\mathcal{I})$ *gilt:*
   $\forall \mathtt{C.State};\ \mathtt{A.State};\ @c : type_{\mathcal{I}}(c) \bullet$
   $\mathrm{pre}\, \mathtt{A.com\_}\ c \wedge Rep \Rightarrow \mathrm{pre}\, \mathtt{C.com\_}\ c$

3. **(Fortschritt)**
   Für alle Kanäle $c \in Chans(\mathcal{I})$ *gilt:*
   $\forall \Delta\mathtt{C.State};\ \mathtt{A.State};\ @c : type_{\mathcal{I}}(c) \bullet$
   $Rep \wedge \mathtt{C.com\_}\ c \Rightarrow (\exists \mathtt{A.State'} \bullet \mathtt{A.com\_}\ c \wedge Rep')$

Bezeichne dann $\mathtt{A} \sim_{FS} \mathtt{C}$*, daß es ein FS-Schema Rep von* A *nach* C *gibt.* □

Definition 3.2 (Rückwärtssimulation "BS")
Seien $\mathtt{A} = \mathtt{spec\,I;\ P;\ Z^A\ end\_spec}$ *und* $\mathtt{C} = \mathtt{spec\,I;\ P;\ Z^C\ end\_spec}$ *zwei* CSP-Z*-Spezifikationen. Ein Repräsentationsschema Rep heißt BS-*Schema *von* A *nach* C*, wenn*

1. **(Initialisierung)**
   $\forall \mathtt{C.State'};\ \mathtt{A.State'} \bullet$
   $\mathtt{C.Init\_State} \wedge Rep' \Rightarrow \mathtt{A.Init\_State}$

2. **(Definiertheit)**
   Für alle Kanäle $c \in Chans(\mathcal{I})$ *gilt:*
   $\forall \mathtt{C.State};\ @c : type_{\mathcal{I}}(c) \bullet \exists \mathtt{A.State} \bullet$
   $Rep \wedge (\mathrm{pre}\, \mathtt{A.com\_}\ c \Rightarrow \mathrm{pre}\, \mathtt{C.com\_}\ c)$

3. **(Fortschritt)**
   Für alle Kanäle $c \in Chans(\mathcal{I})$ *gilt:*
   $\forall \Delta\mathtt{C.State};\ \mathtt{A.State'};\ @c : type_{\mathcal{I}}(c) \bullet$
   $\mathtt{C.com\_}\ c \wedge Rep' \Rightarrow (\exists \mathtt{A.State} \bullet Rep \wedge \mathtt{A.com\_}\ c)$

Bezeichne dann $\mathtt{A} \backsim_{BS} \mathtt{C}$*, daß es ein BS-Schema Rep von* A *nach* C *gibt.* □

Zu der Bedeutung der einzelnen Bedingungen sei auf die Erläuterungen zu den Bemerkungen 2.17 und 2.18 verwiesen. Im Unterschied zu 2.17.3 und 2.18.3 muß hier, wenn eine konkrete Operation ausgeführt werden kann, auch die entsprechende abstrakte ausgeführt werden können.

3.2 Korrektheit von FS und BS

In diesem Abschnitt wird die Korrektheit der Regeln 2.17 und 2.18 bewiesen. Die Beweise bestehen jeweils aus zwei Teilen. Im vorangestellten Lemma wird durch Induktion gezeigt, daß die Failures einer konkreten Spezifikation $\mathtt{C} = \mathtt{spec\,I;\ Z^C\ end\_spec}$ in den Failures der abstrakten Spezifikation $\mathtt{A} = \mathtt{spec\,I;\ Z^A\ end\_spec}$ enthalten sind, wenn die Bedingungen der Regeln erfüllt sind.

Ist $\mathtt{A} \sqsubseteq \mathtt{C}$ bewiesen, dann überträgt sich dieses Ergebnis auf Spezifikationen, in deren Definition ein CSP-Prozeß vorkommt.

Bemerkung 3.3 *Seien* $\mathtt{spec\,I;\ P;\ Z^A\ end\_spec}$*,* $\mathtt{spec\,I;\ P;\ Z^C\ end\_spec} \in$ *SPEC. Dann gilt:*

$$\mathtt{spec\,I;\ Z^A\ end\_spec} \sqsubseteq \mathtt{spec\,I;\ Z^C\ end\_spec} \Rightarrow$$
$$\mathtt{spec\,I;\ P;\ Z^A\ end\_spec} \sqsubseteq \mathtt{spec\,I;\ P;\ Z^C\ end\_spec}$$

Beweis.

$$\begin{array}{ll} & \mathtt{spec\,I;\ P;\ Z^A\ end\_spec} \\ \equiv & \mathtt{spec\,I;\ P\ end\_spec} \parallel \mathtt{spec\,I;\ Z^A\ end\_spec} \\ & \text{[nach Definition 2.7]} \\ \sqsubseteq & \mathtt{spec\,I;\ P\ end\_spec} \parallel \mathtt{spec\,I;\ Z^C\ end\_spec} \\ & \text{[da } \parallel \text{ monoton, und Voraussetzung]} \\ \equiv & \mathtt{spec\,I;\ P;\ Z^C\ end\_spec} \end{array}$$

□

Gilt für zwei Spezifikationen $\mathtt{A} \sim_{FS} \mathtt{C}$, dann kann A alle Traces von C nachvollziehen und die Failures von A bilden eine Obermenge der Failures von C.

Lemma 3.4 *Seien* $\mathtt{A} = \mathtt{spec\,I;\ Z^A\ end\_spec}$ *und* $\mathtt{C} = \mathtt{spec\,I;\ Z^C\ end\_spec}$ *zwei Spezifikationen und Rep ein Repräsentationsschema.*

1. *Gelten 3.1.1 und 3.1.3, so auch für alle* $tr : \mathrm{seq}\, Comm(Chans(\mathcal{I}))$:

$$\forall\, \mathtt{C.State} \bullet \exists\, \mathtt{A.State} \bullet \mathtt{C}^\star_{tr} \Rightarrow \mathtt{A}^\star_{tr} \wedge Rep.$$

2. Gilt außerdem 3.1.2, dann folgt

$$\mathcal{F}[\![\mathtt{C}]\!] \subseteq \mathcal{F}[\![\mathtt{A}]\!]$$

Beweis. Behauptung 1 wird durch Induktion über die Länge der Traces bewiesen:

$$\begin{array}{ll} & \forall\, \mathtt{C.State}' \bullet \mathtt{C.Init\_State} \Rightarrow (\exists\, \mathtt{A.State}' \bullet \mathtt{A.Init\_State} \land \mathit{Rep}') \\ & [\text{nach Vorraussetzung 3.1.1}] \\ \Leftrightarrow & \forall\, \mathtt{C.State}' \bullet \mathtt{C}_{\langle\rangle} \Rightarrow (\exists\, \mathtt{A.State}' \bullet \mathtt{A}_{\langle\rangle} \land \mathit{Rep}') \\ & \left[\text{Definition von } \mathtt{C}_{\langle\rangle}\right] \\ \Leftrightarrow & \forall\, \mathtt{C.State} \bullet \mathtt{C}^{\star}_{\langle\rangle} \Rightarrow (\exists\, \mathtt{A.State} \bullet \mathtt{A}^{\star}_{\langle\rangle} \land \mathit{Rep}) \\ & [\text{Definition von } {}^{\star}] \end{array}$$

Sei nun $tr \frown \langle (c, v) \rangle : \operatorname{seq} \mathit{Comm}(\mathit{Chans}(\mathcal{I}))$.

$$\begin{array}{ll} & \forall\, \mathtt{C.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \Rightarrow (\exists\, \mathtt{C.State} \bullet \mathtt{C}^{\star}_{tr} \land \mathtt{C.com\_}\,(c, v)) \\ \Rightarrow & \forall\, \mathtt{C.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \Rightarrow \\ & \qquad (\exists\, \mathtt{C.State};\ \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr} \land \mathit{Rep} \land \mathtt{C.com\_}\,(c, v)) \\ & [\text{nach Induktionshypothese}] \\ \Rightarrow & \forall\, \mathtt{C.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \Rightarrow \\ & \qquad (\exists\, \Delta\mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr} \land \mathtt{A.com\_}\,(c, v) \land \mathit{Rep}') \\ & [\text{nach Vorraussetzung 3.1.3}] \\ \Leftrightarrow & \forall\, \mathtt{C.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \Rightarrow \\ & \qquad (\exists\, \mathtt{A.State}' \bullet \mathtt{A}_{tr}\, ⨾\, \mathtt{A.com\_}\,(c, v) \land \mathit{Rep}') \\ & [\text{Definition von } ⨾] \\ \Leftrightarrow & \forall\, \mathtt{C.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \Rightarrow (\exists\, \mathtt{A.State}' \bullet \mathtt{A}_{(tr \frown \langle (c,v) \rangle)} \land \mathit{Rep}') \\ & \left[\text{Definition von } \mathtt{A}_{tr \frown \langle (c,v) \rangle}\right] \\ \Rightarrow & \forall\, \mathtt{C.State} \bullet \mathtt{C}^{\star}_{tr \frown \langle (c,v) \rangle} \Rightarrow (\exists\, \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr \frown \langle (c,v) \rangle} \land \mathit{Rep}) \\ & [\text{Definition von } {}^{\star}] \end{array}$$

Zum Beweis der 2. Behauptung:

$$\begin{array}{ll} & (tr, R) \in \mathcal{F}[\![\mathtt{C}]\!] \\ \Leftrightarrow & \exists\, \mathtt{C.State} \bullet \mathtt{C}^{\star}_{tr} \land \mathit{Ref}_{\mathtt{C}}\, R \\ & [\text{Definition von } \mathcal{F}[\![\cdot]\!]] \\ \Rightarrow & \exists\, \mathtt{C.State};\ \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr} \land \mathit{Rep} \land \mathit{Ref}_{\mathtt{C}}\, R \end{array}$$

$$
\begin{array}{ll}
 & \text{[wegen Behauptung 1]} \\
\Rightarrow & \exists\, \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr} \wedge \mathit{Ref}_{\mathtt{A}}\, R \\
 & \text{[nach Voraussetzung 3.1.2]} \\
\Leftrightarrow & (tr, R) \in \mathcal{F}[\![\mathtt{A}]\!]
\end{array}
$$

□

Satz 3.5 (Korrektheit von FS) *Sind* $\mathtt{A} = \mathtt{spec\ I;\ P;\ Z^A\ end\_spec}$ *und* $\mathtt{C} = \mathtt{spec\ I;\ P;\ Z^C\ end\_spec}$ *zwei* CSP-Z*-Spezifikationen, und gilt* $\mathtt{A} \sim_{FS} \mathtt{C}$*, so folgt* $\mathtt{A} \sqsubseteq \mathtt{C}$.

Beweis.
Sei *Rep* ein *FS*-Schema von $\mathtt{spec\ I;\ Z^A\ end\_spec}$ nach $\mathtt{spec\ I;\ Z^A\ end\_spec}$. Wegen 3.4 gilt

$$\mathcal{F}[\![\mathtt{spec\ I;\ Z^A\ end\_spec}]\!] \subseteq \mathcal{F}[\![\mathtt{spec\ I;\ Z^A\ end\_spec}]\!].$$

Da $\mathcal{D}[\![\mathtt{spec\ I;\ Z^A\ end\_spec}]\!] = \varnothing = \mathcal{D}[\![\mathtt{spec\ I;\ Z^C\ end\_spec}]\!]$, folgt mit Bemerkung 3.3 die Behauptung. □

Analog zu Lemma 3.4 gilt auch für den Fall $\mathtt{A} \backsim_{BS} \mathtt{C}$, daß A die Traces von C nachvollziehen kann und die Failures von C in denen von A enthalten sind.

Lemma 3.6 *Seien* $\mathtt{A} = \mathtt{spec\ I;\ Z^A\ end\_spec}$ *und* $\mathtt{C} = \mathtt{spec\ I;\ Z^C\ end\_spec}$ *zwei Spezifikationen und Rep ein Repräsentationsschema.*

1. *Gelten 3.2.1 und 3.2.3, so auch für alle* $tr : \mathrm{seq}\, \mathit{Comm}(\mathit{Chans}(\mathcal{I}))$:

$$\forall\, \mathtt{C.State};\ \mathtt{A.State} \bullet \mathtt{C}^{\star}_{tr} \wedge \mathit{Rep} \Rightarrow \mathtt{A}^{\star}_{tr}$$

2. *Gilt außerdem 3.2.2, dann folgt*

$$\mathcal{F}[\![\mathtt{C}]\!] \subseteq \mathcal{F}[\![\mathtt{A}]\!]$$

Beweis. Behauptung 1 wird wiederum durch Induktion über die Länge der Traces bewiesen:

$$
\begin{array}{ll}
 & \forall\, \mathtt{C.State};\ \forall\, \mathtt{A.State} \bullet \mathtt{C.Init\_State} \wedge \mathit{Rep} \Rightarrow \mathtt{A.Init\_State} \\
 & \text{[nach Voraussetzung 3.2.1]} \\
\Leftrightarrow & \forall\, \mathtt{C.State};\ \mathtt{A.State} \bullet \mathtt{C}^{\star}_{\langle\rangle} \wedge \mathit{Rep} \Rightarrow \mathtt{A}^{\star}_{\langle\rangle}
\end{array}
$$

Sei nun $tr \frown \langle (c, v) \rangle : \operatorname{seq} Comm(Chans(\mathcal{I}))$.

$$
\begin{array}{ll}
 & \forall \mathtt{C.State}'; \mathtt{A.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \land Rep' \Rightarrow \\
 & \quad (\exists \mathtt{C.State} \bullet \mathtt{C}^{\star}_{tr} \land \mathtt{C.com\_}(c, v) \land Rep') \\
\Rightarrow & \forall \mathtt{C.State}'; \mathtt{A.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \land Rep' \Rightarrow \\
 & \quad (\exists \mathtt{C.State}; \mathtt{A.State} \bullet \mathtt{C}^{\star}_{tr} \land Rep \land \mathtt{A.com\_}(c, v)) \\
 & [\text{nach Voraussetzung 3.2.3}] \\
\Rightarrow & \forall \mathtt{C.State}'; \mathtt{A.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \land Rep' \Rightarrow \\
 & \quad (\exists \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr} \land \mathtt{A.com\_}(c, v)) \\
 & [\text{nach Induktionshypothese}] \\
\Leftrightarrow & \forall \mathtt{C.State}'; \mathtt{A.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \land Rep' \Rightarrow \mathtt{A}_{tr \frown \langle (c,v) \rangle} \\
\Leftrightarrow & \forall \mathtt{C.State}; \mathtt{A.State} \bullet \mathtt{C}^{\star}_{tr \frown \langle (c,v) \rangle} \land Rep \Rightarrow \mathtt{A}^{\star}_{tr \frown \langle (c,v) \rangle}
\end{array}
$$

Zum Beweis der 2. Behauptung:

$$
\begin{array}{ll}
 & (tr, R) \in \mathcal{F}[\![\mathtt{C}]\!] \\
\Leftrightarrow & \exists \mathtt{C.State} \bullet \mathtt{C}^{\star}_{tr} \land Ref_{\mathtt{C}}\, R \\
 & [\text{Definition von } \mathcal{F}[\![\cdot]\!]] \\
\Rightarrow & \exists \mathtt{C.State}; \mathtt{A.State} \bullet \mathtt{C}^{\star}_{tr} \land Rep \land Ref_{\mathtt{A}}\, R \\
 & [\text{nach Voraussetzung 3.2.2}] \\
\Rightarrow & \exists \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr} \land Ref_{\mathtt{A}}\, R \\
 & [\text{wegen Behauptung 1}] \\
\Leftrightarrow & (tr, R) \in \mathcal{F}[\![\mathtt{A}]\!]
\end{array}
$$

□

Satz 3.7 (Korrektheit von BS) *Sind* $\mathtt{A} = \mathtt{spec\,I};\ \mathtt{P};\ \mathtt{Z}^{\mathtt{A}}\,\mathtt{end\_spec}$ *und* $\mathtt{C} = \mathtt{spec\,I};\ \mathtt{P};\ \mathtt{Z}^{\mathtt{C}}\,\mathtt{end\_spec}$ *zwei* CSP-Z*-Spezifikationen, und gilt* $\mathtt{A} \backsim_{BS} \mathtt{C}$*, so folgt* $\mathtt{A} \sqsubseteq \mathtt{C}$.

Beweis.
Sei *Rep* ein *BS*-Schema von $\mathtt{spec\,I};\ \mathtt{Z}^{\mathtt{A}}\,\mathtt{end\_spec}$ nach $\mathtt{spec\,I};\ \mathtt{Z}^{\mathtt{C}}\,\mathtt{end\_spec}$. Wegen 3.6 gilt

$$\mathcal{F}[\![\mathtt{spec\,I};\ \mathtt{Z}^{\mathtt{A}}\,\mathtt{end\_spec}]\!] \subseteq \mathcal{F}[\![\mathtt{spec\,I};\ \mathtt{Z}^{\mathtt{A}}\,\mathtt{end\_spec}]\!].$$

Da $\mathcal{D}[\![\mathtt{spec\,I};\ \mathtt{Z}^{\mathtt{A}}\,\mathtt{end\_spec}]\!] = \varnothing = \mathcal{D}[\![\mathtt{spec\,I};\ \mathtt{Z}^{\mathtt{C}}\,\mathtt{end\_spec}]\!]$, folgt mit Bemerkung 3.3 die Behauptung. □

3.3 Eingeschränkte Vollständigkeit

Vorwärtssimulation und Rückwärtssimulation sind zusammen eingeschränkt vollständig, wenn die betrachteten Spezifikationen keine CSP-Prozesse enthalten.

Definition 3.8 (Eingeschränkte Vollständigkeit)
Seien $\mathtt{A} = \mathtt{spec\ I;\ Z^A\ end\_spec}$, $\mathtt{C} = \mathtt{spec\ I;\ Z^C\ end\_spec} \in SPEC$. *Die Regeln* **FS** *und* **BS** *heißen eingeschränkt vollständig, wenn jede Verfeinerung* $\mathtt{A} \sqsubseteq \mathtt{C}$ *durch (mehrmalige) Anwendung der Regeln bewiesen werden kann.* □

Zum Beweis der eingeschränkten Vollständigkeit wird eine kanonische Darstellung von Spezifikationen der Form $\mathtt{S} = \mathtt{spec\ I;\ Z^S\ end\_spec}$ benötigt. Die Normalform von S ist eine Spezifikation $\mathtt{S}_\natural$, deren Zustände $\theta\mathtt{S}_\natural.\mathtt{State}$ Elemente von $\mathcal{F}[\![\mathtt{S}]\!]$ sind. Die Normalform $\mathtt{S}_\natural$ enthält die Failures-Menge der Spezifikation S und vollzieht mit dieser Information das Verhalten von S nach. Intuitiv ist damit bereits klar, daß die beiden Spezifikationen äquivalent sind. Diese Vermutung wird in 3.10 und 3.11 bewiesen. Vorher wird jedoch die Normalform noch formal definiert.

Definition 3.9 (Normalform) *Sei* $\mathtt{S} = \mathtt{spec\ I;\ Z^S\ end\_spec} \in SPEC$ *mit* $[\![\mathtt{S}]\!] = (\mathcal{I}, \mathcal{F}, \varnothing)$ *gegeben. Die* Normalform *von* S *ist definiert durch* $\mathtt{S}_\natural = \mathtt{spec\ I;\ Z^{S_\natural}\ end\_spec}$, *wobei* $\mathtt{Z^{S_\natural}}$ *die nachfolgenden Schemata enthält.*

$\mathtt{S}_\natural.\mathtt{State}$

$$tr : \mathrm{seq} Comm(Chans(\mathcal{I}))$$
$$X : \mathbb{P}\, Comm(Chans(\mathcal{I}))$$
$$\hline$$
$$(tr, X) \in \mathcal{F}$$

$\mathtt{S}_\natural.\mathtt{Init\_State}$

$$\mathtt{S}_\natural.\mathtt{State}'$$
$$\hline$$
$$tr' = \langle\,\rangle \land X' \in \{\, Y \mid (tr', Y) \in \mathcal{F}\}$$

Für alle $c \in Chans(\mathcal{I})$

$\mathtt{S}_\natural.\mathtt{com\_}\, c$

$$\Delta\mathtt{S}_\natural.\mathtt{State}$$
$$@c : type_{\mathcal{I}}(c)$$
$$\hline$$
$$(tr \frown \langle (c, @c)\rangle, \varnothing) \in \mathcal{F} \land (c, @c) \notin X$$
$$tr' = tr \frown \langle (c, @c)\rangle \land X' \in \{\, Y \mid (tr', Y) \in \mathcal{F}\}$$

□

Die Vorbedingung pre $\mathtt{S}_\natural$.com_ c einer Operation $\mathtt{S}_\natural$.com_ c ist äquivalent zum folgenden Schema:

$$
\begin{array}{|l}
tr : \operatorname{seq} \mathit{Comm}(\mathcal{I}[\![\mathtt{S}]\!]) \\
X : \mathbb{P}\, \mathit{Comm}(\mathcal{I}[\![\mathtt{S}]\!]) \\
@c : \mathit{type}_{\mathcal{I}}(c) \\
\hline
(tr \frown \langle (c, @c) \rangle, \varnothing) \in \mathcal{F}[\![S]\!] \\
(c, @c) \notin X \\
\hline
\end{array}
$$

Die nächsten beiden Lemmata beweisen die Äquivalenz zwischen einer Spezifikation und ihrer Normalform.

Lemma 3.10 *Sei* $\mathtt{S} = \mathtt{spec}\ \mathtt{I};\ \mathtt{Z}^{\mathtt{S}}\ \mathtt{end\_spec}$ *mit* $[\![\mathtt{S}]\!] = (\mathcal{I}, \mathcal{F}, \varnothing)$. *Dann gilt:* $\mathtt{S}_\natural \sim_{FS} \mathtt{S}$.

Beweis. Das Repräsentationsschema

Rep

$$
\begin{array}{|l}
\mathtt{S}_\natural.\mathtt{State} \\
\mathtt{S}.\mathtt{State} \\
\hline
S^\star_{tr} \\
\forall\, c : \mathit{Chans}(\mathcal{I}) \bullet \\
\quad \forall\, @c : \mathit{type}_{\mathcal{I}}(c) \bullet \\
\quad \Big((c, @c) \notin X \land (tr \frown \langle (c, @c) \rangle, \varnothing) \in \mathit{fail} \Big) \Leftrightarrow \operatorname{pre} \mathtt{S}.\mathtt{com}\_\ c \\
\hline
\end{array}
$$

ist ein *FS*-Schema von $\mathtt{S}_\natural$ nach $\mathtt{S}$. □

Lemma 3.11 *Sei* $\mathtt{S} = \mathtt{spec}\ \mathtt{I};\ \mathtt{Z}^{\mathtt{S}}\ \mathtt{end\_spec}$. *Dann gilt:* $\mathtt{S} \backsim_{BS} \mathtt{S}_\natural$.

Beweis. Das Repräsentationsschema

Rep

$$
\begin{array}{|l}
\mathtt{S}.\mathtt{State} \\
\mathtt{S}_\natural.\mathtt{State} \\
\hline
S^\star_{tr} \\
\hline
\end{array}
$$

ist ein *BS*-Schema von $\mathtt{S}$ nach $\mathtt{S}_\natural$. □

Zwischen den Normalformen zweier Spezifikationen A und C läßt sich in jedem Fall eine Rückwärtssimulation finden, wenn A durch C verfeinert wird. Das ist das eigentliche Hauptresultat dieses Abschnitts.

Satz 3.12 *Für zwei* CSP-Z*-Spezifikationen* $\mathtt{A} = \mathtt{spec\,I;\ Z^A\ end\_spec}$ *und* $\mathtt{C} = \mathtt{spec\,I;\ Z^C\ end\_spec}$ *gilt:* $\mathtt{A} \sqsubseteq \mathtt{C} \Rightarrow \mathtt{A}_\natural \backsim_{BS} \mathtt{C}_\natural$.

