Technische Universität Dresden

Fakultät Informatik

Großer Beleg

zum Thema

Lineare Logik und Prozeßtheorie

von
Hendrik Tews

Eingereicht am 22. Februar 1995

Verantwortlicher Hochschullehrer: Prof. Dr. rer. nat. habil. H. Reichel

Dieser Beleg wurden mit den Textsatzsystem LaTeX erstellt. Die kategorientheoretischen Diagramme und Beweisbäume wurden mit Hilfe von Makropaketen von Paul Taylor (Commutative diagrams, Version 3.82) und Makoto Tatsuta (proof.sty, Version 1.0) erstellt.

Inhaltsverzeichnis

1. Einführung

Funktionale Programmiersprachen konnten durch die Einführung statischer Typsysteme wesentlich weiterentwickelt werden. Eines der bemerkenswerten Ergebnisse in diesem Zusammenhang ist der Curry-Howard-Isomorphismus [Gir89], der es gestattet, den Typ eines Programms als Formel einer intuitionistischen Logik[1] und das Programm selbst als einen Beweis aufzufassen. Wesentlich für diesen Zusammenhang sind kartesisch abgeschlossene Kategorien, die sowohl ein Modell des einfach getypten λ-Kalküls [Bar92] als auch eines für intuitionistische Logik sind.

Auf dem Gebiet der parallelen Sprachen sind die Fortschritte bescheidener geblieben. Es existieren eine Vielzahl verschiedener Kalküle zur Modellierung von Parallelität. Aber kein Kalkül besitzt eine ähnlich unangefochtene Stellung wie der λ-Kalkül innerhalb der funktionalen Sprachen.

Eine bisher nicht verfolgte Möglichkeit, Fortschritte bei der Modellierung paralleler Sprachen zu erzielen, besteht darin, in Analogie zu der Situation bei funktionalen Sprachen nach einem Curry-Howard-Isomorphismus für parallele Sprachen zu suchen. Mit klassischer Linearer Logik [Gir87] steht eine Logik zur Verfügung, mit der der Verbrauch

[1] Siehe zum Beispiel [FKKK94] für eine zusammenfassende Darstellung.

von Ressourcen modelliert werden kann. Durch Seely wurde in [See89] ein kategorientheoretisches Modell für Lineare Logik vorgestellt: um einen Comonad bereicherte $*$-autonome Kategorien.

In einer Reihe von Publikationen [AGN93, Abr94] entwickelt Abramsky diese Idee und faßt Prozeßspezifikationen, Lineare Logik und $*$-autonome Kategorien zu einem einheitlichen System, den Interaktionskategorien zusammen. Prozesse werden durch Morphismen der Kategorie modelliert, die Komposition von Morphismen spielt die Rolle paralleler Komposition in Verbindung mit Restriktion.

Anliegen der vorliegenden Arbeit ist es, an Beispielen zu untersuchen, wie ausdrucksstark Interaktionskategorien sind und welche Mittel zur Verfügung stehen, um Eigenschaften von Prozessen zu spezifizieren. Es wurden bewußt Beispiele ausgewählt, die bestehende Einschränkungen von Interaktionskategorien, beziehungsweise von $\mathcal{SP}roc$ verdeutlichen. Zum Vergleich, zur Vorstellung der Beispiele und zur Formalisierung von Eigenschaften der Beispiele wird der π-Kalkül [MPW92a, MPW92b] in der in [Mil91] vorgestellten Form in Verbindung mit dem μ-Kalkül [Sti92] benutzt.

Leider war die zu Interaktionskategorien zur Verfügung stehende Literatur sehr begrenzt. Die Doktorarbeit von Simon J. Gay [Gay94], die freundlicherweise vor ihrer Verteidigung zur Verfügung gestellt wurde, erhielt der Autor erst gegen Ende der vorliegenden Arbeit. Die Ergebnisse, die Gay zu $\mathcal{FP}roc$ und zu Spezifikationsstrukturen vorlegt, mußten deshalb ohne Einfluß auf diese Arbeit bleiben.

In Abschnitt zwei wird ein Teil der verwendeten Notation vorgestellt. Allerdings konnte nicht auf alle, den Vorlesungsstoff übersteigende Konzepte eingegangen werden. Zum Verständnis der Arbeit sind deshalb Kenntnisse im π-Kalkül, der klassischen Linearen Logik und in den Grundlagen der Kategorientheorie unabdinglich.