Beweis.

Rep

$\mathtt{A}_\natural.\mathtt{State}$
$\mathtt{C}_\natural.\mathtt{State}$

$A_{\natural tr}^\star = C_{\natural tr}^\star$

ist ein *BS*-Schema von $A_\natural$ nach $C_\natural$. □

Die Vollständigkeit der Regeln **FS** und **BS** ist nun eine einfache Folgerung. Die Vollständigkeit gilt nicht für CSP-Z-Spezifikationen allgemein, denn:

$$\mathtt{spec\,I;\ P;\ Z^A\ end\_spec} \sqsubseteq \mathtt{spec\,I;\ P;\ Z^C\ end\_spec} \not\Rightarrow$$
$$\mathtt{spec\,I;\ Z^A\ end\_spec} \sqsubseteq \mathtt{spec\,I;\ Z^C\ end\_spec}$$

Satz 3.13 (Eingeschränkte Vollständigkeit)
Die Regeln **FS** *und* **BS** *zusammen sind eingeschränkt vollständig.*

Beweis. Seien $\mathtt{A} = \mathtt{spec\,I;\ Z^A\ end\_spec}$ und $\mathtt{C} = \mathtt{spec\,I;\ Z^C\ end\_spec}$ zwei Spezifikationen mit $\mathtt{A} \sqsubseteq \mathtt{C}$. Nach 3.10,3.11 und 3.12 gilt:

$$\mathtt{A} \backsim_{BS} \mathtt{A}_\natural \backsim_{BS} \mathtt{C}_\natural \sim_{FS} \mathtt{C}$$

□

Die Aussage von 3.13 ist nicht, daß sich jede Verfeinerung durch Anwendung einer der beiden Regeln **FS** oder **BS** bewiesen werden kann, sondern daß durch Anwendung beider Regeln in höchstens drei Schritten jede Verfeinerung bewiesen werden kann. Es sind Fälle vorstellbar, in denen zum Beweis einer Verfeinerung für einen Kanal eine Vorwärtssimulation benötigt wird und für einen anderen eine Rückwärtssimulation, daß es aber nicht möglich ist, mit nur einer der beiden Regeln die Verfeinerung zu beweisen [GM89].

3.4 Anmerkungen

Die Regeln 3.1 und 3.2 sind gleichwertig mit denen von Josephs in [Jos88]. Josephs definiert die Normalform einer Spezifikation mittels ihrer Readiness-Semantik. Die hier verwendete Darstellung ähnelt der von Woodcock und Morgan in [WM90]. Sie verwenden jedoch als Grundlage Action-Systeme [Mor90] und den Refinement Calculus [Mor94]. In [Mor90] wird eine Verbindung zwischen Action-Systemen und der Failures/Divergences-Semantik von

CSP hergestellt. Durch die wp-Semantik des Refinement Calculus kommen Divergenzen zustande. Dadurch unterscheiden sich die in [WM90] gefundenen Datenverfeinerungsregeln etwas von den in diesem Kapitel entwickelten Regeln. In [Kin90] wird ein Zusammenhang zwischen der Semantik von Z und der wp-Semantik hergestellt. Das läßt vermuten, daß die im Ansatz von Morgan und Woodcock erzielten Ergebnisse auch hier in ähnlicher Form auftreten. In Kapitel 5 ist dabei insbesondere die Arbeit von Butler [But92, But93] nützlich, um eine Idee von den Datenverfeinerungsregeln zu bekommen, die in CSP-Z möglich sind.

Kapitel 4

Fallstudie A

In [MG90, WD96] werden zur Demonstration der Anwendung von Datenverfeinerungsregeln zwei Spezifikationen eines einfachen Taschenrechners angegeben. Diese dienen als Grundlage für die Spezifikationen dieses Kapitels. Anhand der beiden Spezifikationen wird hier vorgeführt, wie die beiden Regeln **FS** und **BS** zur Datenverfeinerung von CSP-Z-Spezifikationen eingesetzt werden können. In Abschnitt 4.1 werden zwei mögliche Spezifikationen `ACalculator` und `CCalculator` eines solchen Taschenrechners angegeben. In Abschnitt 4.2 wird bewiesen, daß `ACalculator` durch `CCalculator` verfeinert wird und im darauf folgenden Abschnitt die umgekehrte Beziehung.

4.1 Zwei Taschenrechner

Ein einfacher Taschenrechner besitzt eine *clear*-Taste, um den Speicher zu löschen, eine *enter*-Taste, um eine eingegebene Zahl zu bestätigen und eine *mean*-Taste, um den Mittelwert aller seit dem letzten Löschen eingegebenen Zahlen auszugeben.

Der Taschenrechner `ACalculator` auf Seite 54 liest eine Sequenz s von nichtnegativen Zahlen über den Kanal *enter* ein. Erfolgt eine Kommunikation über den Kanal *mean*, so wird die Summe der in der Sequenz s gespeicherten Zahlen berechnet und der Mittelwert ausgegeben. Der CSP-Teil der Spezifikation stellt sicher, daß der Speicher gelöscht wird, bevor irgendeine Zahl eingegeben wird.

Der andere Taschenrechner `CCalculator` auf Seite 55 addiert die über den Kanal *enter* eingelesenen Zahlen in der Variablen *sum* und speichert ihre Anzahl in der Variablen n. Bei einer Kommunikation über den Kanal *mean* gibt er den Mittelwert der in *sum* addierten n Zahlen aus. Der CSP-Teil der Spezifikation `CCalculator` ist mit dem von `ACalculator` identisch.

---

spec ACalculator

channel *clear* : signal;
channel *enter* : $\mathbb{N}$;
channel *mean* : $\mathbb{Q}$;

main $=$ *clear* $\rightarrow$ $(\mu\, X \bullet (\mathit{enter} \rightarrow X \,\Box\, \mathit{mean} \rightarrow X \,\Box$ main$))$

State

$s : \mathit{seq}\mathbb{N}$

Init_State

State$'$

---

true

com_ *clear*

ΔState

$s' = \langle\,\rangle$

com_ *enter*

ΔState
$@\mathit{enter} : \mathbb{N}$

$s' = s \frown \langle @\mathit{enter} \rangle$

com_ *mean*

ΞState
$@\mathit{mean} : \mathbb{Q}$

$s \neq \langle\,\rangle$
$@\mathit{mean} = \frac{\sum_{i=1}^{\#s} s(i)}{\#s}$

end_spec ACalculator

Die Operationsschemata der Spezifikation ACalculator haben die folgenden Vorbedingungen:

pre ACalculator.com_ *clear* $\equiv$
$\quad [\,$ACalculator.State $\mid$ true $]$
pre ACalculator.com_ *enter* $\equiv$
$\quad [\,$ACalculator.State; $@\mathit{enter} : \mathbb{N} \mid$ true $]$
pre ACalculator.com_ *mean* $\equiv$
$\quad [\,$ACalculator.State; $@\mathit{mean} : \mathbb{Q} \mid s \neq \langle\,\rangle\,]$

---

spec CCalculator

channel *clear* : signal;
channel *enter* : $\mathbb{N}$;
channel *mean* : $\mathbb{Q}$;

main $= \mathit{clear} \rightarrow (\mu\, X \bullet (\mathit{enter} \rightarrow X \,\Box\, \mathit{mean} \rightarrow X \,\Box\, \mathtt{main}))$

State

$\mathit{sum} : \mathbb{N}$
$n : \mathbb{N}$

Init_State

State′

true

com_ *clear*

ΔState

$\mathit{sum}' = 0 \land n' = 0$

com_ *enter*

ΔState
$@\mathit{enter} : \mathbb{N}$

$\mathit{sum}' = \mathit{sum} + @\mathit{enter}$
$n' = n + 1$

com_ *mean*

ΞState
$@\mathit{mean} : \mathbb{Q}$

$n \neq 0$
$@\mathit{mean} = \frac{\mathit{sum}}{n}$

end_spec CCalculator

Die zugehörigen Vorbedingungen der Operationsschemata der Spezifikation CCalculator lauten:

pre CCalculator.com_ *clear* $\equiv$
$[\,\mathtt{CCalculator.State} \mid \text{true}\,]$

pre CCalculator.com_ *enter* $\equiv$
$[\,\mathtt{CCalculator.State};\ @\mathit{enter} : \mathbb{N} \mid \text{true}\,]$

pre CCalculator.com_ *mean* $\equiv$
$[\,\mathtt{CCalculator.State};\ @\mathit{mean} : \mathbb{Q} \mid n \neq 0\,]$

4.2 Der Eine verfeinert den Anderen

Es gilt ACalculator $\sqsubseteq$ CCalculator. Da der CSP-Teil beider Spezifikationen gleich ist, braucht nur die Verfeinerungsbeziehung für den Z-Teil bewiesen zu werden. Der Beweis erfolgt durch Anwendung der Regel **FS**, das heißt es wird gezeigt:

$$\texttt{ACalculator} \sim_{FS} \texttt{CCalculator}.$$

Zum Beweis dieser Aussage müssen die drei in Regel **FS** angegebenen Bedingungen "Initialisierung", "Definiertheit" und "Fortschritt" erfüllt sein. Das Schema *Rep* ist ein *FS*-Schema von ACalculator nach CCalculator.

Rep

ACalculator.State
CCalculator.State

$sum = \sum_{i=1}^{\#s} s(i)$
$n = \#s$

Initialisierung

Die Aussage

$$\forall\, sum', n' : \mathbb{N} \bullet \exists\, s' : \operatorname{seq}\mathbb{N} \bullet \text{true} \Rightarrow \text{true} \wedge sum' = \sum_{i=1}^{\#s'} s'(i) \wedge n' = \#s'$$

ist wahr, das heißt

$$\forall\, \texttt{CCalculator.State}' \bullet \exists\, \texttt{ACalculator.State}' \bullet \texttt{CCalculator.Init\_State} \Rightarrow \texttt{ACalculator.Init\_State} \wedge Rep'$$

Definiertheit

Es sind drei Teilbeweise für die Kanäle *clear*, *enter* und *mean* zu führen.

Definiertheit von *clear*

$$\begin{aligned} \forall\, sum, n : \mathbb{N};\ s : \operatorname{seq}\mathbb{N} \bullet & \\ & \operatorname{pre} \texttt{ACalculator.com\_}\, clear \wedge Rep \\ \Leftrightarrow\ & \text{true} \wedge sum = \sum_{i=1}^{\#s} s(i) \wedge n = \#s \\ \Rightarrow\ & \text{true} \\ \Leftrightarrow\ & \operatorname{pre} \texttt{CCalculator.com\_}\, clear \end{aligned}$$

Definiertheit von *enter*

$$
\begin{array}{ll}
\forall\, sum, n : \mathbb{N};\ s : \operatorname{seq}\mathbb{N};\ @enter : \mathbb{N} \bullet & \\
\qquad \operatorname{pre} \texttt{ACalculator.com\_}\ enter \wedge Rep & \\
\Leftrightarrow & \operatorname{true} \wedge sum = \sum_{i=1}^{\#s} s(i) \wedge n = \#s \\
\Rightarrow & \operatorname{true} \\
\Leftrightarrow & \operatorname{pre} \texttt{CCalculator.com\_}\ enter
\end{array}
$$

Definiertheit von *mean*

$$
\begin{array}{ll}
\forall\, sum, n : \mathbb{N};\ s : \operatorname{seq}\mathbb{N};\ @mean : \mathbb{Q} \bullet & \\
\qquad \operatorname{pre} \texttt{ACalculator.com\_}\ mean \wedge Rep & \\
\Leftrightarrow & s \neq \langle\,\rangle \wedge sum = \sum_{i=1}^{\#s} s(i) \wedge n = \#s \\
\Rightarrow & s \neq \langle\,\rangle \wedge n = \#s \\
\Rightarrow & n \neq 0 \\
\Leftrightarrow & \operatorname{pre} \texttt{CCalculator.com\_}\ mean
\end{array}
$$

Fortschritt

Der Beweis der Fortschrittsbedingung ist ebenfalls in drei Teile untergliedert.

Fortschritt von *clear*

$$
\begin{array}{ll}
sum, n : \mathbb{N};\ sum', n' : \mathbb{N};\ s : \operatorname{seq}\mathbb{N} \bullet & \\
\qquad Rep \wedge \texttt{CCalculator.com\_}\ clear & \\
\Leftrightarrow & sum = \sum_{i=1}^{\#s} s(i) \wedge n = \#s \wedge sum' = 0 \wedge n' = 0 \\
\Rightarrow & sum' = \sum_{i=1}^{\#\langle\rangle} \langle\,\rangle(i) \wedge n' = \#\langle\,\rangle \\
\Leftrightarrow & (\exists\, s' : \operatorname{seq}\mathbb{N} \bullet s' = \langle\,\rangle \wedge sum' = \sum_{i=1}^{\#s'} s'(i) \wedge n' = \#s') \\
\Leftrightarrow & (\exists\, \texttt{ACalculator.State} \bullet \texttt{ACalculator.com\_}\ clear \wedge Rep')
\end{array}
$$

Fortschritt von *enter*

$$
\begin{array}{ll}
sum, n : \mathbb{N};\ sum', n' : \mathbb{N};\ s : \operatorname{seq}\mathbb{N};\ @enter : \mathbb{N} \bullet & \\
\qquad Rep \wedge \texttt{CCalculator.com\_}\ enter & \\
\Leftrightarrow & sum = \sum_{i=1}^{\#s} s(i) \wedge n = \#s \wedge sum' = sum + @enter \wedge n' = n + 1 \\
\Rightarrow & sum' = \left(\sum_{i=1}^{\#s} s(i)\right) + @enter \wedge n' = (\#s) + 1
\end{array}
$$

$$\Rightarrow \quad sum' = \sum_{i=1}^{\#(s \frown \langle @enter \rangle)} (s \frown \langle @enter \rangle)(i) \land n' = \#(s \frown \langle @enter \rangle)$$
$$\Leftrightarrow \quad (\exists\, s' : \operatorname{seq} \mathbb{N} \bullet s' = s \frown \langle @enter \rangle \land sum' = \sum_{i=1}^{\#s'} s'(i) \land n' = \#s')$$
$$\Leftrightarrow \quad (\exists\, \texttt{ACalculator.State} \bullet \texttt{ACalculator.com\_}\, enter \land Rep')$$

Fortschritt von *mean*

$$sum, n : \mathbb{N};\ sum', n' : \mathbb{N};\ s : \operatorname{seq} \mathbb{N};\ @mean : \mathbb{Q} \bullet$$
$$Rep \land \texttt{CCalculator.com\_}\, mean$$
$$\Leftrightarrow \quad sum = \sum_{i=1}^{\#s} s(i) \land n = \#s$$
$$\land\ n \neq 0 \land @mean = \frac{sum}{n} \land sum' = sum \land n' = n$$
$$\Rightarrow \quad s \neq \langle\,\rangle \land @mean = \frac{\sum_{i=1}^{\#s} s(i)}{\#s} \land sum' = \sum_{i=1}^{\#s} s(i) \land n' = \#s$$
$$\Leftrightarrow \quad (\exists\, s' : \operatorname{seq} \mathbb{N} \bullet \quad s' = s \land @mean = \frac{\sum_{i=1}^{\#s} s(i)}{\#s}$$
$$\land\ sum' = \sum_{i=1}^{\#s'} s'(i) \land n' = \#s')$$
$$\Leftrightarrow \quad (\exists\, \texttt{ACalculator.State} \bullet \texttt{ACalculator.com\_}\, mean \land Rep')$$

Also ist *Rep* ein *FS*-Schema und `ACalculator` $\sim_{FS}$ `CCalculator` bewiesen. □

4.3 Der Andere verfeinert den Einen

Die beiden Spezifikationen `ACalculator` und `CCalculator` sind äquivalent. Im letzten Abschnitt wurde `ACalculator` $\sqsubseteq$ `CCalculator` mit der Regel **FS** bewiesen. In diesem Abschnitt wird

`CCalculator` $\backsim_{BS}$ `ACalculator`

bewiesen, und insgesamt folgt

`ACalculator` $\equiv$ `CCalculator`.

Das Repräsentationsschema *Req* ist ein *BS*-Schema von `CCalculator` nach `ACalculator`.

Req

`CCalculator.State`
`ACalculator.State`
$sum = \sum_{i=1}^{\#s} s(i) \land n = \#s$

Es sind die drei Bedingungen "Initialisierung", "Definiertheit" und "Ausführbarkeit" der Regel **BS** für das Schema *Req* nachzuweisen.

Initialisierung

$$
\begin{aligned}
&\forall\, s' : \operatorname{seq}\mathbb{N};\ sum', n' : \mathbb{N} \bullet \\
&\qquad \texttt{ACalculator.Init\_State} \land Req' \\
&\Leftrightarrow \quad \text{true} \land sum' = \sum_{i=1}^{\#s'} s'(i) \land n' = \#s \\
&\Rightarrow \quad \text{true} \\
&\Leftrightarrow \quad \texttt{CCalculator.Init\_State}
\end{aligned}
$$

Definiertheit

Zum Nachweis der Definiertheitseigenschaft sind drei Teilbeweise für die Kanäle *clear*, *enter* und *mean* zu führen. Die drei Beweise setzen die folgende Tatsache voraus, die offensichtlich wahr ist:

$$\forall\, s : \operatorname{seq}\mathbb{N} \bullet \exists\, sum, n : \mathbb{N} \bullet Req \tag{4.1}$$

Definiertheit von *clear*

$$
\begin{aligned}
&\forall\, s : \operatorname{seq}\mathbb{N} \bullet \exists\, sum, n : \mathbb{N} \bullet \\
&\qquad Req \\
&\Leftrightarrow \quad Req \land (\text{true} \Rightarrow \text{true}) \\
&\Leftrightarrow \quad Req \land (\operatorname{pre}\texttt{CCalculator.com\_}\, clear \Rightarrow \operatorname{pre}\texttt{ACalculator.com\_}\, clear)
\end{aligned}
$$

Definiertheit von *enter*

$$
\begin{aligned}
&\forall\, s : \operatorname{seq}\mathbb{N};\ @enter : \mathbb{N} \bullet \exists\, sum, n : \mathbb{N} \bullet \\
&\qquad Req \\
&\Leftrightarrow \quad Req \land (\text{true} \Rightarrow \text{true}) \\
&\Leftrightarrow \quad Req \land (\operatorname{pre}\texttt{CCalculator.com\_}\, enter \Rightarrow \operatorname{pre}\texttt{ACalculator.com\_}\, enter)
\end{aligned}
$$

Definiertheit von *mean*

$$
\begin{aligned}
&\forall\, s : \operatorname{seq}\mathbb{N};\ @mean : \mathbb{Q} \bullet \exists\, sum, n : \mathbb{N} \bullet \\
&\qquad Req \\
&\Leftrightarrow \quad sum = \sum_{i=1}^{\#s} s(i) \land n = \#s \land (n \neq 0 \Rightarrow s \neq \langle\,\rangle) \\
&\Leftrightarrow \quad Req \land (\operatorname{pre}\texttt{CCalculator.com\_}\, mean \Rightarrow \operatorname{pre}\texttt{ACalculator.com\_}\, mean)
\end{aligned}
$$

Fortschritt

Die drei angebenen Teilbeweise entsprechen wieder den drei Kanälen der Spezifikationen.

Fortschritt von *clear*

$$
\begin{array}{ll}
\forall\, s : \operatorname{seq}\mathbb{N};\ s' : \operatorname{seq}\mathbb{N};\ sum', n' : \mathbb{N} & \\
\qquad \texttt{ACalculator.com\_}\ clear \wedge Req' & \\
\Leftrightarrow & s' = \langle\,\rangle \wedge sum' = \sum_{i=1}^{\#s'} s'(i) \wedge n' = \#s' \\
\Rightarrow & sum' = 0 \wedge n' = 0 \\
\Leftrightarrow & (\exists\, sum, n : \mathbb{N} \bullet sum = \sum_{i=1}^{\#s} s(i) \wedge n = \#s \wedge sum' = 0 \wedge n' = 0) \\
& [\text{wegen } (4.1)] \\
\Leftrightarrow & (\exists\, sum, n : \mathbb{N} \bullet Req \wedge \texttt{CCalculator.com\_}\ clear)
\end{array}
$$

Fortschritt von *enter*

$$
\begin{array}{ll}
\forall\, s : \operatorname{seq}\mathbb{N};\ s' : \operatorname{seq}\mathbb{N};\ sum', n' : \mathbb{N};\ @enter : \mathbb{N} & \\
\qquad \texttt{ACalculator.com\_}\ enter \wedge Req' & \\
\Leftrightarrow & s' = s \frown \langle @enter \rangle \wedge sum' = \sum_{i=1}^{\#s'} s'(i) \wedge n' = \#s' \\
\Rightarrow & sum' = \sum_{i=1}^{\#(s \frown \langle @enter \rangle)} (s \frown \langle @enter \rangle)(i) \wedge n' = \#(s \frown \langle @enter \rangle) \\
\Rightarrow & sum' = \left(\sum_{i=1}^{\#s} s(i)\right) + @enter \wedge n' = (\#s) + 1 \\
\Leftrightarrow & (\exists\, sum, n : \mathbb{N} \bullet \quad sum = \sum_{i=1}^{\#s} s(i) \wedge n = \#s \\
& \qquad\qquad \wedge\, sum' = sum + @enter \wedge n' = n + 1) \\
\Leftrightarrow & (\exists\, sum, n : \mathbb{N} \bullet Req \wedge \texttt{CCalculator.com\_}\ enter)
\end{array}
$$

Fortschritt von *mean*

$$
\begin{array}{ll}
\forall\, s : \operatorname{seq}\mathbb{N};\ s' : \operatorname{seq}\mathbb{N};\ sum', n' : \mathbb{N};\ @mean : \mathbb{Q} & \\
\qquad \texttt{ACalculator.com\_}\ mean \wedge Req' & \\
\Leftrightarrow & s' = s \wedge @mean = \frac{\sum_{i=1}^{\#s} s(i)}{\#s} \wedge sum' = \sum_{i=1}^{\#s'} s'(i) \wedge n' = \#s' \\
\Rightarrow & n' \neq 0 \wedge @mean = sum' n' \wedge sum' = \sum_{i=1}^{\#s} s(i) \wedge n' = \#s \\
\Leftrightarrow & (\exists\, sum, n : \mathbb{N} \bullet \quad sum' = sum \wedge n' = n \wedge @mean = \frac{sum}{n} \\
& \qquad\qquad \wedge\, sum = \sum_{i=1}^{\#s} s(i) \wedge n = \#s) \\
\Leftrightarrow & (\exists\, sum, n : \mathbb{N} \bullet Req \wedge \texttt{CCalculator.com\_}\ mean)
\end{array}
$$

Damit ist alles bewiesen. □

Kapitel 5

Datenverfeinerung und lokale Kanäle

Ist $\mathtt{S} = \mathtt{T} \setminus \{c\} \in \mathit{SPEC}(\mathcal{I})$ eine Spezifikation mit $c \in \mathit{Chans}(\mathcal{I}[\![\mathtt{T}]\!])$, dann ist c ein lokaler Kanal von S. Kommunikationen dieses Kanals können von der Umgebung von S nicht beobachtet werden. Ist $\mathtt{S} = \mathtt{T} \setminus L$, wobei $L \subseteq \mathit{Chans}(\mathcal{I}[\![\mathtt{T}]\!])$ und T keine lokalen Kanäle enthält, dann ist $\mathcal{L}[\![\mathtt{S}]\!]$ definiert durch

$$\mathcal{I}[\![\mathtt{T}]\!] = \mathcal{I}[\![\mathtt{S}]\!] \cup \mathcal{L}[\![\mathtt{S}]\!].$$

Die Menge $\mathcal{L}[\![\mathtt{S}]\!]$ enthält die Kanalnamen und die zugehörige Typinformation der lokalen Kanäle von S.

Einführung lokaler Kanäle während einer transformationellen Entwicklung bedeutet, zu einer Spezifikation A eine Spezifikation $\mathtt{C} = \mathtt{D} \setminus L$ mit $L \subseteq \mathit{Chans}(\mathcal{I}[\![\mathtt{D}]\!])$ zu finden, so daß $\mathtt{A} \sqsubseteq \mathtt{C}$.

Durch die Einführung lokaler Kanäle in eine Spezifikation können komplexe Operationen sichtbarer Kanäle durch mehrere einfachere Operationen lokaler Kanäle implementiert werden. Aufgrund der Monotonie des Hiding-Operators kann die so erhaltene Spezifikation weiter verfeinert werden.

Im Z-Teil einer Spezifikation können Kommunikationen lokaler Kanäle beliebig oft hintereinander auftreten, solange die Vorbedingung der Operation mindestens eines lokalen Kanals wahr ist, ohne daß eine Kommunikation eines sichtbaren Kanals auftritt. Das bedeutet, daß möglicherweise unbeschränkt viele lokale Kommunikationen direkt hintereinander auftreten können, und die Spezifikation divergiert. Werden in eine Spezifikation lokale Kanäle eingeführt, so dürfen diese nicht zu mehr Divergenzen führen, als in der Ausgangsspezifikation enthalten sind.

Die Datenverfeinerungsregeln zur Einführung lokaler Kanäle verhindern unbeschränkte lokale Kommunikationsfolgen, indem sie die Anzahl der in einem Trace möglichen lokalen Kommunikationen durch eine Zahl beschränken, die aus den sichtbaren Kommunikationen berechnet wird.

Das Verhalten einer Spezifikation kann von der Reihenfolge abhängen, in der lokale Kanäle eingeführt wurden. Der Grund hierfür liegt darin, daß in CSP-Z Hiding im allgemeinen nicht kommutativ ist. Die Gleichung $\mathtt{S} \setminus \{c\} \setminus \{d\} = \mathtt{S} \setminus \{d\} \setminus \{c\}$ ist nur für bestimmte Spezifikationen S und Kanäle c, d wahr.

Die Reihenfolge, in der lokale Kanäle eingeführt werden, spielt keine Rolle, wenn die Zahl der hinzugefügten lokalen Kommunikationen nicht von den Kommunikationswerten der sichtbaren Kommunikationen abhängt, sondern nur von der Anzahl der sichtbaren Kommunikationen.

In Abschnitt 5.1 wird untersucht, unter welchen Umständen durch Anwendung des Hiding-Operators Divergenz entsteht. Abschnitt 5.2 stellt zwei Regeln zur Einführung lokaler Kanäle vor, und in Abschnitt 5.3 wird ihre Korrektheit bewiesen. Werden mehrmals hintereinander lokale Kanäle mittels der Regeln **LIFS** oder **LIBS** aus Abschnitt 5.2 eingeführt, so verhalten sie sich kommutativ bezüglich Hiding, wie in Abschnitt 5.4 gezeigt wird. Die beiden in Abschnitt 5.5 definierten Regeln können verwendet werden, um zu zeigen, daß eine Verfeinerung, die durch eine der Regeln aus Abschnitt 5.2 bewiesen wurde, in Wirklichkeit eine Äquivalenz ist. Ein Vollständigkeitsresultat wie in Kapitel 3 existiert jedoch nicht.

5.1 Divergenz

Sind in einer Spezifikation S unbeschränkt viele Kommunikationen der Kanäle $\{c_1, \ldots, c_n\}$ direkt hintereinander möglich, so divergiert $\mathtt{S} \setminus \{c_1, \ldots, c_n\}$.

Definition 5.1
Sei $\mathtt{S} = \mathtt{spec\ I;\ P;\ Z\ end\_spec} \in SPEC(\mathcal{I})$ *und* $H \subseteq Chans(\mathcal{I})$. *Ein Trace* $tr \in \mathrm{seq}\, Comm(\mathcal{I})$ *heißt*

H-beschränkt (*in* S), *wenn*
$$\exists\, n : \mathbb{N} \bullet \forall\, u : \mathrm{seq}\, Comm(H) \bullet (tr \frown u) \in traces(\mathtt{S}) \Rightarrow \#u < n;$$

H-unbeschränkt (*in* S), *wenn*
$$\forall\, n : \mathbb{N} \bullet \exists\, u : \mathrm{seq}\, Comm(H) \bullet (tr \frown u) \in traces(\mathtt{S}) \wedge \#u > n;$$

Die Spezifikation S *heißt* H-beschränkt, *wenn alle Traces* H*-beschränkt in* S *sind, und* S *heißt* H-unbeschränkt, *wenn es einen Trace gibt, der* H*-unbeschränkt in* S *ist.* □

Aus der H-Beschränktheit einer Spezifikation S folgt, daß $\mathtt{S} \setminus H$ nicht mehr Divergenzen enthält als S. Sind alle Traces einer Spezifikation S bezüglich aller Teilmengen H der Kanäle ihres Interfaces H-beschränkt, so kann durch Anwendung des Hiding-Operators keine Divergenz entstehen. Es gilt:

S ist H-unbeschränkt $\Leftrightarrow$ S ist nicht H-beschränkt

Ist S eine H-unbeschränkte Spezifikation, so sind die Divergenzen von S eine echte Untermenge der Divergenzen von $\mathtt{S} \setminus H$.

5.2 Einführung lokaler Kanäle

Um Lokale Kanäle in eine Spezifikation einzuführen muß garantiert werden, daß sie nicht divergieren. Das wird erreicht, indem ein Terminierungsschema *term* verwendet wird, um die Anzahl der lokalen Kommunikationen am Ende einer Trace durch die Anzahl der sichtbaren Kommunikationen zu beschränken. Dazu werden Ausdrücke $k_c(v) : \mathbb{N}$ eingeführt, die bei jeder sichtbaren Kommunikation (c, v) die Anzahl der möglichen lokalen Kommunikationen höchstens um einen Wert, der allein vom kommunizierten Wert v abhängt, vergrößert. Dadurch ist an einer Trace selbst abzulesen, um wieviele lokale Kommunikationen sie maximal verlängert werden kann. In jeder Trace können nur so viele lokale Kommunikationen direkt hintereinander vorkommen, wie es die Summe der $k_c(v)$, die vorher von den sichtbaren Kommunikationen aufaddiert wurden, zuläßt. Da die Initialisierung nicht sichtbar ist, muß in ihr die Anzahl der lokalen Kommunikationen durch eine Konstante k nach oben abgeschätzt werden. Der Wert der $k_c(v)$ darf nicht vom Zustand der Spezifikation abhängen, da dieser in der Semantik nicht sichtbar ist und nach einer Zustandsänderung der Spezifikation möglicherweise unbeschränkt viele lokale Kommunikationen auftreten können.

Definition 5.2 *Sei* $\mathtt{S} = \mathtt{spec\ I;\ Z^S\ end\_spec}$ *eine Spezifikation mit Interface* $\mathcal{I}$ *und* $L \subseteq Chans(\mathcal{I})$. *Die Spezifikation* S *heißt* schwach L-terminal, *wenn es eine Konstante* $k : \mathbb{N}$, *ein Schema*

term

S.State $t : \mathbb{Z}$
$t = \ldots$ [t ist durch einen Ausdruck über S.State definiert]

und für jeden Kanal $c \in Chans(\mathcal{I}) \setminus L$ *eine totale Funktion*

$$k_c : type_{\mathcal{I}}(c) \to \mathbb{N}$$

gibt, so daß

1. $\forall$ S.State $\bullet\ \exists\, t : \mathbb{Z} \bullet term$
2. $\forall\, term' \bullet$ S.Init_State $\Rightarrow 0 \leq t' \leq k$
3. $\forall\, c \in Chans(\mathcal{I}) \setminus L \bullet$
   $\forall\, \Delta term;\ @c : type_{\mathcal{I}}(c) \bullet$ S.com_ $c \wedge t \geq 0 \Rightarrow 0 \leq t' \leq t + k_c(@c)$

4. $\forall \ell \in L \bullet$
   $\forall \Delta term;\ @\ell : type_{\mathcal{I}}(\ell) \bullet \mathtt{S.com\_}\,\ell \wedge t \geq 0 \Rightarrow 0 \leq t' < t$

Die Komponente t von term heißt Variante. *Sind alle k_c konstante Funktionen, dh. hängen sie nicht von @c ab, dann heißt* S *eine* L-terminale *Spezifikation.* □

Ist S eine schwach L-terminale Spezifikation und

$$tr \in \operatorname{seq} Comm(Chans(\mathcal{I}[\![\mathtt{S}]\!]) \setminus L),$$

so ist die Summe $\sum_{tr}$ über tr definiert durch

$$\sum_{\langle\rangle} = 0$$
$$\sum_{tr^\frown\langle(c,v)\rangle} = \sum_{tr} + k_c(v),$$

wobei $(c, v) \in Comm(Chans(\mathcal{I}[\![\mathtt{S}]\!]) \setminus L)$ ist.

Das nächste Lemma notiert einige wichtige Eigenschaften schwach terminaler und terminaler Spezifikationen. Solche Spezifikationen erlauben es die Anzahl der unsichtbaren Kommunikationen nach oben abzuschätzen.

Lemma 5.3 *Ist* $\mathtt{S} = \mathtt{spec\ I;\ Z^S\ end\_spec}$ *schwach L-terminal, dann gilt für alle* $tr : \operatorname{seq} Comm(Chans(\mathcal{I}[\![\mathtt{S}]\!]))$ *und alle* $u : \operatorname{seq} Comm(L)$*:*

1. $\forall\, term \bullet S^\star_{tr} \Rightarrow 0 \leq t \leq k + \sum_{tr\uparrow L} - \#(tr \downarrow L)$
2. $\forall\, \mathtt{S.State} \bullet S^\star_{tr} \Rightarrow 0 \leq k + \sum_{tr\uparrow L} - \#(tr \downarrow L)$
3. $\forall\, \mathtt{S.State} \bullet S^\star_{tr^\frown u} \Rightarrow \#u \leq k + \sum_{tr\uparrow L} - \#(tr \downarrow L)$
4. *Ist* S *sogar L-terminal, so gilt mit* $\kappa = max\{k_c \mid c \in Chans(\mathcal{I}[\![\mathtt{S}]\!]) \setminus L\}$
   $\forall\, \mathtt{S.State} \bullet S^\star_{tr^\frown u} \Rightarrow \#u \leq k + \kappa \cdot \#(tr \uparrow L) - \#(tr \downarrow L)$

Beweis. Sei *term* ein Schema und $k_c : type_{\mathcal{I}}(c) \to \mathbb{N}$ für $c \in Chans(\mathcal{I}[\![\mathtt{S}]\!]) \setminus L$ totale Funktionen wie in Definition 5.2 angegeben.

Aussage 1 wird durch Induktion über die Länge von Traces bewiesen. Sei $tr : \operatorname{seq} Comm(Chans(\mathcal{I}[\![\mathtt{S}]\!]))$.

Wenn $tr = \langle\rangle$ ist, dann gilt nach 5.2.2

$$\forall\, term \bullet S^\star_{\langle\rangle} \Rightarrow 0 \leq t \leq k + \textstyle\sum_{\langle\rangle\uparrow L} - \#(\langle\rangle \downarrow L),$$

denn es ist $\sum_{\langle\rangle} = 0$ und $\#\langle\rangle = 0$.

Sei $tr = (sr \frown \langle(c, v)\rangle)$. Sei p : S.State ein Zustand und $\alpha : \mathbb{Z}$. Es gelte $(S^\star_{tr} \wedge term)[p/\mathtt{State}, \alpha/t]$, wobei die vorkommenden Schemata als Prädikate aufgefaßt werden. Dann gibt es einen Zustand q : S.State und wegen 5.2.1 eine Zahl $\beta : \mathbb{Z}$, so daß

$$(S^\star_{sr} \wedge term \wedge \mathtt{S.com\_}\, c \wedge term')[p/\mathtt{State}', \alpha/t', q/\mathtt{State}, \beta/t, v/@c].$$

Nach Induktionsvoraussetzung folgt

$$\left(\begin{array}{l} 0 \le t \le k + \sum_{sr\uparrow L} - \#(sr \downarrow L) \\ \wedge\, \mathtt{S.com}\_\, c \wedge term' \end{array}\right) \left[\begin{array}{l} p/\mathtt{State}', \alpha/t', \\ q/\mathtt{State}, \beta/t, v/@c \end{array}\right] \tag{5.1}$$

Es sind zwei Fälle zu unterscheiden:

1. Fall: $c \notin L$**.**
Mit 5.2.3 folgt aus (5.1)

$$\left(\begin{array}{l} 0 \le t \le k + \sum_{sr\uparrow L} - \#(sr \downarrow L) \\ \wedge\, 0 \le t' \le t + k_c(@c) \end{array}\right) [\alpha/t', \beta/t, v/@c]$$

Und damit folgt die Behauptung, denn

$$\begin{array}{ll} & \left(\begin{array}{l} 0 \le t \le k + \sum_{sr\uparrow L} - \#(sr \downarrow L) \\ \wedge\, 0 \le t' \le t + k_c(v) \end{array}\right) [\alpha/t', \beta/t] \\ \Rightarrow & \left(0 \le t' \le k + \sum\nolimits_{sr\uparrow L} + k_c(v) + \#(sr \downarrow L)\right) [\alpha/t'] \\ \Rightarrow & \left(0 \le t' \le k + \sum\nolimits_{tr\uparrow L} + \#(tr \downarrow L)\right) [\alpha/t'] \\ & [\text{da } c \notin L] \end{array}$$

2. Fall: $c \in L$**.**
Mit 5.2.4 folgt aus (5.1)

$$\left(\begin{array}{l} 0 \le t \le k + \sum_{sr\uparrow L} - \#(sr \downarrow L) \\ \wedge\, 0 \le t' < t \end{array}\right) [\alpha/t', \beta/t, v/@c]$$

Die Behauptung folgt, da

$$\begin{array}{ll} & \left(\begin{array}{l} 0 \le t \le k + \sum_{sr\uparrow L} - \#(sr \downarrow L) \\ \wedge\, 0 \le t' < t \end{array}\right) [\alpha/t', \beta/t] \\ \Rightarrow & \left(0 \le t' < k + \sum\nolimits_{sr\uparrow L} - \#(sr \downarrow L)\right) [\alpha/t'] \\ \Rightarrow & \left(0 \le t' \le k + \sum\nolimits_{tr\uparrow L} - \#(tr \downarrow L)\right) [\alpha/t'] \\ & [\text{da } c \in L] \end{array}$$

Die 2. Behauptung ist eine triviale Folgerung aus Behauptung 1. Zum Beweis der 3. Behauptung sei $u : \text{seq } Comm(L)$.

$$\begin{array}{rl} & \forall\, \mathtt{S.State} \bullet S^{\star}_{tr^\frown u} \Rightarrow 0 \leq k + \sum_{(tr^\frown u)\uparrow L} - \#((tr^\frown u) \downarrow L) \\ & [\text{nach Behauptung 2}] \\ \Leftrightarrow & \forall\, \mathtt{S.State} \bullet S^{\star}_{tr^\frown u} \Rightarrow 0 \leq k + \sum_{tr\uparrow L} -\#(tr \downarrow L) - \#(u \downarrow L) \\ \Leftrightarrow & \forall\, \mathtt{S.State} \bullet S^{\star}_{tr^\frown u} \Rightarrow \#(u \downarrow L) \leq k + \sum_{tr\uparrow L} - \#(tr \downarrow L) \end{array}$$

Sei $\kappa = max\{k_c \mid c \in Chans((\mathcal{I}[\![\mathtt{S}]\!]) \setminus L)\}$. Dann folgt Behauptung 4 wegen

$$\begin{array}{rl} & \forall\, \mathtt{S.State} \bullet S^{\star}_{tr^\frown u} \Rightarrow \#(u \downarrow L) \leq k + \sum_{tr\uparrow L} - \#(tr \downarrow L) \\ & [\text{nach Behauptung 3}] \\ \Rightarrow & \forall\, \mathtt{S.State} \bullet S^{\star}_{tr^\frown u} \Rightarrow \#(u \downarrow L) \leq k + \kappa \cdot \#(tr \uparrow L) - \#(tr \downarrow L) \end{array}$$

□

Es kann sofort notiert werden, daß schwach terminale Spezifikationen auch beschränkt sind. Diese Eigenschaft wird später von Bedeutung sein.

Bemerkung 5.4 *Ist* $\mathtt{S} = \mathtt{spec\ I;\ Z^S\ end\_spec} \in SPEC(\mathcal{I})$ *eine schwach L-terminale Spezifikation, dann ist sie auch L-beschränkt, dh.* $\mathcal{D}[\![\mathtt{S} \setminus L]\!] = \varnothing$.

Beweis. Die Behauptung folgt aus 5.3.3. □

Die Simulationen, die in diesem Abschnitt definiert werden, setzen alle voraus, daß die Zielspezifikation einer Verfeinerung (mindestens) schwach terminal ist. Dadurch ist ausgeschlossen, daß Divergenz eingeführt wird.

Definition 5.5 (Vorwärtssimulation "WLIFS")
Seien $\mathtt{A} = \mathtt{spec\ I;\ Z^A\ end\_spec}$ *und* C *zwei Spezifikationen aus* $SPEC(\mathcal{I})$ *mit* $\mathcal{L}[\![\mathtt{A}]\!] = \varnothing$ *und* $\mathcal{L}[\![\mathtt{C}]\!] = L$. *Der* CSP*-Teil beider Spezifikationen sei leer.*

Ist C *schwach L-terminal, dann heißt ein Repräsentationsschema Rep ein WLIFS-*Schema *von* A *nach* C*, wenn*

1. **(Initialisierung)**
   $\forall\, \mathtt{C.State'} \bullet \exists\, \mathtt{A.State'} \bullet$
   $\mathtt{C.Init\_State} \Rightarrow \mathtt{A.Init\_State} \wedge Rep'$

2. **(Definiertheit)**
   Für alle Kanäle $c \in Chans(\mathcal{I}[\![\mathtt{A}]\!])$ *gilt:*
   $\forall\, \mathtt{C.State};\ \mathtt{A.State};\ @c : type_{\mathcal{I}}(c) \bullet$
   $Rep \wedge \mathrm{pre}\, \mathtt{A.com\_}\, c \Rightarrow$
   $\mathrm{pre}\, \mathtt{C.com\_}\, c \vee (\exists\, \ell \in Chans(L) \bullet \exists\, @\ell : type_L \bullet \mathrm{pre}\, \mathtt{C.com\_}\, \ell)$

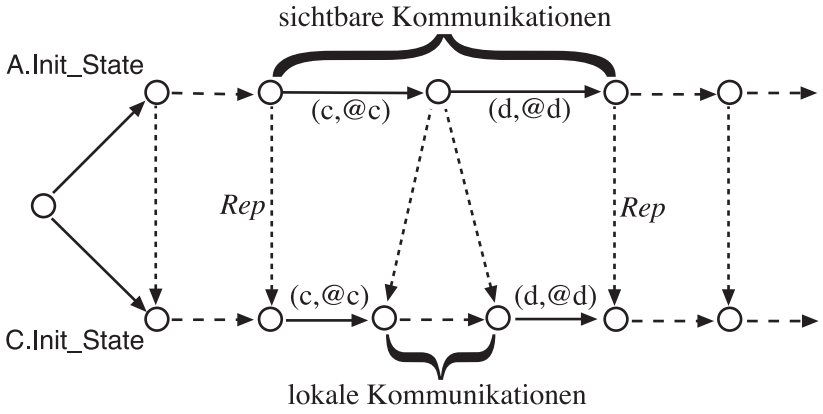


Abbildung 5.1: Vorwärtssimulation

3. **(sichtbarer Fortschritt)**
   Für alle Kanäle $c \in Chans(\mathcal{I}[\![\mathtt{A}]\!])$ *gilt:*
   $\forall \Delta \mathtt{C.State};\ \mathtt{A.State};\ @c : type_{\mathcal{I}}(c) \bullet$
   $Rep \wedge \mathtt{C.com\_}\ c \Rightarrow (\exists\, \mathtt{A.State}' \bullet \mathtt{A.com\_}\ c \wedge Rep')$

4. **(unsichtbarer Fortschritt)**
   Für alle Kanäle $\ell \in Chans(L)$ *gilt:*
   $\forall \Delta \mathtt{C.State};\ \mathtt{A.State};\ @\ell : type_L \bullet$
   $Rep \wedge \mathtt{C.com\_}\ \ell \Rightarrow (\exists\, \mathtt{A.State}' \bullet \Xi \mathtt{A.State} \wedge Rep')$

Existiert ein WLIFS-Schema Rep von A *nach* C*, so wird dies durch* $\mathtt{A} \sim_{WLIFS} \mathtt{C}$ *notiert.* □

Die Aussagen 5.5.1 und 5.5.3 sind bereits aus Definition 3.1 bekannt. Bedingung 5.5.2 bewirkt, daß die Operationen der lokalen Kanäle kein sichtbares zusätzliches Verhalten erzeugen. Die zu den lokalen Kanälen gehörigen Operationen führen nur nach 5.5.4 Zustandsübergänge herbei, die in den durch *Rep* zugeordneten abstrakten Zuständen keine Veränderung bedeuten. Abbildung 5.1 zeigt wie die Vorwärtssimulation die Zustände der abstrakten und der konkreten Spezifikation miteinander in Beziehung setzt. Oben sind die Zustandsübergänge der abstrakten Spezifikation zu sehen und unten die der konkreten Spezifikation.

Definition 5.6 (Rückwärtssimulation "WLIBS")
Seien $\mathtt{A} = \mathtt{spec\ I;\ Z^A\ end\_spec}$ *und* C *zwei Spezifikationen aus* $SPEC(\mathcal{I})$ *mit* $\mathcal{L}[\![\mathtt{A}]\!] = \varnothing$ *und* $\mathcal{L}[\![\mathtt{C}]\!] = L$*. Der* CSP*-Teil beider Spezifikationen sei leer.*

Ist C *schwach L-terminal, dann heißt ein Repräsentationsschema Rep ein WLIBS-*Schema *von* A *nach* C*, wenn*

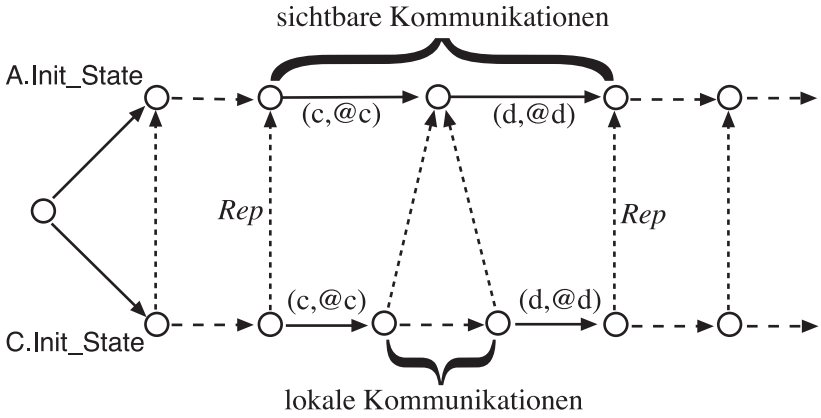


Abbildung 5.2: Rückwärtssimulation

1. **(Initialisierung)**
   $\forall \mathtt{C.State}'; \mathtt{A.State}' \bullet$
   $\mathtt{C.Init\_State} \wedge \mathit{Rep}' \Rightarrow \mathtt{A.Init\_State}$

2. **(Definiertheit)**
   Für alle Kanäle $c \in \mathit{Chans}(\mathcal{I}[\![\mathtt{A}]\!])$ *gilt:*
   $\forall \mathtt{C.State}; @c : \mathit{type}_{\mathcal{I}}(c) \bullet \exists \mathtt{A.State} \bullet$
   $\mathit{Rep} \wedge$
   $($ pre $\mathtt{A.com\_}\, c \Rightarrow$
   pre $\mathtt{C.com\_}\, c \vee (\exists \ell \in \mathit{Chans}(L) \bullet \exists @\ell : \mathit{type}_L \bullet$ pre $\mathtt{C.com\_}\, \ell))$

3. **(sichtbarer Fortschritt)**
   Für alle Kanäle $c \in \mathit{Chans}(\mathcal{I}[\![\mathtt{A}]\!])$ *gilt:*
   $\forall \Delta\mathtt{C.State}; \mathtt{A.State}'; @c : \mathit{type}_{\mathcal{I}}(c) \bullet$
   $\mathtt{C.com\_}\, c \wedge \mathit{Rep}' \Rightarrow (\exists \mathtt{A.State} \bullet \mathit{Rep} \wedge \mathtt{A.com\_}\, c)$

4. **(unsichtbarer Fortschritt)**
   Für alle Kanäle $\ell \in \mathit{Chans}(L)$ *gilt:*
   $\forall \Delta\mathtt{C.State}; \mathtt{A.State}'; @\ell : \mathit{type}_L \bullet$
   $\mathtt{C.com\_}\, \ell \wedge \mathit{Rep}' \Rightarrow (\exists \mathtt{A.State} \bullet \mathit{Rep} \wedge \Xi\mathtt{A.State})$

Existiert ein WLIBS-Schema Rep von A *nach* C*, so wird* $\mathtt{A} \backsim_{WLIBS} \mathtt{C}$ *notiert.* □

Die Bedingungen 5.6.1 und 5.6.3 sind identisch mit denen der Definition 3.2 und wurden dort bereits erläutert. Aussage 5.6.2 besagt, daß es zu jedem konkreten Zustand mindestens einen abstrakten Zustand gibt, und daß die lokalen Kanäle nicht zu mehr sichtbarem Verhalten führen. Die letzte

Aussage 5.6.4 bedeutet, daß die konkreten lokalen Operationen keine Auswirkungen auf entsprechende abstrakte Zustände haben. In Abbildung 5.2 ist zu sehen wie die Zustände der abstrakten und der konkreten Spezifikation bei der Rückwärtssimulation miteinander in Beziehung stehen.