In Abschnitt drei werden Sicherheits- und Lebendigkeitseigenschaften am Beispiel eines Bahnüberganges untersucht. Abschnitt vier wendet sich einigen Aspekten von Mobilität zu. In Abschnitt fünf werden spezielle Lebendigkeitseigenschaften, nämlich erfolgreiche und erfolglose Termination, untersucht.

Der Autor möchte Ulrich Hensel, Yvonne Kalinke und Hannes Federrath für die Korrektur von unfertigen Versionen dieser Arbeit und die Beratung in Layoutfragen danken.

2. Grundlagen

In diesem Abschnitt sollen einige in der Arbeit verwendete Bezeichnungen vorgestellt werden. Desweiteren wird eine kurze Einführung in Lineare Kategorien gegeben und eine wichtige Prozeßkategorie, $\mathcal{SP}roc$, vorgestellt. Es wird vorausgesetzt, daß der Leser mit den Grundlagen der Kategorientheorie [ML71], mit dem μ-Kalkül [Sti92] und dem π-Kalkül [MPW92a, MPW92b] vertraut ist.

2.1 Bezeichnungskonventionen

Das Wort synchron beziehungsweise asynchron wird in Bezug auf Prozeßkalküle in zwei unterschiedlichen Bedeutungen gebraucht. Kalküle werden als synchron bezeichnet, falls sie Kommunikation nach dem Prinzip des „Handshaking“ modellieren. In asynchronen Kalkülen muß der Sender demnach nicht auf die Bereitschaft des Empfängers warten.

In dieser Arbeit bezeichnet synchron jedoch die Eigenschaft, daß an allen Kanälen, gleichsam von zentraler Stelle getaktet, zu jeder Zeiteinheit eine Aktion sichtbar ist. Kalküle sind asynchron, falls in einem Ableitungsschritt oder einer Zeiteinheit nur auf einem Teil der Kanäle eine Aktion abläuft. In diesem Sinne ist der π-Kalkül asynchron.

Seien Ω und Σ zwei Alphabete und $f : \Omega \longrightarrow \Sigma$ eine Funktion. Mit $f^* : \mathcal{M}on(\Omega) \longrightarrow \mathcal{M}on(\Sigma)$ soll die zwischen den freien (Wort-)Monoiden $\mathcal{M}on(\Omega)$ und $\mathcal{M}on(\Sigma)$ durch f induzierte Funktion bezeichet werden.

Ein *markiertes Transitionssystem* ist eine Tripel $(X, L, \longrightarrow)$, wobei X eine Menge von Zuständen, L eine Menge von Markierungen oder Aktionen und $\longrightarrow \subseteq X \times L \times X$ die Transitionsrelation ist. Es wird die Notation $x \xrightarrow{a} y$ anstelle $(x, a, y) \in \longrightarrow$ benutzt. Markierte Transitionssysteme können als markierte gerichtete Graphen dargestellt werden. Dabei entsprechen die Zustände den Knoten, und die Transitionsrelation wird durch die Menge der Kanten dargestellt.

Ein Transitionssystem dessen zugehöriger Graph ein Baum ist, wird auch als *Synchronisationsbaum* bezeichnet.

Sei $P = (X, L, \longrightarrow)$ ein Synchronisationsbaum mit Wurzel $x \in X$. Die Menge aller Pfade, $\mathsf{traces}(P)$, ist

$$\mathsf{traces}(P) \stackrel{\text{def}}{=} \{l_1 l_2 \ldots l_n \mid n \in \mathbb{N} \text{ und } \exists x_1 \,.\, x \xrightarrow{l_1} x_1 \text{ und } \forall i < n \,.\, \exists x_{i+1} \,.\, x_i \xrightarrow{l_{i+1}} x_{i+1}\}$$

Die disjunkte Vereinigung wird mit $\uplus$, die beiden kanonischen Injektionen mit inl und inr bezeichnet. Wenn $f : A \longrightarrow C$ und $g : B \longrightarrow C$, dann wird der durch f und g bis auf Eindeutigkeit bestimmte Morphismus $A \uplus B \longrightarrow C$ mit $[f, g]$ bezeichnet.

2.2 Kategorientheoretische Modelle für Lineare Logik

Die Semantik der Linearen Logik wird in [Gir87] im Stile eines natürlichen Deduktionskalküls mittels Ableitungsregeln definiert. In der gleichen Arbeit schlägt Girard weitere Semantiken vor, zum Beispiel kohärente Räume oder Beweisnetze, für die zum Teil Korrektheits- und Vollständigkeitsbeweise vorliegen. In dieser Arbeit spielen Semantiken für Lineare Logik auf der Basis bestimmter bereicherter Kategorien eine wichtige Rolle.

Die grundlegende Idee bei der Definition einer Semantik für Lineare Logik mittels bereicherter Kategorien besteht darin, eine Korrespondenz zwischen den Formeln der Linearen Logik und den Objekten der Kategorie herzustellen. Ableitungsbäume oder Beweise von Formeln der Linearen Logik sollen durch die Morphismen der Kategorie modelliert werden. Die folgende kurze Darstellung ist [Gay94] entlehnt.

Eine *monadische* Kategorie ist eine Kategorie $\mathbb{C}$ zusammen mit einem Bifunktor $\otimes : \mathbb{C} \times \mathbb{C} \longrightarrow \mathbb{C}$, einem ausgezeichneten Objekt I und natürlichen Isomorphismen $\mathsf{unitl}_A : I \otimes A \longrightarrow A$, $\mathsf{unitr}_A : A \otimes I \longrightarrow A$ und $\mathsf{assoc}_{A,B,C} : A \otimes (B \otimes C) \longrightarrow (A \otimes B) \otimes C$. Diese

Isomorphismen müssen folgende Bedingungen erfüllen. Die Diagramme

$$\begin{tikzcd}
A \otimes (B \otimes (C \otimes D)) \arrow[r, "\mathsf{assoc}_{A,B,C\otimes D}"] \arrow[dd, "\mathsf{id}_A \otimes \mathsf{assoc}_{B,C,D}"'] & (A \otimes B) \otimes (C \otimes D) \arrow[d, "\mathsf{assoc}_{A\otimes B,C,D}"] \\
 & ((A \otimes B) \otimes C) \otimes D \\
A \otimes ((B \otimes C) \otimes D) \arrow[r, "\mathsf{assoc}_{A,B\otimes C,D} \otimes id_D"'] & (A \otimes (B \otimes C)) \otimes D \arrow[u, "\mathsf{assoc}_{A,B,C} \otimes \mathsf{id}_D"']
\end{tikzcd} \tag{2.1}$$

$$\begin{tikzcd}
A \otimes (I \otimes B) \arrow[rr, "\mathsf{assoc}_{A,I,B}"] \arrow[dr, "\mathsf{id}_A \oplus \mathsf{unitl}_B"'] & & (A \otimes I) \otimes B \arrow[dl, "\mathsf{unitr}_A \oplus \mathsf{id}_B"] \\
 & A \otimes B &
\end{tikzcd} \tag{2.2}$$

$$\begin{tikzcd}
I \otimes I \arrow[rr, "\mathsf{unitl}_I"] \arrow[dr, "\mathsf{unitr}_I"'] & & I \arrow[dl, equal] \\
 & I &
\end{tikzcd} \tag{2.3}$$

kommutieren für alle Objekte A, B, C, D aus $\mathbb{C}$. Verschiedene Autoren (z.B. [Poi92]) nennen Diagramm (2.3) nicht. Laut [EK65] folgt (2.3) aus den vorangehenden Axiomen.

Natürliche Isomorphismen heißen *kohärent*, wenn (etwas vereinfacht formuliert) alle Diagramme, die aus ihnen, ihren Inversen und aus Identitäten zusammengesetzt sind, kommutieren. Die Isomorphismen unitl, unitr und assoc sind kohärent (siehe [ML71] für eine ausführlichere Diskussion von Kohärenz und einen Beweis).

Eine *symmetrische* monadische Kategorie ist eine monadische Kategorie $\mathbb{C}$ mit einem natürlichen Isomorphismus $\mathsf{symm}_{A,B} : A \otimes B \longrightarrow B \otimes A$ für den gilt: Die Diagramme

$$\begin{tikzcd}
A \otimes (B \otimes C) \arrow[r, "\mathsf{assoc}_{A,B,C}"] \arrow[d, "\mathsf{id}_A \otimes \mathsf{symm}_{B,C}"'] & (A \otimes B) \otimes C \arrow[d, "\mathsf{symm}_{A\otimes B,C}"] \\
A \otimes (C \otimes B) \arrow[d, "\mathsf{assoc}_{A,C,B}"'] & C \otimes (A \otimes B) \arrow[d, "\mathsf{assoc}_{C,A,B}"] \\
(A \otimes C) \otimes B \arrow[r, "\mathsf{symm}_{A,C} \otimes \mathsf{id}_B"'] & (C \otimes A) \otimes B
\end{tikzcd} \tag{2.4}$$