Die beiden nächsten Regeln schränken die Anwendbarkeit der Regeln **WLIFS** und **WLIBS** auf eine kleinere Anzahl von Spezifikationen ein. Später wird sich herausstellen, das diese Einschränkung dazu führt, daß lokale Kanäle, die mit diesen stärkeren Regeln eingeführt wurden, sich bis zu einem gewissen Grade kommutativ verhalten.

Definition 5.7 (Vorwärtssimulation "LIFS")
Ist in Definition 5.5 die Spezifikation C *sogar L-terminal und ist Rep ein WLIFS-Schema von* A *nach* C*, dann heißt Rep ein LIFS-*Schema*. Existiert ein LIFS-Schema von* A *nach* C*, so wird* $\mathtt{A} \sim_{LIFS} \mathtt{C}$ *geschrieben.* □

Definition 5.8 (Rückwärtssimulation "LIBS")
Ist in Definition 5.6 die Spezifikation C *sogar L-terminal und ist Rep ein WLIBS-Schema von* A *nach* C*, dann heißt Rep ein LIBS-*Schema*. Existiert ein LIBS-Schema von* A *nach* C*, so wird* $\mathtt{A} \sim_{LIBS} \mathtt{C}$ *geschrieben.* □

5.3 Korrektheit von WLIFS und WLIBS

Die Beweise zur Korrektheit der Regeln verlaufen völlig analog zu denen in Abschnitt 3.2. Zum Beweis, daß keine der beiden Regeln Divergenzen einführt, werden die Eigenschaften schwach terminaler Spezifikationen ausgenutzt, die im letzten Abschnitt bewiesen wurden.

Lemma 5.9 *Seien* $\mathtt{A} = \mathtt{spec\ I;\ Z^A\ end\_spec}$ *und* $\mathtt{C} \setminus L$ *zwei Spezifikationen aus* $SPEC(\mathcal{I})$ *ohne* CSP*-Teil mit* $L = Chans(\mathcal{L}[\![\mathtt{C} \setminus L]\!])$ *und* C *schwach L-terminal. Sei Rep ein Repräsentationsschema.*

1. *Gelten die Bedingungen 5.5.1, 5.5.3 und 5.5.4, dann folgt für alle* tr : seq $Comm(Chans(\mathcal{I}) \cup L)$:

$$\forall\, \mathtt{C.State} \bullet \exists\, \mathtt{A.State} \bullet \mathtt{C}^\star_{tr} \Rightarrow \mathtt{A}^\star_{tr\uparrow L} \wedge Rep.$$

2. *Gilt außerdem 5.5.2, dann folgt*

$$\mathcal{F}[\![\mathtt{C} \setminus L]\!] \subseteq \mathcal{F}[\![\mathtt{A}]\!]$$

Beweis. Die erste Behauptung wird durch Induktion über die Länge der Sequenzen aus seq $Comm(Chans(\mathcal{I}) \cup L)$ bewiesen. Der Induktionsanfang ist bereits in Lemma 3.4 bewiesen worden. Der Induktionsschritt ist in zwei Fälle untergliedert. Sei $tr \frown \langle (c, v) \rangle$: seq $Comm(Chans(\mathcal{I}) \cup L)$. Ist $c \notin L$,

dann folgt die Behauptung analog zu dem Induktionsschritt in Lemma 3.4, da $(tr \upharpoonright L) \frown \langle (c,v) \rangle = (tr \frown \langle (c,v) \rangle) \upharpoonright L$. Die Behauptung ist auch für den anderen Fall $c \in L$ wahr, denn

$$
\begin{array}{ll}
 & \forall\, \mathtt{C.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \Rightarrow (\exists\, \mathtt{C.State} \bullet \mathtt{C}^{\star}_{tr} \wedge \mathtt{C.com\_}(c,v)) \\
\Rightarrow & \forall\, \mathtt{C.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \Rightarrow \\
 & \qquad (\exists\, \mathtt{C.State};\ \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr \upharpoonright L} \wedge \mathit{Rep} \wedge \mathtt{C.com\_}(c,v)) \\
 & [\text{nach Induktionshypothese}] \\
\Rightarrow & \forall\, \mathtt{C.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \Rightarrow \\
 & \qquad (\exists\, \Delta \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr \upharpoonright L} \wedge \Xi \mathtt{A.State} \wedge \mathit{Rep}') \\
 & [\text{nach 5.5.4}] \\
\Leftrightarrow & \forall\, \mathtt{C.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \Rightarrow \\
 & \qquad (\exists\, \mathtt{A.State}' \bullet \mathtt{A}_{(tr \frown \langle (c,v) \rangle) \upharpoonright L} \wedge \mathit{Rep}') \\
 & [\text{da } c \in L] \\
\Rightarrow & \forall\, \mathtt{C.State} \bullet \mathtt{C}^{\star}_{tr \frown \langle (c,v) \rangle} \Rightarrow \\
 & \qquad (\exists\, \mathtt{A.State} \bullet \mathtt{A}^{\star}_{(tr \frown \langle (c,v) \rangle) \upharpoonright L} \wedge \mathit{Rep})
\end{array}
$$

Zur zweiten Behauptung.

$$
\begin{array}{ll}
 & (tr, R) \in \mathcal{F}[\![\mathtt{C} \setminus L]\!] \\
\Leftrightarrow & \exists\, sr : \mathrm{seq}\, \mathit{Comm}(\mathit{Chans}(\mathcal{I} \cup L)) \bullet \\
 & \qquad \exists\, \mathtt{C.State} \bullet sr \upharpoonright L = tr \wedge \mathtt{C}_{sr} \wedge \mathit{Ref}_{\mathtt{C}}\,(R \cup \mathcal{L}[\![\mathtt{C}]\!]) \\
 & [\text{da nach Bemerkung 5.4 gilt: } \mathcal{D}[\![\mathtt{C} \setminus L]\!] = \varnothing] \\
\Rightarrow & \exists\, sr : \mathrm{seq}\, \mathit{Comm}(\mathit{Chans}(\mathcal{I} \cup L)) \bullet \\
 & \qquad \exists\, \mathtt{C.State};\ \mathtt{A.State} \bullet sr \upharpoonright L = tr \wedge \mathtt{A}_{sr \upharpoonright L} \wedge \mathit{Rep} \wedge \mathit{Ref}_{\mathtt{C}}\,(R \cup \mathcal{L}[\![\mathtt{C}]\!]) \\
 & [\text{nach 1.}] \\
\Rightarrow & \exists\, sr : \mathrm{seq}\, \mathit{Comm}(\mathit{Chans}(\mathcal{I} \cup L)) \bullet \\
 & \qquad \exists\, \mathtt{A.State} \bullet sr \upharpoonright L = tr \wedge \mathtt{A}_{sr \upharpoonright L} \wedge \mathit{Ref}_{\mathtt{A}}\, R \\
 & [\text{nach 5.5.2}] \\
\Leftrightarrow & \exists\, \mathtt{A.State} \bullet \mathtt{A}_{tr} \wedge \mathit{Ref}_{\mathtt{A}}\, R \\
\Leftrightarrow & (tr, R) \in \mathcal{F}[\![\mathtt{A}]\!]
\end{array}
$$

□

Die Korrektheit von **WLIFS** folgt nun sofort wegen der Divergenzfreiheit der konkreten Spezifikation. Die Korrektheit wird hier nur für Spezifikationen mit leerem CSP-Teil bewiesen, da es wenig sinnvoll scheint die abstrakte und die konkrete Spezifikation mit demselben CSP-Prozeß parallel zu komponieren. Denn dann würden gar keine lokalen Kommunikationen

in der konkreten Spezifikation auftreten können. Abgesehen davon gilt im allgemeinen *nicht* (siehe [Hoa85, Seite 112]):

$$(\mathtt{S} \parallel \mathtt{T}) \setminus L \equiv (\mathtt{S} \setminus L) \parallel (\mathtt{T} \setminus L).$$

Satz 5.10 (Korrektheit von WLIFS) *Für* A,C $\in SPEC(\mathcal{I})$ *mit leerem* CSP*-Teil gilt:*

$$\mathtt{A} \sim_{WLIFS} \mathtt{C} \Rightarrow \mathtt{A} \sqsubseteq \mathtt{C}.$$

Beweis. Es gilt $\mathcal{D}[\![\mathtt{C}]\!] = \varnothing$ nach Bemerkung 5.4 und per Definition $\mathcal{D}[\![\mathtt{A}]\!] = \varnothing$. Wegen 5.5.2 gilt $\mathcal{F}[\![\mathtt{C}]\!] \subseteq \mathcal{F}[\![\mathtt{A}]\!]$. □

Gilt $\mathtt{A} \backsim_{WLIBS} \mathtt{C}$, dann folgt die Failures-Inklusion ganz ähnlich wie in Kapitel 3.

Lemma 5.11 *Seien* $\mathtt{A} = \mathtt{spec\ I};\ \mathtt{Z}^{\mathtt{A}}\ \mathtt{end\_spec}$ *und* $\mathtt{C} \setminus L$ *zwei Spezifikationen aus* $SPEC(\mathcal{I})$ *ohne* CSP*-Teil mit* $L = Chans(\mathcal{L}[\![\mathtt{C} \setminus L]\!])$ *und* C *schwach L-terminal. Sei Rep ein Repräsentationsschema.*

1. *Gelten die Bedingungen 5.6.1, 5.6.3 und 5.6.4, so auch für alle tr* : seq $Comm(Chans(\mathcal{I}) \cup L)$:

$$\exists\, \mathtt{C.State};\ \mathtt{A.State} \bullet \mathtt{C}^{\star}_{tr} \wedge Rep \Rightarrow \mathtt{A}^{\star}_{tr \upharpoonright L}$$

2. *Gilt außerdem 5.6.2, dann folgt*

$$\mathcal{F}[\![\mathtt{C} \setminus L]\!] \subseteq \mathcal{F}[\![\mathtt{A}]\!]$$

Beweis. Analog zu Lemma 5.9 läßt sich auch hier der Beweis zum Lemma 3.6 wiederverwenden. Die erste Behauptung wird durch Induktion über die Länge von Traces geführt. Der Induktionsanfang ist identisch mit dem im Beweis zu Lemma 3.6. Im Induktionsschritt werden die lokalen Kanäle von C und die Kanäle des Interfaces getrennt behandelt. Sei $tr \frown \langle (c, v) \rangle$: seq $Comm(Chans(\mathcal{I}) \cup L)$. Ist $c \notin L$, so folgt die Behauptung genauso wie in Lemma 3.6. Ist $c \in L$, so folgt die Behautung, da

$$\begin{aligned}
& \forall\, \mathtt{C.State}';\ \mathtt{A.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \wedge Rep' \Rightarrow \\
& \qquad (\exists\, \mathtt{C.State} \bullet \mathtt{C}^{\star}_{tr} \wedge \mathtt{C.com_}\,(c, v) \wedge Rep') \\
\Rightarrow\ & \forall\, \mathtt{C.State}';\ \mathtt{A.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \wedge Rep' \Rightarrow \\
& \qquad (\exists\, \mathtt{C.State};\ \mathtt{A.State} \bullet \mathtt{C}^{\star}_{tr} \wedge Rep \wedge \Xi \mathtt{A.State}) \\
& \text{[nach Voraussetzung 5.6.4]} \\
\Rightarrow\ & \forall\, \mathtt{C.State}';\ \mathtt{A.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \wedge Rep' \Rightarrow \\
& \qquad (\exists\, \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr \upharpoonright L} \wedge \Xi \mathtt{A.State}) \\
& \text{[nach Induktionshypothese]} \\
\Leftrightarrow\ & \forall\, \mathtt{C.State}';\ \mathtt{A.State}' \bullet \mathtt{C}_{tr \frown \langle (c,v) \rangle} \wedge Rep' \Rightarrow \mathtt{A}_{(tr \frown \langle (c,v) \rangle) \upharpoonright L} \\
& \text{[da } c \in L\text{]} \\
\Leftrightarrow\ & \forall\, \mathtt{C.State};\ \mathtt{A.State} \bullet \mathtt{C}^{\star}_{tr \frown \langle (c,v) \rangle} \wedge Rep \Rightarrow \mathtt{A}^{\star}_{(tr \frown \langle (c,v) \rangle) \upharpoonright L}
\end{aligned}$$

Zum Beweis der zweiten Behauptung.

$$
\begin{array}{ll}
 & (tr, R) \in \mathcal{F}[\![\mathtt{C} \setminus L]\!] \\
\Leftrightarrow & \exists\, sr : \mathrm{seq}\, Comm(Chans(\mathcal{I} \cup L)) \bullet \\
 & \qquad \exists\, \mathtt{C.State} \bullet sr \uparrow L = tr \wedge \mathtt{C}_{sr} \wedge Ref_{\mathtt{C}}\,(R \cup \mathcal{L}[\![\mathtt{C}]\!]) \\
 & [\text{da nach Bemerkung 5.4 gilt: } \mathcal{D}[\![\mathtt{C} \setminus L]\!] = \varnothing] \\
\Rightarrow & \exists\, sr : \mathrm{seq}\, Comm(Chans(\mathcal{I} \cup L)) \bullet \\
 & \qquad \exists\, \mathtt{C.State};\ \mathtt{A.State} \bullet sr \uparrow L = tr \wedge \mathtt{C}_{sr} \wedge Rep \wedge Ref_{\mathtt{A}}\, R \\
 & [\text{nach Voraussetzung 5.6.2}] \\
\Rightarrow & \exists\, sr : \mathrm{seq}\, Comm(Chans(\mathcal{I} \cup L)) \bullet \\
 & \qquad \exists\, \mathtt{A.State} \bullet sr \uparrow L = tr \wedge \mathtt{A}_{sr \uparrow L} \wedge Ref_{\mathtt{A}}\, R \\
 & [\text{nach 1.}] \\
\Leftrightarrow & \exists\, \mathtt{A.State} \bullet \mathtt{A}_{tr} \wedge Ref_{\mathtt{A}}\, R \\
\Leftrightarrow & (tr, R) \in \mathcal{F}[\![\mathtt{A}]\!]
\end{array}
$$

□

Da **WLIBS** verlangte, daß die konkrete Spezifikation divergenzfrei ist, folgt die Korrektheit von **WLIBS** mit dem letzten Lemma.

Satz 5.12 (Korrektheit von WLIBS) *Für* A,C $\in SPEC(\mathcal{I})$ *mit leerem* CSP*-Teil gilt:*

$$\mathtt{A} \sim_{WLIBS} \mathtt{C} \Rightarrow \mathtt{A} \sqsubseteq \mathtt{C}.$$

Beweis. Nach Lemma 5.11 gilt $\mathcal{F}[\![\mathtt{C}]\!] \subseteq \mathcal{F}[\![\mathtt{A}]\!]$. Und wegen Bemerkung 5.4 ist $\mathcal{D}[\![\mathtt{C}]\!] = \varnothing$. Also folgt die Behauptung. □

5.4 Kommutativität

Sei $\mathtt{D} \in SPEC(\mathcal{I})$ eine Spezifikation und $H, L \subseteq Chans(\mathcal{I})$ mit $H \cap L = \varnothing$. Der Hiding-Operator ($\setminus$) heißt (H, L)-*kommutativ* (für D), wenn gilt:

$$\mathtt{D} \setminus L \setminus H \equiv \mathtt{D} \setminus H \setminus L \equiv \mathtt{D} \setminus (H \cup L).$$

Die (H, L)-Kommutativität von $\setminus$ für eine Spezifikation D folgt bereits aus der entsprechenden Eigenschaft der Divergenzen von D.

Lemma 5.13 *Sei* D *eine Spezifikationen aus* $SPEC(\mathcal{I})$*, und seien* $H, L \subseteq \mathcal{I}$ *mit* $H \cap L = \varnothing$*. Wenn gilt:*

$$\mathcal{D}[\![\mathtt{D} \setminus H \setminus L]\!] = \mathcal{D}[\![\mathtt{D} \setminus L \setminus H]\!] = \mathcal{D}[\![\mathtt{D} \setminus (H \cup L)]\!] \tag{5.2}$$

dann folgt $\mathcal{F}[\![\mathtt{D} \setminus H \setminus L]\!] = \mathcal{F}[\![\mathtt{D} \setminus L \setminus H]\!] = \mathcal{F}[\![\mathtt{D} \setminus (H \cup L)]\!]$.

Beweis.

$$
\begin{aligned}
& \mathcal{F}[\![\mathtt{D} \setminus L \setminus H]\!] \\
= \quad & \{(s \uparrow H, X) \mid (s, X \cup \mathit{Comm}(H)) \in \mathcal{F}[\![\mathtt{D} \setminus L]\!]\} \\
& \cup \{(s, X) \mid s \in \mathcal{D}[\![\mathtt{D} \setminus L \setminus H]\!]\} \\
& [\text{Definition von } \mathcal{F}] \\
= \quad & \{(s \uparrow H, X) \mid \\
& \quad (s, X \cup \mathit{Comm}(H)) \in \{(t \uparrow L, Y) \mid (t, Y \cup \mathit{Comm}(L)) \in \mathcal{F}[\![\mathtt{D}]\!]\}\} \\
& \cup \{(s \uparrow H, X) \mid (s, X \cup \mathit{Comm}(H)) \in \{(t, Y) \mid t \in \mathcal{D}[\![\mathtt{D} \setminus L]\!]\}\} \\
& \cup \{(s, X) \mid s \in \mathcal{D}[\![\mathtt{D} \setminus L \setminus H]\!]\} \\
& [\text{Definition von } \mathcal{F}] \\
= \quad & \{(s \uparrow H \uparrow L, X) \mid (s, X \cup \mathit{Comm}(H) \cup \mathit{Comm}(L)) \in \mathcal{F}[\![\mathtt{D}]\!]\} \\
& \cup \{(s \uparrow H, X) \mid s \in \mathcal{D}[\![\mathtt{D} \setminus L]\!]\} \\
& \cup \{(s, X) \mid s \in \mathcal{D}[\![\mathtt{D} \setminus L \setminus H]\!]\} \\
& [\text{Eigenschaft von } \mathcal{D} \text{ und da } \mathcal{D}[\![\mathtt{D} \setminus L \setminus H]\!] = \mathcal{D}[\![\mathtt{D} \setminus (H \cup L)]\!]] \\
= \quad & \{(s \uparrow (H \cup L), X) \mid (s, X \cup \mathit{Comm}(H \cup L)) \in \mathcal{F}[\![\mathtt{D}]\!]\} \\
& \cup \{(s, X) \mid s \in \mathcal{D}[\![\mathtt{D} \setminus H \setminus L]\!]\} \\
& [\text{da nach Definition von } \mathcal{D} \text{ gilt: } \{s \uparrow H \mid s \in \mathcal{D}[\![\mathtt{D} \setminus L]\!]\} \subseteq \mathcal{D}[\![\mathtt{D} \setminus H \setminus L]\!]] \\
= \quad & \{(s \uparrow (H \cup L), X) \mid (s, X \cup \mathit{Comm}(H \cup L)) \in \mathcal{F}[\![\mathtt{D}]\!]\} \\
& \cup \{(s, X) \mid s \in \mathcal{D}[\![\mathtt{D} \setminus (H \cup L)]\!]\} \\
& [\text{nach (5.2). Damit ist } \mathcal{F}[\![\mathtt{D} \setminus L \setminus H]\!] = \mathcal{F}[\![\mathtt{D} \setminus (H \cup L)]\!] \text{ bewiesen.}] \\
= \quad & \{(s \uparrow L \uparrow H, X) \mid (s, X \cup \mathit{Comm}(H \cup L)) \in \mathcal{F}[\![\mathtt{D}]\!]\} \\
& \cup \{(s \uparrow L, X) \mid s \in \mathcal{D}[\![\mathtt{D} \setminus H]\!]\} \\
& \cup \{(s, X) \mid s \in \mathcal{D}[\![\mathtt{D} \setminus H \setminus L]\!]\} \\
& [\text{nach (5.2), und Definition von } \mathcal{D}] \\
= \quad & \{(s \uparrow L, X) \mid (s, X \cup \mathit{Comm}(L)) \in \mathcal{F}[\![\mathtt{D} \setminus H]\!]\} \\
& \cup \{(s, X) \mid s \in \mathcal{D}[\![\mathtt{D} \setminus H \setminus L]\!]\} \\
& [\text{Definition von } \mathcal{F} \text{ und da } \mathcal{D}[\![\mathtt{D} \setminus H \setminus L]\!] = \mathcal{D}[\![\mathtt{D} \setminus (H \cup L)]\!]] \\
= \quad & \mathcal{F}[\![\mathtt{D} \setminus H \setminus L]\!]
\end{aligned}
$$

□

Sind die Divergenzen einer Spezifikation $\mathtt{D} \setminus H \setminus L$ leer, dann ist $\setminus$ folglich (H, L)-kommutativ für $\mathtt{D}$.

Lemma 5.14 *Gilt für eine Spezifikation* $\mathtt{D}$*, daß* $\mathcal{D}[\![\mathtt{D} \setminus (H \cup L)]\!] = \varnothing$*, dann ist der Hiding-Operator* (H, L)*-kommutativ für* $\mathtt{D}$*.*

Beweis. Die Behauptung folgt mit Lemma 5.13, da

$$\mathcal{D}[\![\mathtt{D} \setminus L \setminus H]\!] \subseteq \mathcal{D}[\![\mathtt{D} \setminus (H \cup L)]\!]$$

und

$$\mathcal{D}[\![\mathtt{D} \setminus H \setminus L]\!] \subseteq \mathcal{D}[\![\mathtt{D} \setminus (H \cup L)]\!].$$

□

Aus $\mathtt{D} \setminus L \setminus H \equiv \mathtt{D} \setminus (L \cup H)$ folgt nicht unbedingt auch $\mathtt{D} \setminus H \setminus L \equiv \mathtt{D} \setminus (L \cup H)$. Aber bereits eine der beiden Äquivalenzen kann nützlich sein, um lokale Kanäle zusammenzufassen, die nacheinander eingeführt wurden.

Lemma 5.15 (Eingeschränkte Kommutativität) *Sei* D *eine Spezifikationen und* $H, L \subseteq \mathcal{I}$ *mit* $H \cap L = \varnothing$*, und es gelte*

1. $\mathcal{D}[\![\mathtt{D} \setminus L]\!] = \varnothing$
2. $\forall\, t : \mathrm{seq}\, Comm(L \cup H) \bullet$
   t *ist* $H \cup L$*-unbeschränkt in* D $\Rightarrow t \uparrow L$ *ist* H*-unbeschränkt in* $\mathtt{D} \setminus L$

Dann folgt $\mathtt{D} \setminus L \setminus H \equiv \mathtt{D} \setminus (L \cup H)$.

Beweis. Die Divergenzen von $\mathtt{D} \setminus (L \cup H)$ sind definiert durch

$$\begin{aligned} & \mathcal{D}[\![\mathtt{D} \setminus (L \cup H)]\!] \\ = \; & \{s \uparrow (L \cup H) \frown t \mid \forall\, n : \mathbb{N} \bullet \exists\, u : \mathrm{seq}\, Comm(L \cup H) \bullet \\ & \qquad \#u > n \wedge s \frown u \in traces(\mathtt{D})\} \\ & \{s \uparrow (L \cup H) \frown t \mid \forall\, n : \mathbb{N} \bullet \exists\, u : \mathrm{seq}\, Comm(L \cup H) \bullet \\ & \qquad \#u > n \wedge (s \frown u, \varnothing) \in \mathcal{F}[\![\mathtt{D}]\!]\} \end{aligned}$$

Für die Divergenzen von $\mathtt{D} \setminus L \setminus H$ gilt

$$\begin{aligned} & \mathcal{D}[\![\mathtt{D} \setminus L \setminus H]\!] \\ = \; & \{s \uparrow H \mid \forall\, n : \mathbb{N} \bullet \exists\, u : \mathrm{seq}\, Comm(H) \bullet \\ & \qquad \#u > n \wedge s \frown u \in traces(\mathtt{D} \setminus L)\} \\ & \text{[nach Voraussetzung 1]} \\ = \; & \{s \uparrow H \mid \forall\, n : \mathbb{N} \bullet \exists\, u : \mathrm{seq}\, Comm(H) \bullet \\ & \qquad \#u > n \wedge (s \frown u, Comm(L)) \in \mathcal{F}[\![\mathtt{D}]\!]\} \\ & \text{[wieder nach Voraussetzung 1]} \end{aligned}$$

Sei $t \in \mathcal{D}[\![\mathtt{D} \setminus (L \cup H)]\!]$ und $s \uparrow (L \cup H)$ ein Präfix von t, das in $traces(\mathtt{D})$ enthalten ist, so daß s eine $H \cup L$-unbeschränkte Trace in D ist. Nach 2 ist $s \uparrow L$ dann H-unbeschränkt in $\mathtt{D} \setminus L$. Also liegt $s \uparrow (L \cup H)$ in $\mathcal{D}[\![\mathtt{D} \setminus L \setminus H]\!]$ und damit auch t. Der Rest folgt mit Lemma 5.13 mit der Einschränkung, daß nur $\mathtt{D} \setminus L \setminus H \equiv \mathtt{D} \setminus (L \cup H)]$ bewiesen werden kann. □

Beispiel 5.16 *Seien* `A` *und* `C` *zwei Spezifikationen:*

`spec A`

`channel` $in : \mathbb{F}\,\mathbb{N}$;

`State`

$M : \mathbb{F}\,\mathbb{N}$
$done : \{0, 1\}$

`Init_State`

`State`$'$

$M' = \varnothing \land done' = 0$

`com_` in

Δ`State`
$@in : \mathbb{F}\,\mathbb{N}$

$done = 0 \land done' = 1 \land M' = @in$

`end_spec A`

Der Kanal *in* der Spezifikation `A` liest eine endliche Menge natürlicher Zahlen in die Zustandsvariable `A`.M ein. Die Spezifikation hält dann. Dem Kanal *in* der Spezifikation `C` ist dieselbe Operation zugeordnet, wie dem Kanal *in* der Spezifikation `A`. Die Spezifikation `C` hält aber nicht nach einer Kommunikation über den Kanal *in*. Stattdessen wird über den Kanal *copy* die eingelesene Menge elementweise in eine andere Zustandsvariable N kopiert. Erst nachdem alle Elemente kopiert worden sind, hält `C`.

`spec C`

`channel` $in : \mathbb{F}\,\mathbb{N}$;
`channel` $copy : \mathbb{N}$;

`State`

$M, N : \mathbb{F}\,\mathbb{N}$
$done : \{0, 1\}$

`Init_State`

`State`$'$

$M' = \varnothing \land done' = 0$

`com_` in

Δ`State`
$@in : \mathbb{F}\,\mathbb{N}$

$done = 0 \land done' = 1$
$M' = @in$

`com_` $copy$

Δ`State`
$@copy : \mathbb{N}$

$M \neq \varnothing$
$M = M' \cup \{@copy\}$
$@copy \notin M'$
$N' = N \cup \{@copy\}$

`end_spec C`

Es gilt:

1. $\mathtt{A} \sim_{WLIFS} \mathtt{C} \setminus \{copy\}$
2. $\mathtt{A} \setminus \{in\} \sqsubseteq \mathtt{C} \setminus \{copy\} \setminus \{in\}$
3. $\mathcal{D}[\![\mathtt{C} \setminus \{in\} \setminus \{copy\}]\!] = \mathcal{D}[\![\mathtt{C} \setminus \{in, copy\}]\!] \neq \varnothing$
4. $\mathtt{C} \setminus \{copy\} \setminus \{in\} \not\equiv \mathtt{C} \setminus \{in\} \setminus \{copy\} \equiv \mathtt{C} \setminus \{in, copy\}$

Beweis. Spezifikation C ist schwach $\{copy\}$-beschränkt. Seien das Schema *term* und die Funktion k_{in} folgendermaßen gewählt:

term

$$\mathtt{C.State}$$
$$t : \mathbb{Z}$$
$$t = \#\mathtt{C}.M$$

$$k_{in} : \mathbb{F}\,\mathbb{N} \to \mathbb{N}$$
$$\forall\, x : \mathbb{F}\,\mathbb{N} \bullet k_{in}(x) = \#x$$

Da die Variante t für alle Zustände $\Theta\mathtt{C.State}$ definiert ist, gilt 5.2.1, und mit $k = 0$ gilt 5.2.2. Bedingung 5.2.3 ist wahr, da:

$$\begin{aligned}
&\forall\, \Delta term \bullet \\
&\qquad \mathtt{C}.done = 0 \land \mathtt{C}.done' = 1 \land \mathtt{C}.M' = @in \land t \geq 0 \\
&\Rightarrow \quad \#\mathtt{C}.M' \leq t + \#(@in) \\
&\Rightarrow \quad 0 \leq t' \leq t + k_{in}(@in)
\end{aligned}$$

Die dritte Bedingung 5.2.4 gilt ebenfalls:

$$\begin{aligned}
&\forall\, \Delta term \bullet \\
&\qquad \mathtt{C}.M \neq \varnothing \land \mathtt{C}.M = \mathtt{C}.M' \cup \{@copy\} \land \mathtt{C}.@copy \notin M' \\
&\qquad \land\, \mathtt{C}.K' = \mathtt{C}.K \cup \{@copy\} \land t \geq 0 \\
&\Rightarrow \quad \#\mathtt{C}.M' < t \\
&\Rightarrow \quad 0 \leq t' < t
\end{aligned}$$

Es gilt $\mathtt{A} \sim_{WLIFS} \mathtt{C} \setminus \{copy\}$. Das Repräsentationsschema *Rep* ist ein *WLIFS*-Schema, mit dem diese Behauptung bewiesen werden kann.

Rep

$$\mathtt{A.State}$$
$$\mathtt{C.State}$$
$$\mathtt{A}.M = \mathtt{C}.M \cup \mathtt{C}.K \land \mathtt{A}.done = \mathtt{C}.done$$

Die Bedingungen "Initialisierung" und "Definiertheit" gelten, da die Initialisierungen und die Operationen $\mathtt{com\_}\,in$ beider Spezifikationen identisch sind. Die anderen beiden Bedingungen sind ebenfalls erfüllt:

Sichtbarer Fortschritt:

$$\begin{aligned}
&\forall\, \Delta\mathtt{C.State};\ \mathtt{A.State};\ @in : \mathbb{F}\,\mathbb{N} \bullet \\
&\qquad \mathtt{A}.M = \mathtt{C}.M \cup \mathtt{C}.K \wedge \mathtt{A}.done = \mathtt{C}.done \\
&\qquad \wedge\ \mathtt{C}.done = 0 \wedge \mathtt{C}.done' = 1 \wedge \mathtt{C}.M' = @in \\
&\Rightarrow\ (\exists\, \mathtt{A.State}' \bullet \quad \mathtt{A}.done = 0 \wedge \mathtt{A}.done' = 1 \wedge \mathtt{A}.M' = @in \\
&\qquad\qquad \wedge\ \mathtt{A}.M' = \mathtt{C}.M' \cup \mathtt{C}.K' \wedge \mathtt{A}.done' = \mathtt{C}.done')
\end{aligned}$$

Unsichtbarer Fortschritt:

$$\begin{aligned}
&\forall\, \Delta\mathtt{C.State};\ \mathtt{A.State};\ @in : \mathbb{F}\,\mathbb{N} \bullet \\
&\qquad \mathtt{A}.M = \mathtt{C}.M \cup \mathtt{C}.K \wedge \mathtt{A}.done = \mathtt{C}.done \\
&\qquad \wedge\ \mathtt{C}.M = \mathtt{C}.M' \cup \{@copy\} \wedge \mathtt{C}.@copy \notin M' \wedge \mathtt{C}.K' = \mathtt{C}.K \cup \{@copy\} \\
&\Rightarrow\ (\exists\, \mathtt{A.State}' \bullet \quad \Xi\mathtt{A.State} \\
&\qquad\qquad \wedge\ \mathtt{A}.M' = \mathtt{C}.M' \cup \mathtt{C}.K' \wedge \mathtt{A}.done' = \mathtt{C}.done')
\end{aligned}$$

Also ist auch Behauptung 2 wahr. Da gilt:

$$\mathcal{D}[\![\mathtt{C} \setminus \{in\} \setminus \{copy\}]\!] = \mathcal{D}[\![\mathtt{C} \setminus \{in, copy\}]\!] = \{\langle\rangle\},$$

folgen die beiden letzten Behauptungen. □

Mittels der Regeln **LIFS** und **LIBS** eingeführte lokale Kanäle verhalten sich auf jeden Fall eingeschränkt kommutativ bezüglich aller anderen Kanäle der Spezifikation.

Satz 5.17 *Gilt für zwei Spezifikationen* $\mathtt{A}$ *und* $\mathtt{C} \setminus L$ *aus* $SPEC(\mathcal{I})$ *eine der Beziehungen* $\mathtt{A} \sim_{LIFS} \mathtt{C} \setminus L$ *oder* $\mathtt{A} \sim_{LIBS} \mathtt{C} \setminus L$*, dann folgt für alle* $H \subseteq \mathcal{I}$*:*

$$\begin{aligned}
&\forall\, t : \mathrm{seq}\, Comm(L \cup H) \bullet \\
&\qquad t \textit{ ist } H \cup L\textit{-unbeschränkt in } \mathtt{C} \ \Rightarrow\ t \uparrow L \textit{ ist } H\textit{-unbeschränkt in } \mathtt{C} \setminus L \\
&\textit{und } \mathcal{D}[\![\mathtt{C} \setminus L]\!] = \varnothing.
\end{aligned}$$

Beweis. Sei $s \in \mathrm{seq}\, Comm(Chans(\mathcal{I}) \cup L)$. Nach 5.7.4 gilt:

$$(\exists\, \mathtt{C.State} \bullet \mathtt{C}^{\star}_{s}) \Rightarrow \#u \leq k + \kappa \cdot \#(s \uparrow L) - \#(s \downarrow L) \tag{5.3}$$

Das heißt, jede Trace s von $\mathtt{C}$ läßt sich zu einer Trace $s \frown w$ fortsetzen, so daß $(s \frown w, L) \in \mathcal{F}[\![\mathtt{C}]\!]$.

Sei $t \in \mathrm{seq}\, Comm(Chans(\mathcal{I}) \cup L)$. Dann:

$$
\begin{array}{ll}
& t \text{ ist } H \cup L\text{-unbeschränkt in } \mathtt{C} \\
\Leftrightarrow & \forall\, m \in \mathbb{N} \bullet \exists\, u : \operatorname{seq} \mathit{Comm}(H \cup L);\ \mathtt{C.State} \bullet \\
& t \in \mathit{traces}(\mathtt{C}) \wedge \#u > m \\
\Leftrightarrow & \forall\, m \in \mathbb{N} \bullet \exists\, u : \operatorname{seq} \mathit{Comm}(H \cup L);\ \mathtt{C.State} \bullet \\
& \mathtt{C}^{\star}_{t \frown u} \wedge \#u > m \\
& [\mathtt{C} \text{ enthält keine Divergenzen}] \\
\Rightarrow & \forall\, m \in \mathbb{N} \bullet \exists\, u : \operatorname{seq} \mathit{Comm}(H \cup L);\ \mathtt{C.State} \bullet \\
& \mathtt{C}^{\star}_{t \frown u} \wedge 0 \leq k + \kappa \cdot \#((t \frown u) \uparrow L) - \#((t \frown u) \downarrow L) \wedge \#u > m \\
& [\text{nach } 5.3.4] \\
\Leftrightarrow & \forall\, m \in \mathbb{N} \bullet \exists\, u : \operatorname{seq} \mathit{Comm}(H \cup L);\ \mathtt{C.State} \bullet \\
& \quad \mathtt{C}^{\star}_{t \frown u} \\
& \wedge\, 0 \leq k + \kappa \cdot \#(t \uparrow L) + \kappa \cdot \#(u \uparrow L) - \#(t \downarrow L) - \#(u \downarrow L) \\
& \wedge\, m < \#(u \uparrow L) + \#(u \downarrow L) \\
\Leftrightarrow & \forall\, m \in \mathbb{N} \bullet \exists\, u : \operatorname{seq} \mathit{Comm}(H \cup L);\ \mathtt{C.State} \bullet \\
& \mathtt{C}^{\star}_{t \frown u} \wedge m < k + \kappa \cdot \#(t \uparrow L) - \#(t \downarrow L) + (\kappa + 1) \cdot \#(u \uparrow L) \\
\Rightarrow & \forall\, m \in \mathbb{N} \bullet \exists\, u : \operatorname{seq} \mathit{Comm}(H \cup L);\ \mathtt{C.State} \bullet \\
& \mathtt{C}^{\star}_{t \frown u} \wedge m < \#(u \uparrow L) \\
& [\text{da } k + \kappa \cdot \#(t \uparrow L) - \#(t \downarrow L) \text{ und } (\kappa + 1) \text{ konstant für } t] \\
\Leftrightarrow & \forall\, m \in \mathbb{N} \bullet \exists\, u : \operatorname{seq} \mathit{Comm}(H \cup L);\ \mathtt{C.State} \bullet \\
& \mathtt{C}^{\star}_{t \frown u} \wedge m < \#(u \downarrow H) \\
\Rightarrow & \forall\, m \in \mathbb{N} \bullet \exists\, u : \operatorname{seq} \mathit{Comm}(H \cup L) \bullet \\
& ((t \frown u) \uparrow L, L) \in \mathcal{F}[\![\mathtt{C}]\!] \wedge m < \#(u \downarrow H) \\
& \Big[\text{da nach (5.3) } (t \frown u \frown w, L) \in \mathcal{F}[\![\mathtt{C}]\!] \text{ für ein } w \in \operatorname{seq} \mathit{Comm}(L)\Big] \\
\Leftrightarrow & \forall\, m \in \mathbb{N} \bullet \exists\, u : \operatorname{seq} \mathit{Comm}(H) \bullet \\
& ((t \frown u) \uparrow L, L) \in \mathcal{F}[\![\mathtt{C}]\!] \wedge m < \#u \\
\Leftrightarrow & \forall\, m \in \mathbb{N} \bullet \exists\, u : \operatorname{seq} \mathit{Comm}(H) \bullet \\
& (t \frown u) \uparrow L \in \mathit{traces}(\mathtt{C} \setminus L) \wedge m < \#u \\
\Leftrightarrow & t \uparrow L \text{ ist } H\text{-unbeschränkt in } \mathtt{C} \setminus L
\end{array}
$$

Da wegen (5.3) auch $\mathcal{D}[\![\mathtt{C} \setminus L]\!] = \varnothing$ gilt, ist alles bewiesen. □

Wurde eine Verfeinerung $\mathtt{A} \sqsubseteq \mathtt{C} \setminus L$ mit einer der Regeln **LIFS** oder **LIBS** bewiesen, so verhalten sich die Kanäle L der Spezifikation $\mathtt{C}$ eingeschränkt kommutativ zu allen anderen Kanälen $H \subseteq \mathit{Chans}(\mathcal{I}[\![\mathtt{C} \setminus L]\!])$.

Korollar 5.18 *Seien* $\mathtt{A}$ *und* $\mathtt{C} \setminus L$ *zwei Spezifikationen aus* $SPEC(\mathcal{I})$*. Gilt* $\mathtt{A} \sim_{LIFS} \mathtt{C} \setminus L$ *oder* $\mathtt{A} \sim_{LIBS} \mathtt{C} \setminus L$*, dann folgt für alle* $H \subseteq Chans(\mathcal{I})$

$$\mathtt{C} \setminus L \setminus H \equiv \mathtt{C} \setminus (L \cup H)$$

Beweis. Die Behauptung folgt aus Satz 5.17 und Lemma 5.15. □

Werden hintereinander mehrfach die Regeln *LIFS* und *LIBS* zur Einführung lokaler Kanäle H und L verwendet, so verhalten sich diese bezüglich Hiding echt kommutativ zueinander.

Korollar 5.19 *Seien* $\mathtt{A}$ *und* $\mathtt{C} \setminus L$ *Spezifikationen aus* $SPEC(\mathcal{I})$*. Gilt* $\mathtt{A} \sim_{LIFS} \mathtt{C} \setminus L$ *oder* $\mathtt{A} \backsim_{LIBS} \mathtt{C} \setminus L$*, und gilt* $\mathcal{D}[\![\mathtt{C} \setminus L \setminus H]\!] = \varnothing$*, dann ist der Hiding-Operator* (H, L)*-kommutativ für* $\mathtt{C}$*.*

Beweis. Nach Korollar 5.18 ist $\mathtt{C} \setminus L \setminus H \equiv \mathtt{C} \setminus (L \cup H)$, also insbesondere $\mathcal{D}[\![\mathtt{C} \setminus (L \cup H)]\!] = \varnothing$. Die Behauptung folgt nun mit Lemma 5.14. □

Kommutativität zweier Kanalmengen H und L einer Spezifikation kann auch direkt nachgewiesen werden, indem gezeigt wird, daß die Spezifikation bezüglich der Vereinigung dieser Kanalmengen schwach terminal ist. Dann folgt, daß die Divergenzen der Spezifikation nach dem verstecken der Kanalmenge $H \cup L$ in jedem Fall leer sind und daraus mit Lemma 5.14 die Kommutativität.

5.5 Elimination lokaler Kanäle

Hat eine Spezifikation $\mathtt{A} \setminus L$ lokale Kanäle L, so lassen sich diese in einem transformationellen Schritt eliminieren, indem eine Spezifikation $\mathtt{C}$ ohne lokale Kanäle gefunden wird, so daß $\mathtt{A} \setminus L \sqsubseteq \mathtt{C}$ gilt. In einigen Fällen wird es möglich sein, die Äquivalenz der Spezifikationen $\mathtt{A} \setminus L$ und $\mathtt{C}$ zu beweisen, wenn mit einer der Regeln **WLIFS** oder **WLIBS** auch die Verfeinerung $\mathtt{C} \sqsubseteq \mathtt{A} \setminus L$ bewiesen werden kann.

In anderen Arbeiten [AL91, But92] wurde bereits gezeigt, daß Verfeinerungsregeln, die lokale Kanäle einbeziehen nicht vollständig sein können. Das heißt auch durch Kombination mehrer Regeln kann eine Verfeinerung $\mathtt{S} \sqsubseteq \mathtt{T}$ nicht unbedingt bewiesen werden.

Definition 5.20 (Vorwärtssimulation "LEFS")
Seien $\mathtt{A}$ *und* $\mathtt{C}$ *mit* $\mathcal{L}[\![\mathtt{C}]\!] = \varnothing$ *und* $\mathcal{L}[\![\mathtt{A}]\!] = L$ *Spezifikationen aus* $SPEC(\mathcal{I})$ *ohne* CSP*-Teil. Ist* $\mathtt{A}$ *schwach* L*-terminal, dann wird ein Repräsentationsschema* Rep *als* $LEFS$-Schema *von* $\mathtt{A}$ *nach* $\mathtt{C}$ *bezeichnet, wenn:*

1. **(Initialisierung)**
   $\forall\, \mathtt{C.State}' \bullet \exists\, \mathtt{A.State}' \bullet$
   $\mathtt{C.Init\_State} \Rightarrow \mathtt{A.Init\_State} \wedge Rep'$

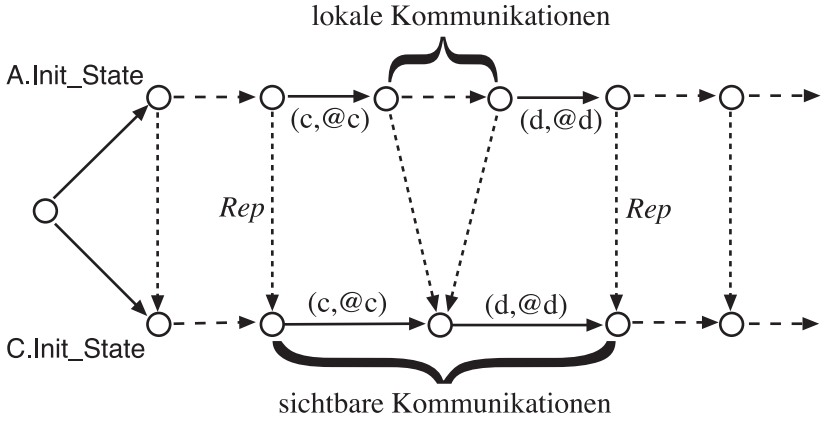


Abbildung 5.3: Vorwärtssimulation

2. **(Definiertheit)**
   Für alle Kanäle $c \in Chans(\mathcal{I})$ *gilt:*
   $\forall$ C.State; A.State; $@c : type_{\mathcal{I}}(c) \bullet$
   pre A.com_ $c \wedge Rep \Rightarrow$ pre C.com_ c

3. **(Fortschritt)**
   Für alle Kanäle $c \in Chans(\mathcal{I})$ *gilt:*
   $\forall \Delta$C.State; A.State; $@c : type_{\mathcal{I}}(c) \bullet$
   $(Rep \wedge$ C.com_ $c \Rightarrow (\exists$ A.State$' \bullet$ A.com_ $c \wedge Rep')$
   $\vee (\exists \ell \in Chans(\mathcal{L}) \bullet \exists @\ell : type_{\mathcal{L}}(\ell) \bullet$
   $Rep \wedge \Xi$C.State $\wedge$ pre C.com_ $c \Rightarrow (\exists$ A.State$' \bullet$ A.com_ $\ell \wedge Rep'))$

Existiert ein LEFS-Schema von A *nach* C*, so wird dies durch* A $\sim_{LEFS}$ C *notiert.* □

Die Bedingungen 5.20.1 und 5.20.2 sind aus früheren Regeln bekannt.

Aussage 5.20.3 bedeutet grob gesprochen, daß die Operationen der sichtbaren Kanäle der abstrakten Spezifikation das Verhalten der konkreten Operationen der lokalen Kanäle *und* der konkreten Operationen der entsprechenden sichtbaren Kanäle simulieren.