$$
\begin{tikzcd}
A \otimes B \arrow[rr, "\mathsf{symm}_{A,B}"] \arrow[dr, equal] & & B \otimes A \arrow[dl, "\mathsf{symm}_{A,B}"] \\
& A \otimes B &
\end{tikzcd}
\quad (2.5)
\qquad
\begin{tikzcd}
A \otimes I \arrow[rr, "\mathsf{symm}_{A,I}"] \arrow[dr, "\mathsf{unitr}_A"'] & & I \otimes A \arrow[dl, "\mathsf{unitl}_A"] \\
& A &
\end{tikzcd}
\quad (2.6)
$$

kommutieren für alle Objekte A, B, C aus $\mathbb{C}$.

Eine symmetrische monadische Kategorie $\mathbb{C}$ heißt *abgeschlossen*, falls für jedes Objekt X aus $\mathbb{C}$ der Funktor $-\otimes X : \mathbb{C} \longrightarrow \mathbb{C}$ einen rechtsadjungierten Funktor $X \multimap -$ besitzt[2]. Dieser Funktor läßt sich auf natürliche Weise zu einem Bifunktor $- \multimap - : \mathbb{C} \times \mathbb{C} \longrightarrow \mathbb{C}$ erweitern und wird als interner Hom-Funktor bezeichnet. Mit Hilfe dieses Funktors läßt sich eine Korrespondenz zwischen den Morphismen $f : X \longrightarrow Y$ und den „Elementen" $f^{\#}$ des Objektes $X \multimap Y$ herstellen:

$$
\begin{array}{cl}
X \xrightarrow{\quad f \quad} Y & \\
\hline
I \otimes X \xrightarrow{\mathsf{unitl}_X \, ; \, f} Y & \mathsf{unitl} \\
\hline
I \xrightarrow{\quad f^{\#} \quad} X \multimap Y & (-\otimes X) \dashv (X \multimap -)
\end{array}
\tag{2.7}
$$

Eine symmetrische monadische Kategorie, die abgeschlossen ist, wird auch als *autonom* bezeichnet.

Im folgenden werden zusätzliche Isomorphismen eingeführt, mit denen sich die Konnektoren der Linearen Logik interpretieren lassen. Betrachten wir zuerst die Lineare Negation: Ein Objekt $\bot$, für das der Funktor $(-)^{\bot} \stackrel{\mathsf{def}}{=} - \multimap \bot : \mathbb{C} \longrightarrow \mathbb{C}$ eine Äquivalenz von Kategorien [ML71] ist, heißt *dualisierendes Objekt*.

Eine autonome Kategorie mit einem dualisierenden Objekt heißt $*$*-autonome Kategorie*.

Zur Interpretation des Konnektors $⅋$ wird ein Bifunktor $⅋ : \mathbb{C}\times\mathbb{C} \longrightarrow \mathbb{C}$ mit Hilfe der Dualität definiert: $A ⅋ B \stackrel{\mathsf{def}}{=} (A^{\bot} \otimes B^{\bot})^{\bot}$. Ein so definierter Funktor liefert eine zweite monadische Struktur auf $\mathbb{C}$. Eine ausführlichere Beschreibung und ein Beweis findet sich in [MOM90].

Für eine Modellierung der additiven Konnektoren muß die Kategorie Produkte ($\&$) und Coprodukte ($\oplus$) enthalten. Das terminale Objekt wird mit $\top$ bezeichnet. Zu beachten ist, daß eine $*$-autonome Kategorie mit endlichen Produkten und einem dualisierenden Objekt immer auch Summen beziehungsweise Coprodukte enthält. Diese erhält man mit den Definitionen $A \oplus B \stackrel{\mathsf{def}}{=} (A^{\bot} \,\&\, B^{\bot})^{\bot}$ und $\mathbf{0} \stackrel{\mathsf{def}}{=} \top^{\bot}$.