Befinden sich die abstrakte und die konkrete Spezifikation in einander entsprechenden Zuständen und kann die konkrete Operation C.com_ c ausgeführt werden, dann

- kann die entsprechende abstrakte Operation A.com_ c ausgeführt werden, und der Zustand der abstrakten Spezifikation hinterher entspricht dem der konkreten Spezifikation nach Ausführung von C.com_ c,

- oder eine Operation $\mathtt{A.com\_}\ell$ eines lokalen Kanals ℓ von A kann ausgeführt werden, und der Zustand der konkreten Spezifikation bleibt unberührt.

In Abbildung 5.3 ist dieser Sachverhalt dargestellt.

Bei der Rückwärtssimulation spielen die lokalen Kanäle der abstrakten Spezifikation keine Rolle. Die Simulation bezieht nur die sichtbaren Kanäle des Interfaces ein.

Definition 5.21 (Rückwärtssimulation "LEBS")
Seien A *und* C *mit* $\mathcal{L}[\![\mathtt{C}]\!] = \varnothing$ *und* $\mathcal{L}[\![\mathtt{A}]\!] = L$ *Spezifikationen aus* $SPEC(\mathcal{I})$ *ohne* CSP*-Teil. Ist* A *schwach L-terminal, dann wird ein Repräsentationsschema Rep als LEBS-*Schema *von* A *nach* C *bezeichnet, wenn:*

1. **(Initialisierung)**
   $\forall\, \mathtt{C.State}'; \mathtt{A.State}' \bullet$
   $\mathtt{C.Init\_State} \wedge Rep' \Rightarrow \mathtt{A.Init\_State}$

2. **(Definiertheit)**
   Für alle Kanäle $c \in Chans(\mathcal{I})$ *gilt:*
   $\forall\, \mathtt{C.State};\ @c : type_{\mathcal{I}}(c) \bullet \exists\, \mathtt{A.State} \bullet$
   $Rep \wedge (\mathrm{pre}\, \mathtt{A.com\_}\, c \Rightarrow \mathrm{pre}\, \mathtt{C.com\_}\, c)$

3. **(Fortschritt)**
   Für alle Kanäle $c \in Chans(\mathcal{I})$ *gilt:*
   $\forall\, \Delta\mathtt{C.State};\ \mathtt{A.State}';\ @c : type_{\mathcal{I}}(c) \bullet$
   $\mathtt{C.com\_}\, c \wedge Rep' \Rightarrow (\exists\, \mathtt{A.State} \bullet Rep \wedge \mathtt{A.com\_}\, c)$

Existiert ein LEBS-Schema von A *nach* C*, so wird dies durch* $\mathtt{A} \sim_{LEBS} \mathtt{C}$ *notiert.* □

5.6 Korrektheit von LEFS und LEBS

Seien $\mathtt{A} \setminus L$ und C Spezifikationen und *Rep* ein *LEFS*-Schema von $\mathtt{A} \setminus L$ nach C. Damit irgendwann kein lokaler Kanal der abstrakten Spezifikation $\mathtt{A} \setminus L$ mehr bereit ist, an einer Kommunikation teilzunehmen, muß die Spezifikation schwach terminal sein. Zusätzlich darf eine Sequenz lokaler Kommunikationen in einer Trace von A den Zustand der konkreten Spezifikation C unter der Repräsentationsbeziehung *Rep* nicht berühren. Das Lemma 5.22 ermöglicht es Traces *tr* von A so fortzusetzen, daß keine weitere lokale Kommunikation mehr möglich ist, und deshalb eine sichtbare Kommunikation stattfinden muß.

Lemma 5.22 *Seien* $\mathtt{A} \setminus L$ *und* $\mathtt{C}$ *zwei Spezifikationen aus* $SPEC(\mathcal{I})$ *mit* $L = Chans(\mathcal{L}[\![\mathtt{A} \setminus L]\!])$ *und* $\mathcal{L}[\![\mathtt{C}]\!] = \varnothing$*, und sei* $\mathtt{A} \setminus L$ *schwach L-terminal. Gilt 5.20.3, dann auch*

$$\begin{aligned}
&\forall\, tr : \mathrm{seq}\, Comm(\mathcal{I}[\![\mathtt{A}]\!]);\ \mathtt{C.State};\ (c,v) : Comm(\mathcal{I}[\![\mathtt{A}]\!]) \bullet \\
&\qquad (\exists\, \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr} \wedge Rep \wedge \mathrm{pre}\, \mathtt{C.com_}\,(c,v)) \\
&\Rightarrow\ (\exists\, u : \mathrm{seq}\, Comm(L);\ \mathtt{A.State};\ \mathtt{C.State}' \bullet \\
&\qquad\quad \mathtt{A}^{\star}_{tr \frown u} \wedge Rep \wedge \Xi\mathtt{C.State} \wedge \mathrm{pre}\, \mathtt{C.com_}\,(c,v) \\
&\qquad \wedge\ (\forall\, \ell : L;\ @\ell : type_L(\ell) \bullet \neg\, (\exists\, \mathtt{A.State}' \bullet \mathtt{A.com_}\,\ell \wedge Rep')))
\end{aligned}$$

Beweis. Die Spezifikation $\mathtt{A}$ ist L-beschränkt nach Bemerkung 5.4, dh. für alle $tr : \mathrm{seq}\, Comm(\mathcal{I}[\![\mathtt{A}]\!])$ und alle $u : \mathrm{seq}\, Comm(L)$ gilt:

$$(\exists\, \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr \frown u}) \Rightarrow \#u \leq \kappa_{tr}$$

für eine natürliche Zahl κ_{tr}, die nur tr abhängt. Daraus folgt

$$(\exists\, \mathtt{A.State};\ \mathtt{C.State} \bullet \mathtt{A}^{\star}_{tr \frown u} \wedge Rep \wedge \mathrm{pre}\, \mathtt{C.com\_}\,(c,v)) \Rightarrow \#u \leq \kappa_{tr},$$

wobei $(c,v) \in Comm(\mathcal{I}[\![\mathtt{C}]\!])$. Also gibt es zu jedem $tr : \mathrm{seq}\, Comm(\mathcal{I}[\![\mathtt{A}]\!])$ und jedem $u : \mathrm{seq}\, Comm(L)$ ein $w : \mathrm{seq}\, Comm(L)$, so daß gilt

$$\exists\, \mathtt{A.State};\ \mathtt{C.State} \bullet \mathtt{A}^{\star}_{tr \frown u \frown w} \wedge Rep \wedge \mathrm{pre}\, \mathtt{C.com\_}\,(c,v),$$

aber kein $x : Comm(L)$ existiert mit:

$$\exists\, \mathtt{A.State};\ \mathtt{C.State} \bullet \mathtt{A}^{\star}_{tr \frown u \frown w \frown \langle x \rangle} \wedge Rep \wedge \mathrm{pre}\, \mathtt{C.com\_}\,(c,v).$$

Das heißt

$$\begin{aligned}
&\exists\, \mathtt{A.State};\ \mathtt{C.State};\ \mathtt{C.State}' \bullet \\
&\quad \mathtt{A}^{\star}_{tr \frown u \frown w} \wedge Rep \wedge \Xi\mathtt{C.State} \wedge \mathrm{pre}\, \mathtt{C.com_}\,(c,v) \\
&\wedge\ (\forall\, \ell : L;\ @\ell : type_L(\ell) \bullet \neg\, (\exists\, \mathtt{A.State}' \bullet \mathtt{A.com_}\,\ell \wedge Rep'))
\end{aligned}$$

Die Trace $tr \frown u \frown w$ heißt dann *L-maximal* in $\mathtt{A}$. Es bleibt noch zu zeigen, daß ausgehend von einer Trace tr von $\mathtt{A}$ eine L-maximale Trace $tr \frown u$ erreicht werden kann, ohne den Wert $\Theta\mathtt{C.State}$ der Repräsentationsrelation zu verändern. Der Beweis erfolgt durch Induktion über die Länge der Reststücke u, die an eine Trace tr von $\mathtt{A}$ angehängt werden können, so daß $tr \frown u$ eine Trace von $\mathtt{A}$ ist.

Es gilt:

$$\begin{aligned}
&\forall\, tr : \mathrm{seq}\, Comm(\mathcal{I}[\![\mathtt{A}]\!]);\ \mathtt{C.State};\ (c,v) : Comm(\mathcal{I}[\![\mathtt{C}]\!]) \bullet \\
&\qquad (\exists\, \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr} \wedge Rep \wedge \mathrm{pre}\, \mathtt{C.com_}\,(c,v)) \\
&\Rightarrow\ (\exists\, u : \mathrm{seq}\, Comm(L);\ \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr \frown u} \wedge Rep \wedge \mathrm{pre}\, \mathtt{C.com_}\,(c,v) \\
&\qquad \wedge\ tr \frown u \text{ ist } L\text{-maximal})
\end{aligned}$$

Seien $tr : \operatorname{seq} Comm(\mathcal{I}[\![\mathtt{A}]\!])$, $p : \mathtt{C.State}$ und $(c, v) : Comm(\mathcal{I}[\![\mathtt{C}]\!])$, so daß gilt:

$$(\exists\, \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr} \wedge Rep \wedge \operatorname{pre} \mathtt{C.com}\_\,(c, v))[p/\mathtt{C.State}].$$

Ist $u = \langle\,\rangle$, dann ist tr bereits L-maximal, und es ist nichts zu zeigen.

Sei $u = \langle x \rangle \frown w$ mit $w : \operatorname{seq} Comm(L)$ und $x : Comm(L)$, so daß $tr \frown u$ eine L-maximale Trace von $\mathtt{A}$ ist. Die Trace tr ist also nicht L-maximal und muß um mindestens eine Kommunikation x verlängert werden. Mit Voraussetzung 5.20.3 folgt nun für eine passende Kommunikation x:

$$\begin{aligned}&(\exists\, \Delta\mathtt{A.State};\ \mathtt{C.State}' \bullet \\ &\mathtt{A}^{\star}_{tr} \wedge \mathtt{A.com}\_\, x \wedge Rep \wedge \Xi\mathtt{C.State} \wedge (\operatorname{pre} \mathtt{C.com}\_\,(c, v))')[p/\mathtt{C.State}].\end{aligned}$$

Und daraus:

$$(\exists\, \mathtt{A.State} \bullet \mathtt{A}^{\star}_{tr \frown x} \wedge Rep \wedge \operatorname{pre} \mathtt{C.com}\_\,(c, v))[p/\mathtt{C.State}].$$

Die Anwendung der Induktionshypothese auf w liefert:

$$\begin{aligned}&(\exists\, \mathtt{A.State} \bullet \quad \mathtt{A}^{\star}_{tr \frown x \frown w} \wedge Rep \wedge \operatorname{pre} \mathtt{C.com}\_\,(c, v))[p/\mathtt{C.State}] \\ &\text{und } tr \frown \langle x \rangle \frown w \text{ ist } L\text{-maximal}\end{aligned}$$

□

Das folgende Lemma ist in ähnlicher Form bereits aus den letzten Abschnitten bekannt. Durch den Nachweis der Failures-Inklusion ist fast alles bewiesen.

Lemma 5.23 *Seien* $\mathtt{A} \setminus L$, $\mathtt{C} \in SPEC(\mathcal{I})$ *und* $\mathcal{L}[\![\mathtt{A} \setminus L]\!] = L$, $\mathcal{L}[\![\mathtt{C}]\!] = \varnothing$. *Und sei Rep ein Repräsentationsschema.*

1. *Gelten 5.20.1 und 5.20.3, so auch für alle* $tr : \operatorname{seq} Comm(Chans(\mathcal{I}))$:

$$\begin{aligned}&\forall\, \mathtt{C.State} \bullet \exists\, sr : \operatorname{seq} Comm(\mathcal{I}[\![\mathtt{A}]\!]);\ \mathtt{A.State} \bullet \\ &sr \uparrow L = tr \wedge (\mathtt{C}^{\star}_{tr} \Rightarrow \mathtt{A}^{\star}_{sr} \wedge Rep).\end{aligned}$$

2. *Gilt außerdem 5.20.2, dann folgt*

$$\mathcal{F}[\![\mathtt{C}]\!] \subseteq \mathcal{F}[\![\mathtt{A}]\!]$$

Beweis. Induktion über die Länge von Traces.

Für $tr = \langle\,\rangle$ folgt die Behauptung wegen Voraussetzung 5.20.1.

Sei $tr = ur \frown \langle (c, v) \rangle : \operatorname{seq} Comm(Chans(\mathcal{I}))$.

$$
\begin{array}{ll}
& \forall\, \mathtt{C.State}' \bullet \mathtt{C}_{tr} \Rightarrow \\
& \qquad \exists\, \mathtt{C.State} \bullet \mathtt{C}^{\star}_{ur} \wedge \mathtt{C.com}_{\_}\,(c, v) \\
\Rightarrow & \forall\, \mathtt{C.State}' \bullet \mathtt{C}_{tr} \Rightarrow \\
& \qquad \exists\, \mathtt{C.State};\ sr : \mathrm{seq}\, \mathit{Comm}(\mathcal{I}[\![\mathtt{A}]\!]);\ \mathtt{A.State} \bullet \\
& \qquad sr \uparrow L = ur \wedge \mathtt{A}^{\star}_{sr} \wedge \mathit{Rep} \wedge \mathtt{C.com}_{\_}\,(c, v) \\
& [\text{nach Induktionshypothese}] \\
\Rightarrow & \forall\, \mathtt{C.State}'' \bullet \mathtt{C}_{tr}[\mathtt{State}''/\mathtt{State}'] \Rightarrow \\
& \qquad \exists\, \Delta\mathtt{C.State};\ sr : \mathrm{seq}\, \mathit{Comm}(\mathcal{I}[\![\mathtt{A}]\!]);\ \mathtt{A.State} \bullet \\
& \qquad\quad sr \uparrow L = ur \wedge \mathtt{A}^{\star}_{sr} \wedge \mathit{Rep} \wedge \mathtt{C.com}_{\_}\,(c, v)[\mathtt{State}''/\mathtt{State}'] \\
& \qquad \wedge\, \Xi\mathtt{State} \\
& \qquad \wedge\, (\forall\, \ell : L;\ @\ell : \mathit{type}_{\mathcal{L}}(\ell) \bullet \neg\, (\exists\, \mathtt{A.State}' \bullet \mathtt{A.com}_{\_}\,\ell \wedge \mathit{Rep}')) \\
& [\text{nach Lemma 5.22}] \\
\Rightarrow & \forall\, \mathtt{C.State}' \bullet \mathtt{C}_{tr} \Rightarrow \\
& \qquad \exists\, sr : \mathrm{seq}\, \mathit{Comm}(\mathcal{I}[\![\mathtt{A}]\!]);\ \Delta\mathtt{A.State} \bullet \\
& \qquad sr \uparrow L = ur \wedge \mathtt{A}^{\star}_{sr} \wedge \mathtt{A.com}_{\_}\,(c, v) \wedge \mathit{Rep}' \\
& [\text{nach Voraussetzung 5.20.3}] \\
\Rightarrow & \forall\, \mathtt{C.State} \bullet \mathtt{C}^{\star}_{tr} \Rightarrow \\
& \qquad \exists\, sr \frown \langle (c, v) \rangle : \mathrm{seq}\, \mathit{Comm}(\mathcal{I}[\![\mathtt{A}]\!]);\ \mathtt{A.State} \bullet \\
& \qquad sr \frown \langle (c, v) \rangle \uparrow L = tr \wedge \mathtt{A}^{\star}_{sr \frown \langle (c,v) \rangle} \wedge \mathit{Rep}
\end{array}
$$

Zum Beweis der zweiten Behauptung.

$$
\begin{array}{ll}
& (tr, R) \in \mathcal{F}[\![\mathtt{C}]\!] \\
\Leftrightarrow & \exists\, \mathtt{C.State} \bullet \mathtt{C}^{\star}_{tr} \wedge \mathit{Ref}_{\mathtt{C}}\, R \\
\Rightarrow & \exists\, \mathtt{C.State};\ \mathtt{A.State};\ sr : \mathrm{seq}\, \mathit{Comm}(\mathcal{I}[\![\mathtt{A}]\!]) \bullet \\
& \qquad sr \uparrow L = tr \wedge \mathtt{A}^{\star}_{sr} \wedge \mathit{Rep} \wedge \mathit{Ref}_{\mathtt{C}}\, R \\
& [\text{nach 1}] \\
\Rightarrow & \exists\, \mathtt{A.State};\ sr : \mathrm{seq}\, \mathit{Comm}(\mathcal{I}[\![\mathtt{A}]\!]) \bullet \\
& \qquad sr \uparrow L = tr \wedge \mathtt{A}^{\star}_{sr} \wedge \mathit{Ref}_{\mathtt{A}}\, R \\
& [\text{nach Voraussetzung 5.20.2}] \\
\Rightarrow & \exists\, \mathtt{A.State};\ sr : \mathrm{seq}\, \mathit{Comm}(\mathcal{I}[\![\mathtt{A}]\!]) \bullet \\
& \qquad sr \uparrow L = tr \wedge \mathtt{A}^{\star}_{sr} \wedge \mathit{Ref}_{\mathtt{A}}\, (R \cup \mathit{Comm}(L)) \\
& [\text{nach Bemerkung 5.4}] \\
\Leftrightarrow & (tr, R) \in \mathcal{F}[\![\mathtt{A} \setminus \mathtt{L}]\!]
\end{array}
$$

□

Die Korrektheit der Regel **LEFS** ist nun eine einfache Folgerung. Wie für die Regeln **WLIFS** und **WLIBS** wird auch hier verlangt, daß die Spezifikationen keinen CSP-Prozeß enthalten.

Satz 5.24 (Korrektheit von LEFS) *Für zwei Spezifikationen* A *und* C *aus* $SPEC(\mathcal{I})$ *mit leerem* CSP*-Teil gilt:*

$$\mathtt{A} \sim_{LEFS} \mathtt{C} \Rightarrow \mathtt{A} \sqsubseteq \mathtt{C}.$$

Beweis. Nach Definition gilt $\mathcal{D}[\![\mathtt{C}]\!] = \varnothing$. und wegen 5.23.2 auch $\mathcal{F}[\![\mathtt{C}]\!] \subseteq \mathcal{F}[\![\mathtt{A}]\!]$. □

Die Korrektheit der Regel **LEBS** ließe sich auch beweisen, wenn die Voraussetzung weggelassen würde, daß die abstrakte Spezifikation schwach L-terminal ist. Die zum Beweis notwendigen Fakten sind bereits alle bekannt.

Satz 5.25 (Korrektheit von LEBS) *Für zwei Spezifikationen* A *und* C *aus* $SPEC(\mathcal{I})$ *mit leerem* CSP*-Teil gilt:*

$$\mathtt{A} \sim_{LEBS} \mathtt{C} \Rightarrow \mathtt{A} \sqsubseteq \mathtt{C}.$$

Beweis. Nach Definition gilt $\mathcal{D}[\![\mathtt{C}]\!] = \varnothing$. Der Beweis für $\mathcal{F}[\![\mathtt{C}]\!] \subseteq \mathcal{F}[\![\mathtt{A}]\!]$ ist beinahe identisch mit dem zur Regel **BS**. Alles andere folgt wie im Beweis von 5.23. □

5.7 Anmerkungen

In [But92, But93] wird eine Verfeinerungsregel für Action Systeme angegeben, welche die Grundlage für die Formulierung der beiden Regeln **WLIFS** und **WLIBS** war. In [But92] befindet sich eine Diskussion über Vollständigkeitsresultate von Beweisregeln im Zusammenhang mit lokalen Kanälen bzw. unsichtbaren Zustandsübergängen, die durch Hiding entstehen. Es stellt sich heraus, daß die betrachtete Klasse von Programmen stark eingeschränkt werden muß, damit ein Kalkül, der lokale Kanäle einbezieht, vollständig sein kann.

In [Hoa81] schlägt Hoare vor, zur Vermeidung von Divergenz durch Anwendung des Hiding-Operators auf eine Spezifikation S und eine Menge L von Kanälen dieser Spezifikation, eine Funktion f zu benutzen, die die Anzahl der unsichtbaren Kommunikationen in Traces von S begrenzt. Der Wert von f darf nur von den Kommunikationswerten der sichtbaren Kommunikationen oder der Anzahl der sichtbaren Kommunikationen abhängen:

$$tr \in traces(\mathtt{S}) \Rightarrow \#(tr \downarrow L) \leq f(tr \uparrow L).$$

Die Definition schwach L-terminaler Spezifikationen führt zu Abschätzungen der Anzahl unsichtbarer Kommunikationen durch Betrachtung von Termen der Form

$$tr \in \mathit{traces}(\mathtt{S}) \Rightarrow \#(tr \downarrow L) \leq g(tr \uparrow L),$$

wobei $g(sr) = k + \sum_{sr}$. Die Funktion g entspricht einer speziellen Wahl für die von Hoare vorgeschlagene Funktion f.

Kapitel 6

Fallstudie B

Diese Fallstudie ist eine Variante zweier Fallstudien aus [WD96] und [Fis96]. Sie beschreibt einen Ausschnitt des Telekommunikationsprotokolls "Signalling System No. 7". Anhand von drei möglichen Spezifikationen dieses Protokolls wird die Anwendung zweier Regeln des vorigen Kapitels vorgeführt.

Das Protokoll verfügt über eine Menge M von Nachrichten, die es behandeln kann. Diese Menge wird durch einen Basistypen modelliert:

$$[M]$$

Nachrichten können verschickt und empfangen werden. Dabei dürfen keine Nachrichten verloren gehen, und die Reihenfolge, in der sie verschickt wurden, darf nicht verändert werden.

In Abschnitt 6.1 wird eine erste Spezifikation für das Protokoll entworfen. Diese läßt sich vereinfachen, so daß in weiteren Verfeinerungschritten von der einfacheren Spezifikation aus Abschnitt 6.2 ausgegangen werden kann. Schließlich wird in Abschnitt 6.3 eine Spezifikation gezeigt, die näher an realen Systemen liegt, welche das Protokoll implementieren. Insgesamt wird bewiesen, daß die Spezifikation aus dem letzten Abschnitt dieses Kapitels eine Verfeinerung der Spezifikation aus dem ersten Abschnitt 6.1 ist.

6.1 Eine erste Spezifikation

Alle Nachrichten $m \in M$ werden über einen Kanal *transmit* verschickt und über den Kanal *receive* empfangen. Sie werden in verschiedenen Puffern zwischengespeichert, bis sie ihr Empfangsziel erreichen.

Die Spezifikation `Attempt` auf der nächsten Seite beschreibt das Kommunikationsprotokoll. Alle verschickten und nicht empfangenen Nachrichten werden in einer der beiden Sequenzen *in* oder *out* gespeichert. Eine Kommunikation über den Kanal *in* hängt eine Nachricht an den Anfang der Sequenz *in*, während eine Kommunikation über den Kanal *out* das letzte Element der Sequenz *out* entfernt. Außer den sichtbaren Kanälen *transmit* und *receive*

spec Attempt

channel *transmit*, *receive* : M
local_channel *pass* : signal

State

$in : \operatorname{seq} M$
$out : \operatorname{seq} M$

Init_State

State′

$in' = \langle\,\rangle \land out' = \langle\,\rangle$

com_ *transmit*

ΔState
$@transmit : M$

$in' = \langle @transmit \rangle \frown in$
$out' = out$

com_ *receive*

ΔState
$@receive : M$

$out' \frown \langle @receive \rangle = out$
$in' = in$

com_ *pass*

ΔState

$out' = \langle last\ in \rangle \frown out$
$in' = front\ in$

end_spec Attempt

verfügt Attempt über einen lokalen Kanal *pass*, der Verbindung zwischen dem Eingabepuffer *in* und dem Ausgabepuffer *out* herstellt, indem er das letzte Element des Puffers *in* an den Anfang des Puffers *out* verschiebt.