Ein *Comonad* $(!, \epsilon, \delta)$ besteht aus einem Funktor $! : \mathbb{C} \longrightarrow \mathbb{C}$ und zwei natürlichen Isomorphismen $\epsilon : \,! \longrightarrow \mathsf{Id}$ und $\delta : \,! \longrightarrow\, !!$, so daß die folgenden Diagramme kommutieren:

[2] In anderen nicht zu Linearer Logik in Beziehung stehenden Publikationen wird dieser Funktor gewöhnlich mit $(-)^X$ bezeichnet.

$$
\begin{tikzcd}
! \arrow[r, "\delta"] \arrow[d, "\delta"'] & !! \arrow[d, "!\,;\,\delta"] \\
!! \arrow[r, "\delta\,;\,!"'] & !!!
\end{tikzcd}
\tag{2.8}
$$

$$
\begin{tikzcd}
 & ! \arrow[dl, equal] \arrow[d, "\delta"] \arrow[dr, equal] & \\
! & !! \arrow[l, "\epsilon\,;\,!"] \arrow[r, "!\,;\,\epsilon"'] & !
\end{tikzcd}
\tag{2.9}
$$

Ein Objekt A aus $\mathbb{C}$ ist ein *kommutativer Comonoid*, falls es zwei Morphismen $\eta_A : A \longrightarrow I$, $\mu_A : A \longrightarrow A \otimes A$ gibt, so daß die Diagramme

$$
\begin{tikzcd}
A \otimes A \arrow[d, "\mathsf{id}_A \otimes \mu_A"'] & A \arrow[l, "\mu_A"'] \arrow[r, "\mu_A"] & A \otimes A \arrow[d, "\mu_A \otimes \mathsf{id}_A"] \\
A \otimes (A \otimes A) \arrow[rr, "\mathsf{assoc}_{A,A,A}"'] & & (A \otimes A) \otimes A
\end{tikzcd}
\tag{2.10}
$$

$$
\begin{tikzcd}
 & A \arrow[dl, "\mathsf{unitl}_A^{-1}"'] \arrow[d, "\mu_A"] \arrow[dr, "\mathsf{unitr}_A^{-1}"] & \\
I \otimes A & A \otimes A \arrow[l, "\eta_A \otimes \mathsf{id}_A"] \arrow[r, "\mathsf{id}_A \otimes \eta_A"'] & A \otimes I
\end{tikzcd}
\tag{2.11}
$$

$$
\begin{tikzcd}
 & A \arrow[dl, "\mu_A"'] \arrow[dr, "\mu_A"] & \\
A \otimes A \arrow[rr, "\mathsf{symm}_{A,A}"'] & & A \otimes A
\end{tikzcd}
\tag{2.12}
$$

kommutieren.

In der Bezeichnung mit [See89] brechend, soll hier der Terminus Lineare Kategorie für eine Kategorie mit zusätzlicher Struktur zur Interpretation von ! und ? vorbehalten sein. Eine *Lineare Kategorie* ist eine $*$-autonome Kategorie mit endlichen Produkten und einem Comonad $(!, \epsilon, \delta)$, so daß für alle Objekte A aus $\mathbb{C}$ das Objekt $!A$ ein kommutativer Comonoid ist und die zusätzlichen Isomorphismen

$$
\begin{aligned}
!\,(A \,\&\, B) &\cong\ !\,A \otimes !\,B \\
!\,\mathbf{1} &\cong I
\end{aligned}
$$

gelten.

Eine Lineare Kategorie ist ein Modell für klassische Lineare Logik. Für die Interpretation von ? kann ein Funktor $? : \mathbb{C} \longrightarrow \mathbb{C}$ mittels Dualität definiert werden: $?A \stackrel{\text{def}}{=} (!\,(A^{\perp}))^{\perp}$. Die Regeln „weakening“, „contraction“ und „dereliction“ werden jeweils von den Morphismen η, μ und ϵ interpretiert. Falls $f : !A \longrightarrow B$ ist, dann kann der durch die Regel „promotion“ entstehende Morphismus $f^! : !A \longrightarrow !B$ mit $f^! \stackrel{\text{def}}{=} \delta\,;\,!f$ konstruiert werden.

2.3 Interaktionskategorien: $\mathcal{SP}roc$

Wie schon in der Einleitung erörtert, befinden sich die Arbeiten zu Interaktionskategorien noch im Anfangsstadium. So ist es zur Zeit zum Beispiel noch nicht klar, ob sich die verschiedenen Beispiele für Interaktionskategorien in einem einheitlichen Axiomensystem darstellen lassen. Sowohl in [AGN93] als auch in [Gay94] werden ähnliche Vorschläge zur Axiomatisierung unterbreitet. Beide sind jedoch nicht zufriedenstellend, da ein wichtiges Beispiel, $\mathcal{ASP}roc$, die Kategorie der asynchronen Prozesse, den Axiomen jeweils nicht genügt. Allerdings sind Prozeßkategorien gegenwärtig Gegenstand intensiver Forschung, so daß hier weiterführende Ergebnisse zu erwarten sind.