Die Vorbedingungen der Operationsschemata der Spezifikation Attempt lauten:

pre Attempt.com_ *transmit* ≡
$[\,\text{Attempt.State};\ @transmit : M \mid \text{true}\,]$

pre Attempt.com_ *receive* ≡
$[\,\text{Attempt.State};\ @receive : M \mid out \neq \langle\,\rangle \land last\ out = @receive\,]$

pre Attempt.com_ *pass* ≡
$[\,\text{Attempt.State} \mid in \neq \varnothing\,]$

spec Simple

channel *transmit*, *receive* : M

State

$s : \operatorname{seq} M$

Init_State

$\texttt{State}'$

$s' = \langle\,\rangle$

com_ *transmit*

$\Delta\texttt{State}$

$@transmit : M$

$s' = \langle @transmit \rangle \frown s$

com_ *receive*

$\Delta\texttt{State}$

$@receive : M$

$s' \frown \langle @receive \rangle = s$

end_spec Simple

6.2 Eine einfachere Spezifikation

Die Spezifikation `Attempt` modelliert direkt den Gedanken, daß Nachrichten vom Puffer des Senders über ein Medium in Puffer des Empfängers übertragen wird. Für folgende Verfeinerungsschritte ist es sinnvoll, von der einfacheren Spezifikation `Simple` auszugehen.

Die Spezifikation `Simple` benutzt nur einen einzigen Puffer s, um verschickte Nachrichten zu speichern. Bei einer Kommunikation über den Kanal *transmit* werden Nachrichten an den Anfang des Puffers gehängt, und bei einer Kommunikation über den Kanal *receive* werden sie vom Ende des Puffers entfernt.

Die Vorbedingungen der Operationen von `Simple` lauten:

$$\operatorname{pre} \texttt{Simple.com\_}\mathit{transmit} \equiv [\texttt{Simple.State};\ @transmit : M \mid \text{true}]$$

$$\operatorname{pre} \texttt{Simple.com\_}\mathit{receive} \equiv [\texttt{Simple.State};\ @receive : M \mid s \neq \langle\,\rangle \wedge \mathit{last}\ s = @receive]$$

6.2.1 Die Einfache verfeinert die Erste

Es gilt `Attempt` $\sqsubseteq$ `Simple`. Diese Behauptung läßt sich mit der Regel **LEFS** beweisen:

$$\texttt{Attempt} \sim_{LEFS} \texttt{Simple}$$

Um eine der Regeln zur Entfernung lokaler Kanäle anwenden zu können, muß zuerst nachgewiesen werden, daß die Spezifikation, die aus `Attempt` entsteht, wenn der Kanal *pass* sichtbar gemacht wird, mindestens schwach $\{pass\}$-terminal ist.

Bemerkung 6.1 *Sei* `Test` $\setminus \{pass\} =$ `Attempt` *die Spezifikation, die syntaktisch mit* `Attempt` *übereinstimmt, außer daß pass ein sichtbarer Kanal von* `Test` *ist. Dann ist* `Test` *eine* $\{pass\}$*-terminale Spezifikation.*

Beweis. Sei *term* ein Schema, $k = 0$ eine Konstante, und $k_{transmit}$ und $k_{receive}$ totale Funktionen wie in Definition 5.2.

term

$$\texttt{Test.State}$$
$$t : \mathbb{Z}$$
$$t = \#in$$

$$k_{transmit} : M \to \mathbb{N} \qquad \forall\, m : M \bullet k_{transmit}(m) = 1$$

$$k_{receive} : M \to \mathbb{N} \qquad \forall\, m : M \bullet k_{receive}(m) = 0$$

Für die Spezifikation `Test` sind nun die vier in Definition 5.2 angegebenen Bedingungen zu beweisen. Da t für alle Werte von *in* definiert ist, ist 5.2.1 erfüllt.

Zu 5.2.2:

$$\begin{aligned}
&\forall\, term' \bullet \\
&\qquad \texttt{Test.Init_State} \\
&\Leftrightarrow \quad in' = \langle\,\rangle \wedge out' = \langle\,\rangle \\
&\Rightarrow \quad \#in' = 0 \\
&\Leftrightarrow \quad 0 \le t' \le k
\end{aligned}$$

Zu 5.2.3:

$$\begin{aligned}
&\forall\, \Delta term;\ @transmit : M \bullet \\
&\qquad \texttt{Test.com_}\, transmit \wedge t \ge 0 \\
&\Leftrightarrow \quad in' = \langle @transmit \rangle \frown in \wedge out' = out \wedge t \ge 0 \\
&\Rightarrow \quad \#in' = (\#in) + 1 \wedge \#in' \ge 1 \\
&\Leftrightarrow \quad 0 \le t' \le t + k_{transmit}(@transmit)
\end{aligned}$$

$$
\begin{array}{ll}
\forall\, \Delta term;\ @receive : M \bullet & \\
\quad \texttt{Test.com\_}\ receive \land t \geq 0 & \\
\Leftrightarrow & out' \frown \langle @receive \rangle = out \land in' = in \land t \geq 0 \\
\Rightarrow & \#in' = \#in \land \#in' \geq 0 \\
\Leftrightarrow & 0 \leq t' \leq t + k_{receive}(@receive)
\end{array}
$$

Zu 5.2.4:

$$
\begin{array}{ll}
\forall\, \Delta term \bullet & \\
\quad \texttt{Test.com\_}\ pass \land t \geq 0 & \\
\Leftrightarrow & out' = \langle last\ in \rangle \frown out \land in' = front\ in \land t \geq 0 \\
\Rightarrow & in \neq \langle\,\rangle \land in' = front\ in \land \#in \geq 1 \\
\Rightarrow & \#in' = (\#in) - 1 \land \#in' \geq 0 \\
\Rightarrow & 0 \leq t' < t
\end{array}
$$

□

Beweis der Verfeinerung

Das Repräsentationsschema *Rep* ist ein *LEFS*-Schema von `Attempt` nach `Simple`.

Rep

`Attempt.State`
`Simple.State`

$s = in \frown out$

Zum Beweis müssen die Bedingungen "Initialisierung", "Definiertheit" und "Fortschritt" der Regel **LEFS** erfüllt sein.

Initialisierung

Alle Sequenzen der beiden Spezifikationen werden anfangs durch die leere Sequenz initialisiert. Der Beweis dieser Bedingung ist entsprechend einfach.

$$
\begin{array}{ll}
\forall\, s' : \mathrm{seq}\, M \bullet & \\
\quad \texttt{Simple.Init\_State} & \\
\Leftrightarrow & s' = \langle\,\rangle \\
\Leftrightarrow & s' = \langle\,\rangle \frown \langle\,\rangle \\
\Leftrightarrow & (\exists\, in', out' : \mathrm{seq}\, M \bullet in' = \langle\,\rangle \land out' = \langle\,\rangle \land s' = in' \frown out') \\
\Leftrightarrow & (\exists\, in', out' : \mathrm{seq}\, M \bullet \texttt{Attempt.Init\_State} \land Rep')
\end{array}
$$

Definiertheit

Die Definiertheitsbedingung ist für beide Kanäle *transmit* und *receive* erfüllt, denn:

$$
\begin{aligned}
&\forall\, s : seqM;\ in, out : seqM;\ @transmit : M \bullet \\
&\qquad \mathrm{pre}\, \texttt{Attempt.com\_}\ transmit \wedge Rep \\
&\Leftrightarrow \quad \mathrm{true} \wedge s = in \frown out \\
&\Rightarrow \quad \mathrm{true} \\
&\Leftrightarrow \quad \mathrm{pre}\, \texttt{Simple.com\_}\ transmit
\end{aligned}
$$

$$
\begin{aligned}
&\forall\, s : seqM;\ in, out : seqM;\ @receive : M \bullet \\
&\qquad \mathrm{pre}\, \texttt{Attempt.com\_}\ receive \wedge Rep \\
&\Leftrightarrow \quad out \neq \langle\,\rangle \wedge last\ out = @receive \wedge s = in \frown out \\
&\Rightarrow \quad s \neq \langle\,\rangle \wedge last\ s = @receive \\
&\Leftrightarrow \quad \mathrm{pre}\, \texttt{Simple.com\_}\ receive
\end{aligned}
$$

Fortschritt

Die Fortschrittsbedingung für den Kanal *transmit* kann ohne Berücksichtigung des lokalen Kanals *pass* bewiesen werden.

$$
\begin{aligned}
&\forall\, s, s' : \mathrm{seq}\, M;\ in, out : \mathrm{seq}\, M;\ @transmit : M \bullet \\
&\qquad Rep \wedge \texttt{Simple.com\_}\ transmit \\
&\Leftrightarrow \quad s = in \frown out \wedge s' = \langle @transmit \rangle \frown s \\
&\Rightarrow \quad s' = \langle @transmit \rangle \frown in \frown out \\
&\Leftrightarrow \quad (\exists\, in', out' : \mathrm{seq}\, M \bullet \quad in' = \langle @transmit \rangle \frown in \wedge out' = out \\
&\qquad\qquad\qquad\qquad\qquad\qquad \wedge\, s' = in' \frown out') \\
&\Leftrightarrow \quad (\exists\, in', out' : \mathrm{seq}\, M \bullet \texttt{Attempt.com\_}\ transmit \wedge Rep')
\end{aligned}
$$

Da aus $s \neq \langle\,\rangle \wedge s = in \frown out$ folgt $in \neq \varnothing \vee out \neq \langle\,\rangle$, folgt auch die Fortschrittsbedingung für den Kanal *receive*:

$$
\begin{aligned}
&\forall\, s, s' : \mathrm{seq}\, M;\ in, out : \mathrm{seq}\, M;\ @receive : M \bullet \\
&\qquad Rep \wedge \texttt{Simple.com\_}\ receive \\
&\Leftrightarrow \quad s = in \frown out \wedge s' \frown \langle @receive \rangle = s
\end{aligned}
$$

$$
\begin{array}{ll}
\Rightarrow & (\exists\, in', out' : \operatorname{seq} M \bullet \quad in' = in \land out' \frown \langle @receive \rangle = out \\
& \qquad\qquad\qquad\qquad \land\ s' = in' \frown out') \\
& [\text{falls } out \neq \langle\,\rangle] \\
\Leftrightarrow & (\exists\, in', out' : \operatorname{seq} M \bullet \texttt{Attempt.com\_}\ receive \land Rep')
\end{array}
$$

$$
\begin{array}{ll}
\forall\, s, s' : \operatorname{seq} M;\ in, out : \operatorname{seq} M;\ @receive : M \bullet & \\
\qquad Rep \land \Xi \texttt{Simple.State} \land \operatorname{pre} \texttt{Simple.com\_}\ receive & \\
\Leftrightarrow & s = in \frown out \land s' = s \land s \neq \langle\,\rangle \land last\ s = @receive \\
\Rightarrow & s' = in \frown out \land s' \neq \langle\,\rangle \\
\Rightarrow & (\exists\, in', out' : \operatorname{seq} M \bullet \quad in' = front\ in \land out' = \langle last\ in \rangle \frown out \\
& \qquad\qquad\qquad\qquad \land\ s' = in' \frown out') \\
& [\text{falls } in \neq \langle\,\rangle] \\
\Leftrightarrow & (\exists\, in', out' : \operatorname{seq} M \bullet \texttt{Attempt.com\_}\ pass \land Rep')
\end{array}
$$

Damit ist alles bewiesen. □

6.3 Eine realistischere Spezifikation

Nachrichten $m \in M$ werden üblicherweise nicht direkt vom Sender zum Emfänger geleitet, sondern passieren auf ihrem Weg mehrere Zwischenstationen, bis sie den Empfänger erreichen. Sei

$$[SPC]$$

eine Menge von Signalling Point Codes, deren Elemente Stationen in einem Netzwerk identifizieren, über das die Nachrichten m übertragen werden. Werden Nachrichten über das Netz verschickt, so müssen sie dabei eine bestimmte Route *route* benutzen. Diese Route ist eine Sequenz von verschiedenen *SPC*s. Jede Station der Route kann Nachrichten empfangen und senden. Die Länge einer Route ist durch die Konstante *RouteSize* gegeben.

$$
\begin{array}{|l}
\hline
RouteSize : \mathbb{N} \\
\hline
RouteSize > 0 \\
\hline
\end{array}
$$

Die Spezifikation `Sections` auf der nächsten Seite formalisiert das gerade beschriebene System. Über den Kanal *transmit* können Nachrichten verschickt werden. Diese Nachrichten passieren auf ihrem Weg zum Empfänger eine Reihe von Sektionen, in denen sie zwischengespeichert werden. Das Weiterreichen der Nachrichten von einer Sektion zur nächsten wird durch den

spec Sections

channel $transmit, receive : M$
local_channel $daemon$: signal

State

$route : \mathrm{iseq}\, SPC$
$ins : \mathrm{seq}\, \mathrm{seq}\, M$

$route \neq \langle\rangle \land \#ins = \#route$
$\#route = RouteSize$

Init_State

State′

$\forall i : \mathrm{dom}\, route' \bullet$
$ins'(i) = \langle\rangle$

com_ *transmit*

ΔState
$@transmit : M$

$route' = route$
$head\ ins' = \langle @transmit \rangle \frown (head\ ins)$
$tail\ ins' = tail\ ins$

com_ *receive*

ΔState
$@receive : M$

$route' = route$
$(last\ ins') \frown \langle @receive \rangle = last\ ins$
$front\ ins' = front\ ins$

com_ *daemon*

ΔState

$route' = route$
$\exists i : 1 \,..\, \#route - 1 \mid ins(i) \neq \langle\rangle \bullet$
$ins'(i) = front\ ins(i)$
[Die älteste Nachricht aus Sektion i wird entfernt.]
$\land\ ins'(i+1) = \langle last\ ins(i) \rangle \frown ins(i+1)$
[Die Nachricht wird an Sektion i+1 weitergereicht.]
$\land \forall j : \mathrm{dom}\, route \mid j \notin \{i, i+1\} \bullet ins'(j) = ins(j)$
[Alle anderen Sektionen bleiben unverändert.]

end_spec Sections

lokalen Kanal *daemon* modelliert. Ist eine Nachricht in der letzten Sektion angekommen, so kann sie über den Kanal *receive* empfangen werden.

Die Vorbedingungen der Operationsschemata der Spezifikation `Sections` lauten:

$$\begin{aligned}
&\text{pre } \texttt{Sections.com_}\ \mathit{transmit} \equiv \\
&\quad [\texttt{Sections.State};\ @\mathit{transmit} : M \mid \text{true}] \\
&\text{pre } \texttt{Sections.com_}\ \mathit{receive} \equiv \\
&\quad [\texttt{Sections.State};\ @\mathit{receive} : M \mid \\
&\qquad \mathit{last\ ins} \neq \langle\rangle \land \mathit{last}(\mathit{last\ ins}) = @\mathit{receive}] \\
&\text{pre } \texttt{Sections.com_}\ \mathit{daemon} \equiv \\
&\quad [\texttt{Sections.State} \mid \exists\, i : 1 \,..\, \#\mathit{route} - 1 \bullet \mathit{ins}(i) \neq \langle\rangle]
\end{aligned}$$

6.3.1 Die Realistischere verfeinert die Einfache

Es gilt `Simple` $\sqsubseteq$ `Sections`. Diese Aussage kann auch herangezogen werden, um eine Äquivalenz zwischen den beiden Spezifikationen `Attempt` und `Simple` zu erhalten.

Sei `System` die Spezifikation, die syntaktisch mit `Sections` übereinstimmt, außer daß der Kanal *daemon* ein sichtbarer Kanal von `System` ist, also

$$\texttt{System} \setminus \{\mathit{daemon}\} \equiv \texttt{Sections}.$$

Für die Spezifikation `System` muß nachgewiesen werden, daß sie schwach $\{\mathit{daemon}\}$-terminal ist, damit eine der Verfeinerungsregeln zur Einführung lokaler Kanäle angewandt werden kann.

Bemerkung 6.2 *Die Spezifikation* `System` *ist schwach* $\{\mathit{daemon}\}$*-terminal.*

Beweis. Der folgende Term dient als Variante. Den Nachrichten, die sich in den einzelnen Sektionen befinden sind Gewichte zugeordnet, die zum Ende der Route kleiner werden.

$$\sum_{i=1}^{\mathit{RouteSize}-1} \#(\mathit{ins}(i)) \cdot (\mathit{RouteSize} - i)$$

Als Konstante für die Initialisierung kommt $k = 0$ in Frage. Die Funktionen für die sichtbaren Kommunikationen sind k_{transmit} und k_{receive}.

term

$$\begin{array}{|l}
\hline
\texttt{System.State} \\
t : \mathbb{Z} \\
\hline
t = \sum_{i=1}^{\mathit{RouteSize}-1} \#(\mathit{ins}(i)) \cdot (\mathit{RouteSize} - i) \\
\hline
\end{array}$$

$$\begin{array}{|l} k_{transmit} : M \to \mathbb{N} \\ \hline \forall\, m : M \bullet k_{transmit}(m) = RouteSize - 1 \end{array}$$

$$\begin{array}{|l} k_{receive} : M \to \mathbb{N} \\ \hline \forall\, m : M \bullet k_{receive}(m) = 0 \end{array}$$

Unter Benutzung dieser Definitionen lassen sich nun die vier in Definition 5.2 angegebenen Bedingungen beweisen. Bedingung 5.2.1 ist erfüllt, da t für alle Zustände Θ`System.State` definiert ist.

Zu 5.2.2

Da alle Sektionen anfangs leer sind, ist der Wert von t nach der Initialisierung gleich 0.

$$\begin{array}{ll} \forall\, term' \bullet & \\ & \texttt{System.Init\_State} \\ \Leftrightarrow & \forall\, i : \operatorname{dom} route' \bullet ins'(i) = \langle\,\rangle \\ \Rightarrow & \sum_{i=1}^{RouteSize-1} \#(ins'(i)) \cdot (RouteSize - i) = 0 \\ \Rightarrow & 0 \leq t' \leq k \end{array}$$

Zu 5.2.3

Nach einer Kommunikation über den Kanal *transmit* können höchstens *RouteSize* viele Kommunikationen mehr über den Kanal *daemon* erfolgen.

$$\begin{array}{ll} \forall\, \Delta term;\ @transmit : M \bullet & \\ & \texttt{System.com\_}transmit \land t \geq 0 \\ \Leftrightarrow & route' = route \\ & \land\ head\ ins' = \langle @transmit \rangle \frown (head\ ins) \\ & \land\ tail\ ins' = tail\ ins \land t \geq 0 \\ \Rightarrow & \sum_{i=1}^{RouteSize-1} \#(ins'(i)) \cdot (RouteSize - i) = t + RouteSize - 1 \\ \Leftrightarrow & 0 \leq t' \leq t + k_{transmit}(@transmit) \end{array}$$

Eine Kommunikation über den Kanal *receive* ändert die Anzahl der über den Kanal *daemon* möglichen Kommunikationen nicht.

$$
\begin{array}{ll}
\forall \Delta term;\ @receive : M \bullet & \\
\quad \texttt{System.com\_}\ receive \land t \geq 0 & \\
\Leftrightarrow \quad route' = route & \\
\quad \land\ (last\ ins') \frown \langle @receive \rangle = last\ ins & \\
\quad \land\ front\ ins' = front\ ins \land t \geq 0 & \\
\Rightarrow \quad \sum_{i=1}^{RouteSize-1} \#(ins'(i)) \cdot (RouteSize - i) = t & \\
\Leftrightarrow \quad 0 \leq t' \leq t + k_{receive}(@receive) &
\end{array}
$$

Zu 5.2.4

Eine Kommunikation über den Kanal *daemon* verschiebt eine Nachricht in Richtung Ende der Route und vermindert damit ihr Gewicht.

$$
\begin{array}{ll}
\forall \Delta term \bullet & \\
\quad \texttt{System.com\_}\ daemon \land t \geq 0 & \\
\Leftrightarrow \quad t \geq 0 \land route' = route & \\
\quad \land \exists\, i : 1 \,..\, \#route - 1 \mid ins(i) \neq \langle\,\rangle \bullet & \\
\qquad ins'(i) = front\ ins(i) & \\
\qquad \land\ ins'(i+1) = \langle last\ ins(i) \rangle \frown ins(i+1) & \\
\qquad \land \forall\, j : \mathrm{dom}\ route \mid j \notin \{i, i+1\} \bullet ins'(j) = ins(j) & \\
\Rightarrow \quad \sum_{i=1}^{RouteSize-1} \#(ins'(i)) \cdot (RouteSize - i) = t - 1 \land t \geq 1 & \\
\Leftrightarrow \quad 0 \leq t' < t &
\end{array}
$$

□

Beweis der Verfeinerung

Es gilt `Simple` $\sqsubseteq$ `Sections`. Das Repräsentationsschema *Req* ist ein *LIFS*-Schema von `Simple` nach `Sections`.

Req

`Simple.State`
`Sections.State`
$s = \frown / ins$

Die vier Bedingungen "Initialisierung", "Definiertheit", "sichtbarer Fortschritt" und "unsichtbarer Fortschritt" der Regel **LIFS** sind zu beweisen.

Initialisierung

Die Spezifikation `Sections` initialisiert alle Sektionen $ins(i)$ mit der leeren Sequenz, und `Simple` initialisiert den Puffer s mit der leeren Sequenz.

$$
\begin{array}{ll}
\forall \texttt{Sections.State}'; \texttt{Simple.State}' \bullet & \\
\quad \texttt{Sections.Init\_State} & \\
\Leftrightarrow & (\forall i : \mathrm{dom}\, route' \bullet ins'(i) = \langle\,\rangle) \\
\Rightarrow & \frown\!/\, ins'(i) = \langle\,\rangle \\
& [\text{da } \mathrm{dom}\, route' = \mathrm{dom}\, ins'] \\
\Leftrightarrow & (\exists s' : \mathrm{seq}\, M \bullet s' = \langle\,\rangle \wedge s' = \frown\!/\, ins'(i)) \\
\Leftrightarrow & (\exists s' : \mathrm{seq}\, M \bullet \texttt{Simple.Init\_State} \wedge Req')
\end{array}
$$

Definiertheit

Die Definiertheitsbedingung ist für die beiden Kanäle *transmit* und *receive* zu beweisen.

Definiertheit von *transmit*

Kann der Kanal *transmit* der Spezifikation `Simple` an einer Kommunikation teilnehmen, dann kann es auch der Kanal *transmit* der Spezifikation `Sections`.

$$
\begin{array}{ll}
\forall \texttt{Sections.State}; \texttt{Simple.State}; @transmit : M & \\
\quad Req \wedge \mathrm{pre}\, \texttt{Simple.com\_}\, transmit & \\
\Leftrightarrow & s = \frown\!/\, ins \wedge \mathrm{true} \\
\Rightarrow & \mathrm{true} \\
\Leftrightarrow & \mathrm{pre}\, \texttt{Sections.com\_}\, transmit
\end{array}
$$

Definiertheit von *receive*

Kann der Kanal *receive* der Spezifikation `Simple` an einer Kommunikation teilnehmen, dann kann der Kanal *receive* der Spezifikation `Sections` ebenfalls an einer Kommunikation teilnehmen, oder es kann eine Kommunikation über den lokalen Kanal *daemon* auftreten.

$$
\begin{array}{ll}
\forall \texttt{Sections.State}; \texttt{Simple.State}; @receive : M & \\
\quad Req \wedge \mathrm{pre}\, \texttt{Simple.com\_}\, receive & \\
\Leftrightarrow & s = \frown\!/\, ins \wedge s \neq \langle\,\rangle \wedge last\, s = @receive \\
\Rightarrow & \quad (last\, ins \neq \langle\,\rangle \wedge last(last\, ins) = @receive) \\
& \vee (\exists i : 1 \,..\, \#route - 1 \bullet ins(i) \neq \langle\,\rangle) \\
\Leftrightarrow & \mathrm{pre}\, \texttt{Sections.com\_}\, receive \vee \mathrm{pre}\, \texttt{Sections.com\_}\, daemon
\end{array}
$$

Sichtbarer Fortschritt

Die sichtbaren Kanäle der Spezifikationen sind *transmit* und *receive*.

Sichtbarer Fortschritt von *transmit*

Der Kommunikationswert *@transmit* wird an den Anfang von s bzw. $\frown/\, ins$ gehängt.

$$
\begin{aligned}
&\forall \Delta \texttt{Sections.State}; \texttt{Simple.State}; @transmit : M \bullet \\
&\quad Req \land \texttt{Sections.com\_}\, transmit \\
\Leftrightarrow\ &\quad s = \frown/\, ins \land route' = route \\
&\quad \land\, head\, ins' = \langle @transmit \rangle \frown (head\, ins) \\
&\quad \land\, tail\, ins' = tail\, ins \\
\Rightarrow\ &\langle @transmit \rangle \frown s = \frown/\, ins' \\
\Leftrightarrow\ &(\exists\, s' : \operatorname{seq} M \bullet s' = \langle @transmit \rangle \frown s \land s' = \frown/\, ins') \\
\Leftrightarrow\ &(\exists\, s' : \operatorname{seq} M \bullet \texttt{Sections.com\_}\, transmit \land Req')
\end{aligned}
$$

Sichtbarer Fortschritt von *receive*

Der Kommunikationswert *@receive* wird vom Ende von s bzw. $\frown/\, ins$ entfernt.

$$
\begin{aligned}
&\forall \Delta \texttt{Sections.State}; \texttt{Simple.State}; @receive : M \bullet \\
&\quad Req \land \texttt{Sections.com\_}\, receive \\
\Leftrightarrow\ &\quad s = \frown/\, ins \land route' = route \\
&\quad \land\, (last\, ins') \frown \langle @receive \rangle = last\, ins \\
&\quad \land\, front\, ins' = front\, ins \\
\Rightarrow\ &s = (\frown/\, ins') \frown \langle @receive \rangle \\
\Leftrightarrow\ &(\exists\, s' : \operatorname{seq} M \bullet s' \frown \langle @receive \rangle = s \land s' = \frown/\, ins') \\
\Leftrightarrow\ &(\exists\, s' : \operatorname{seq} M \bullet \texttt{Sections.com\_}\, receive \land Req')
\end{aligned}
$$

Unsichtbarer Fortschritt

Kommunikationen über den lokalen Kanal *daemon* von `Sections` können von der Spezifikation `Simple` durch die Identische Operation simuliert werden.

$$
\begin{aligned}
&\forall \Delta \texttt{Sections.State}; \texttt{Simple.State}; @daemon : M \bullet \\
&\quad Req \land \texttt{Sections.com\_}\, daemon
\end{aligned}
$$

$$
\begin{array}{ll}
\Leftrightarrow & s = \frown/\, ins \land route' = route \\
& \land\, (\exists\, i : 1 \,..\, \#route - 1 \mid ins(i) \neq \langle\,\rangle \bullet \\
& \qquad ins'(i) = front\, ins(i) \\
& \qquad \land\, ins'(i+1) = \langle last\, ins(i)\rangle \frown ins(i+1) \\
& \qquad \land\, \forall\, j : \mathrm{dom}\, route \mid j \notin \{i, i+1\} \bullet ins'(j) = ins(j) \\
\Rightarrow & s = \frown/\, ins \land (\frown/\, ins' = \frown/\, ins) \\
\Rightarrow & (\exists\, s' : \mathrm{seq}\, M \bullet s' = s \land s' = \frown/\, ins' \\
\Leftrightarrow & (\exists\, s' : \mathrm{seq}\, M \bullet \Xi \texttt{Simple.State} \land Req')
\end{array}
$$

Insgesamt ist nun $\texttt{Simple} \sim_{LIFS} \texttt{Sections}$ gezeigt, und damit

$$\texttt{Simple} \sqsubseteq \texttt{Sections}.$$

Da bereits bekannt ist, daß

$$\texttt{Attempt} \sqsubseteq \texttt{Simple},$$

folgt wegen der Transitivität von $\sqsubseteq$:

$$\texttt{Attempt} \sqsubseteq \texttt{Sections}.$$

□

Schluß

In [WD96] werden große Teile der Datenverfeinerungstheorie aus dem 2. Kapitel behandelt. Die Darstellungsweise dort wirkt jedoch ad hoc. Hier wird durch die Semantikdefinition der Z-Datentypen versucht, eine schlüssige abgerundete Darstellung zu erreichen. Die Darstellungen des 3. Kapitels werden von Josephs [Jos88] in ähnlicher Form für State-Transition-Systeme behandelt. Die Ergebnisse sind dieselben und die Regeln für die Vorwärtssimulation und Rückwärtssimulation sind im wesentlichen gleich.

Butler definiert in [But92] Regeln zur Datenverfeinerung in Action-Systemen, die lokale Kanäle einbezieht. Die Regeln **LIFS** und **LIBS** aus Kapitel 5 wurden mit Hilfe der Regeln aus [But92] formuliert. Die Untersuchungen zur Kommutativität werden in keiner mir bekannten Arbeit durchgeführt. In der Arbeit von Butler ist der Hiding-Operator uneingeschränkt kommutativ, da er zum Infinite-Traces-Modell von CSP übergegangen ist.

Regeln zur Elimination lokaler Kanäle werden in keiner anderen Arbeit angegeben. Es scheint mir aber sinnvoll zu sein, auch solche Verfeinerungsregeln zur Verfügung zu haben (siehe Fallstudie B).

Die Regel **LEBS** ließe sich noch etwas schwächer formulieren, wenn nicht nur schwache Terminalität der abstrakten Spezifikation verlangt würde, sondern auch *inverse* schwache Terminalität. Das heißt, wenn die Operationsschemata der abstrakten Spezifikation rückwärts ausgeführt würden, dann müßten sie sich wieder schwach terminal verhalten. Daß **LEBS** sich nicht so allgemein formulieren läßt wie **LEFS**, ist ein Hinweis darauf, daß Vollständigkeit der Kalkülregeln in dieser Situation nicht zu erreichen ist.

Danke

Es ist gut, daß Clemens Fischer an einem guten Gelingen dieser Arbeit interessiert war. Seine Kritik und die Diskussionen waren notwendig, um inhaltliche Schwächen aufzudecken und mich an den Schreibtisch zu bewegen. Während seiner Abwesenheit hat Herr Olderog diese Aufgabe übernommen. Beiden bin ich zu Dank verpflichtet.

Anhang A

Z-Glossar

Logik

$\forall x : X \mid P(x) \bullet Q(x)$	ist äquivalent zu $\forall x : X \bullet P(x) \Rightarrow Q(x)$
$\exists x : X \mid P(x) \bullet Q(x)$	ist äquivalent zu $\exists x : X \bullet P(x) \wedge Q(x)$

Mengen

$\{x \mid P(x) \bullet A(x)\}$	Komprehension, wobei P ein Prädikat über x ist, und A ein Ausdruck über x; Die Menge $\{x \mid P(x) \bullet A(x)\}$ enthält alle $A(x)$ für die P(x) wahr ist.
$\{x \mid P(x)\}$	Komprehension: $\{x \mid P(x)\} = \{x \mid P(x) \bullet x\}$
$\{x \bullet A(x)\}$	Komprehension: $\{x \bullet A(x)\} = \{x \mid \text{true} \bullet A(x)\}$
$\mathbb{F}\, M$	Endliche Teilmengen von M
$\#M$	Anzahl der Elemente der endlichen Menge M
$\overline{M}$	Komplement von M. Ist $M : \mathbb{P}\, X$, dann $\overline{M} == X \setminus M$.
$n_1 \mathinner{..} n_2$	Zahlenbereich $(n_1, n_2 : \mathbb{Z})$ $n_1 \mathinner{..} n_2 = \{k : \mathbb{Z} \mid n_1 \leq k \leq n_2\}$

Relationen

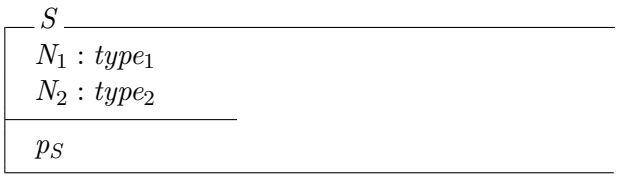
$\mathrm{dom}\, R$	Domain einer Relation R
$\mathrm{ran}\, R$	Range einer Relation R
$R \fatsemi S$	Relationale Komposition $x \mapsto y \in R \land y \mapsto z \in S \Leftrightarrow x \mapsto z \in (R \fatsemi S)$
$M \lhd R$	Domain-Restriktion $x \mapsto y \in R \land x \in M \Leftrightarrow x \mapsto y \in (M \lhd R)$
$R \rhd M$	Range-Restriktion $x \mapsto y \in R \land y \in M \Leftrightarrow x \mapsto y \in (R \rhd M)$
$M \lhd\!\!\!\cdot\, R$	Domain-Anti-Restriktion $x \mapsto y \in R \land x \notin M \Leftrightarrow x \mapsto y \in (M \lhd\!\!\!\cdot\, R)$
$R \rhd\!\!\!\cdot\, M$	Range-Anti-Restriktion $x \mapsto y \in R \land y \notin M \Leftrightarrow x \mapsto y \in (R \rhd\!\!\!\cdot\, M)$

Sequenzen

$\mathrm{seq}\, X$	Sequenzen von Elementen von X. Ist $t : seqX$, so ist $head\, t$ das erste Element von t und $last\, t$ das letzte Element. Es gilt: $t = \langle head\, t \rangle \frown (tail\, t)$ und $t = (front\, t) \frown \langle last\, t \rangle$.
$s \frown t$	Konkatentation zweier Sequenzen $(\#(s \frown t) = \#s + \#t)$
$\frown / s$	verteilte Konkatenation ("flatten")
$\langle\, \rangle$	Leere Sequenz
$\langle x_1, \ldots, x_n \rangle$	Schreibweise für Elemente von Sequenzen
$s \downarrow M$	Restriktion von s auf M. Es werden alle $s(i)$ mit $s(i) \notin M$ aus der Sequenz s entfernt. Das Ergebnis ist wieder eine Sequenz.
$s \uparrow M$	Anti-Restriktion von s auf M. Es werden alle $s(i)$ mit $s(i) \in M$ aus der Sequenz s entfernt. Das Ergebnis ist wieder eine Sequenz.

Schemakalkül

$\exists$ Existentielle Quantifikation eines Schemas

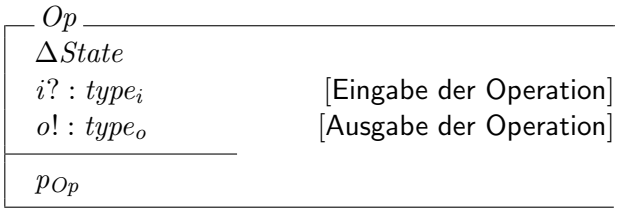


$\exists N_2 \bullet S = [\, N_1 : type_1 \mid \exists N_2 : type_2 \bullet p_S \,]$

Δ Abkürzende Schreibweise $\Delta S \mathrel{\widehat=} [\, S;\ S' \,]$

$[N_1/N_2]$ Umbenennung von Komponentennamen

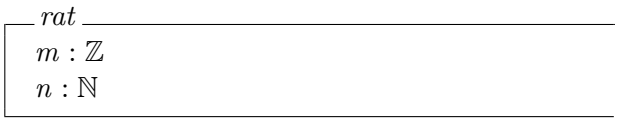
pre Vorbedingung eines Operationsschemas Op



$\text{pre}\, Op = \exists\, State';\ o! : type_o \bullet Op$

$\fatsemi$ Sequentielle Komposition von Operationsschemata
$Op_1 \mathrel{\widehat=} [\, \Delta State \mid p_{Op_1} \,]$
$Op_2 \mathrel{\widehat=} [\, \Delta State \mid p_{Op_2} \,]$
$Op_1 \fatsemi Op_2 =$
$\quad (Op_1[State''/State'] \land Op_2[State''/State]) \setminus (State'')$

Θ Die charakteristische Bindung bindet Komponentennamen an Werte.



$\Theta rat = \langle\!|\ m \rightsquigarrow m, n \rightsquigarrow n\ |\!\rangle$; Die Namen m und n auf der rechten Seiten der Pfeile müssen in der Umgebung des Θ-Ausdrucks deklariert sein, also Werte haben. Eine mögliche *Bindung* ist $\langle\!|\ m \rightsquigarrow -1, n \rightsquigarrow 1\ |\!\rangle$.

Ξ Abürzende Schreibweise $\Xi S \mathrel{\widehat=} [\, \Delta S \mid \Theta S = \Theta S' \,]$

$\setminus$ Hiding, dh. verstecken von Komponenten
$S \setminus (N_1) \mathrel{\widehat=} \exists N_1 : type_1 \bullet S$

Anhang B

Semantikdefinition der CSP-Z-Operatoren

Die Semantikdefinition ist aus [Fis96] entnommen.
Seien S und T Spezifikationen.

Interne und externe Wahl

$$
\begin{array}{rcl}
\mathcal{I}[\![\mathtt{S} \sqcap \mathtt{T}]\!] & = & \mathcal{I}[\![\mathtt{S} \,\Box\, \mathtt{T}]\!] = \mathcal{I}[\![\mathtt{S}]\!] (= \mathcal{I}[\![\mathtt{T}]\!]) \\
\mathcal{D}[\![\mathtt{S} \sqcap \mathtt{T}]\!] & = & \mathcal{D}[\![\mathtt{S} \,\Box\, \mathtt{T}]\!] = \mathcal{D}[\![\mathtt{S}]\!] \cup \mathcal{D}[\![\mathtt{T}]\!] \\
\mathcal{F}[\![\mathtt{S} \sqcap \mathtt{T}]\!] & = & \mathcal{F}[\![\mathtt{S}]\!] \cup \mathcal{F}[\![\mathtt{T}]\!] \\
\mathcal{F}[\![\mathtt{S} \,\Box\, \mathtt{T}]\!] & = & \{(\langle\rangle, X) \mid \quad ((\langle\rangle, X) \in \mathcal{F}[\![\mathtt{S}]\!] \cap \mathcal{F}[\![\mathtt{T}]\!]) \\
 & & \qquad\qquad \vee\, (\mathcal{F}[\![\mathtt{T}]\!] = \varnothing \wedge (\langle\rangle, X) \in \mathcal{F}[\![\mathtt{S}]\!]) \\
 & & \qquad\qquad \vee\, (\mathcal{F}[\![\mathtt{S}]\!] = \varnothing \wedge (\langle\rangle, X) \in \mathcal{F}[\![\mathtt{T}]\!])\} \\
 & & \cup\, \{(tr, X) \mid tr \neq \langle\rangle \wedge (tr, X) \in \mathcal{F}[\![\mathtt{S}]\!] \cup \mathcal{F}[\![\mathtt{T}]\!]\} \\
 & & \cup\, \{(\langle\rangle, X) \mid \langle\rangle \in \mathcal{D}[\![\mathtt{S} \,\Box\, \mathtt{T}]\!]\}
\end{array}
$$

Umbenennung

Sei f eine Funktion von Kanalnamen in Kanalnamen mit

$$f(c_1) = f(c_2) \Rightarrow type_{\mathcal{I}[\![\mathtt{S}]\!]}(c_1) = type_{\mathcal{I}[\![\mathtt{S}]\!]}(c_2).$$

Für f wird folgende Schreibweise definiert

$$\forall (c, v) : Comm(\mathcal{I}[\![\mathtt{S}]\!]) \bullet f(c, v) = (f(c), v).$$

(Und analog auch für Sequenzen und Teilmengen von $Comm(\mathcal{I}[\![\mathtt{S}]\!])$)

$$
\begin{array}{lcl}
\mathcal{I}[\![f(\mathtt{S})]\!] & = & f(\mathcal{I}[\![\mathtt{S}]\!]) \\
\mathcal{D}[\![f(\mathtt{S})]\!] & = & \{f(tr) \mid tr \in \mathcal{D}[\![\mathtt{S}]\!]\} \\
\mathcal{F}[\![f(\mathtt{S})]\!] & = & \{(f(tr), X) \mid (tr, f^{-1}(X)) \in \mathcal{F}[\![\mathtt{S}]\!]\}
\end{array}
$$

Hiding

Sei $H \subseteq \mathit{Chans}(\mathcal{I}[\![\mathtt{S}]\!])$.

$$
\begin{array}{lcl}
\mathcal{I}[\![\mathtt{S} \setminus H]\!] & = & \mathcal{I}[\![\mathtt{S}]\!] \upharpoonright H \\
\mathcal{D}[\![\mathtt{S} \setminus H]\!] & = & \quad \{(tr \upharpoonright H) \frown sr \mid \forall\, n : \mathbb{N} \bullet \exists\, u : \mathrm{seq}\, \mathit{Comm}(H) \bullet \\
 & & \qquad\qquad \#u \geq n \land tr \frown u \in \mathit{traces}(\mathtt{S})\} \\
 & & \cup\, \{tr \upharpoonright H \mid tr \in \mathcal{D}[\![\mathtt{S}]\!]\} \\
\mathcal{F}[\![\mathtt{S} \setminus H]\!] & = & \quad \{(tr \upharpoonright H, X) \mid (tr, X \cup \mathit{Comm}(H)) \in \mathcal{F}[\![\mathtt{S}]\!]\} \\
 & & \cup\, \{(tr, X) \mid tr \in \mathcal{D}[\![\mathtt{S} \setminus H]\!]\}
\end{array}
$$

Parallelkomposition

Sei $K \subseteq \mathit{Chans}(\mathcal{I}[\![\mathtt{S}]\!]) \cup \mathit{Chans}(\mathcal{I}[\![\mathtt{T}]\!])$.

$$
\begin{array}{lcl}
\mathcal{I}[\![\mathtt{S} \,|[\, K \,]|\, \mathtt{T}]\!] & = & \mathcal{I}[\![\mathtt{S}]\!] \cup \mathcal{I}[\![\mathtt{T}]\!] \\
\mathcal{D}[\![\mathtt{S} \,|[\, K \,]|\, \mathtt{T}]\!] & = & \{tr \frown v \mid \exists\, sr : \mathrm{seq}\, \mathit{Comm}(\mathit{Chans}(\mathcal{I}[\![\mathtt{S}]\!])); \\
 & & \qquad ur : \mathrm{seq}\, \mathit{Comm}(\mathit{Chans}(\mathcal{I}[\![\mathtt{T}]\!])) \mid \\
 & & \qquad sr \in \mathit{traces}(\mathtt{S}) \land ur \in \mathit{traces}(\mathtt{T}) \bullet \\
 & & \qquad\quad (sr \in \mathcal{D}[\![\mathtt{S}]\!] \lor ur \in \mathcal{D}[\![\mathtt{T}]\!]) \\
 & & \qquad \land\, tr \in \mathit{merge}_{K, \mathcal{I}[\![\mathtt{S}]\!] \cup \mathcal{I}[\![\mathtt{T}]\!]}(sr, ur)\} \\
\mathcal{F}[\![\mathtt{S} \,|[\, K \,]|\, \mathtt{T}]\!] & = & \quad \{(tr, X \cup Y) \mid \exists\, sr : \mathrm{seq}\, \mathit{Comm}(\mathit{Chans}(\mathcal{I}[\![\mathtt{S}]\!])); \\
 & & \qquad\qquad ur : \mathrm{seq}\, \mathit{Comm}(\mathit{Chans}(\mathcal{I}[\![\mathtt{T}]\!])) \mid \\
 & & \qquad\qquad (sr, X) \in \mathcal{F}[\![\mathtt{S}]\!] \land (ur, Y) \in \mathcal{F}[\![\mathtt{T}]\!] \bullet \\
 & & \qquad\qquad tr \in \mathit{merge}_{K, \mathcal{I}[\![\mathtt{S}]\!] \cup \mathcal{I}[\![\mathtt{T}]\!]}(sr, ur)\} \\
 & & \cup\, \{(tr, X) \mid tr \in \mathcal{D}[\![\mathtt{S} \,|[\, K \,]|\, \mathtt{T}]\!]\}
\end{array}
$$

Die Funktion $\mathit{merge}_{K,I}(s, t)$ berechnet die Menge der "merged" Traces aus s und t bezüglich dem Synchronisationsalphabet K:

$$\mathit{merge}_{K,I}(\langle\rangle, \langle\rangle) = \{\langle\rangle\}$$

$$
\begin{aligned}
merge_{K,I}(s,t) = \{\langle (c,v)\rangle \frown tr : \mathrm{seq}\ Comm(I) \mid \\
\quad (head(s) = (c,v) \wedge c \notin K \wedge tr \in merge_{K,I}(tail(s),t)) \\
\vee\ (head(t) = (c,v) \wedge c \notin K \wedge tr \in merge_{K,I}(s,tail(t))) \\
\vee\ (\quad head(s) = head(t) = (c,v) \wedge c \in K \\
\quad \wedge\ tr \in merge_{K,I}(tail(s),tail(t)))\ \}
\end{aligned}
$$

Literaturverzeichnis

[AL91] Martìn Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82:253–284, 1991.

[BN92] Stephen Brien and John Nicholls. Z base standard. Technical report, Oxford University Computing Laboratory, Programming Research Group, 1992.

[But92] M. J. Butler. *A CSP Approach To Action Systems.* PhD thesis, Oxford University, Programming Research Group, 1992.

[But93] Michael J. Butler. Refinement and decomposition of value-passing action systems. In Eike Best, editor, *CONCUR'93: 4th International Conference on Concurrency Theory*, volume 715 of *LNCS*, pages 217–232, 1993.

[Fis96] Clemens Fischer. Combining Z and CSP. Technical report, University Of Oldenburg, 1996.

[GM89] P. H. B. Gardiner and Carroll Morgan. A single complete rule for data refinement. Technical Report PRG-TR-7-89, Oxford University Computing Laboratory, Programming Research Group, 1989.

[HHS87] C. A. R. Hoare, Jifeng He, and J. W. Sanders. Prespecification in data refinement. *Information Processing Letters*, 25:71–76, 1987.

[Hoa81] C. A. R. Hoare. A calculus of total correctness for communicating processes. Technical monograph PRG-23, Programming Research Group, Oxford University, 1981.

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

[Jos88] Mark. B. Josephs. A state-based approach to communicating processes. *Distributed Computing*, 3:9–18, 1988.

[Kin90] Steve King. Z and the refinement calculus. In D. Bjørner, C. A. R. Hoare, and H. Langmaack, editors, *VDM and Z – Formal Methods in Software Development*, volume 428 of *LNCS*, pages 164–188, 1990.

[MG90] Carroll Morgan and P. H. B. Gardiner. Data refinement by calculation. *Acta Informatica*, 27:481–503, 1990.

[Mor88] Carroll Morgan. Auxiliary variables in data refinement. *Information Processing Letters*, 29:293–296, 1988.

[Mor90] Carroll Morgan. Of wp and CSP. In W. H. J. Feijen, A. J. M. van Gasteren, D. Gries, and J. Misra, editors, *Beauty is our business: a birthday salute to Edsger W. Dijkstra*. Springer-Verlag, 1990.

[Mor94] Carroll Morgan. *Programming from Specifications*. Prentice Hall International, 2 edition, 1994.

[Rö94] S. Rössig. *A Transformational Approach to the Design of Communicating Systems*. PhD thesis, University Of Oldenburg, 1994.

[Ros88a] A. W. Roscoe. An alternative order for the failures model. Technical monograph PRG-67, Programming Research Group, Oxford University, 1988.

[Ros88b] A. W. Roscoe. Unbounded nondeterminism in CSP. Technical monograph PRG-67, Programming Research Group, Oxford University, 1988.

[Smi96] Graeme Smith. A semantic integration of Object-Z and CSP for the specification of concurrent systems. Technical report, Technische Universität Berlin, 1996.

[Spi92] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International, 1992.

[WD96] Jim Woodcock and Jim Davies. *Using Z*. Prentice Hall International, 1996.

[WM90] J. C. P. Woodcock and C. C. Morgan. Refinement of state-based concurrent systems. In D. Bjørner, C. A. R. Hoare, and H. Langmaack, editors, *VDM and Z – Formal Methods in Software Development*, volume 428 of *LNCS*, pages 340–351, 1990.