Im begrenzten Rahmen dieser Arbeit soll nur ein Beispiel einer Interaktionskategorie angegeben werden: $\mathcal{SP}roc$, die Kategorie der synchronen Prozesse. Sie wurde gewählt, weil sie einerseits in den Darstellungen zu Interaktionskategorien den größten Raum einnimmt. Andererseits bietet die für einen Vergleich mit dem π-Kalkül vielleicht geeignetere Kategorie $\mathcal{ASP}roc$ keine Struktur zur Interpretation von !. Auf ! kann aber im Abschnitt 4 nicht verzichtet werden. Und drittens lassen sich (etwas umständlicher) in $\mathcal{SP}roc$ auch asynchrone Prozesse modellieren.

Eine Möglichkeit besteht in der Konstruktion eines Delay-Monads. Allerdings ist es nicht möglich aus $\mathcal{SP}roc$ mittels dieses Monads eine Kategorie asynchroner Prozesse zu erhalten [Gay94].

Ebenfalls laut [Gay94] gibt es eine Einbettung von $\mathcal{ASP}roc$ in $\mathcal{SP}roc$. Die Idee der Einbettung besteht darin, eine zusätzliche Aktion wait einzuführen und in den Sicherheitsspezifikationen an jeder möglichen Stelle beliebige Folgen von wait-Aktionen einzufügen.

Wenn in dieser Arbeit asynchrone Prozeßsysteme in $\mathcal{SP}roc$ modelliert werden (wie zum Beispiel in Abschnitt 3.2) soll das allerdings auf etwas einfacherem Wege erfolgen. Die zusätzliche Aktion wait wird nur an den notwendigen Stellen in den Sicherheitsspezifikationen zugelassen. Dieses Verfahren ist zwar nicht universell verwendbar, hat aber einen Vorteil: durch eine entsprechende Anzahl von wait-Aktionen können die durch Komposition entstehenden Prozesse nicht divergieren. Überlegungen, wie in [Gay94] bezüglich Fairness und Divergenz geführt, können hier entfallen.

Die aus dieser Behandlung von asynchronem Verhalten resultierenden technischen Probleme und Unterschiede in der Modellierung der Beispiele sind auf die Anlage der vorliegenden Arbeit und die zur Zeit noch unbefriedigende Situation bei asynchronen Interaktionskategorien zurückzuführen und sollen deshalb bei der Bewertung beider Konzepte unberücksichtigt bleiben.

Die folgende kurze Vorstellung von $\mathcal{SP}roc$ ist im wesentlichen [GN94] entnommen. Der interessierte Leser wird für eine ausführliche Darstellung der zugrundeliegenden Ideen allerdings auf [AGN93] verwiesen.

$\mathcal{SP}roc$ ist eine bereicherte Kategorie, deren Objekte und Morphismen als Typspezifikationen von Prozessen verstanden werden. In Anlehnung an den Curry-Howard-Isomorphismus ist $\mathcal{SP}roc$ auch ein Modell für klassische Lineare Logik. Im folgenden wird unter einem Prozeß eine Äquivalenzklasse von nach starker Bisimulation faktorisierten Synchronisationsbäumen verstanden. Die Begiffe Prozeß und Synchronisationsbaum werden synonym verwendet.

Ein Objekt der Kategorie $\mathcal{SP}roc$ ist ein Paar $A = (\Sigma_A, S_A)$, wobei Σ_A ein Alphabet und S_A eine nichtleere, präfixabgeschlossene Teilmenge aller endlichen Worte über dem Alphabet Σ_A ist. S_A wird als Sicherheitsspezifikation verstanden. Ein Prozeß p hat den Typ A, geschrieben $p : A$, wenn jeder Pfad des Synchronisationsbaumes von p in S_A enthalten ist: $\mathsf{traces}(p) \subseteq S_A$.

Es ist üblich, die Morphismen von $\mathcal{SP}roc$ mit Hilfe des internen Hom-Funktors zu definieren. Deshalb hier zunächst die Definition des Tensors $\otimes$: Für gegebene A und B ist das Objekt $A \otimes B$ das Paar $(\Sigma_{A\otimes B}, S_{A\otimes B})$ mit

$$\begin{aligned} \Sigma_{A\otimes B} &= \Sigma_A \times \Sigma_B \\ S_{A\otimes B} &= \{\sigma \in \Sigma^*_{A\otimes B} \mid \mathsf{fst}^*(\sigma) \in S_A \wedge \mathsf{snd}^*(\sigma) \in S_B\}. \end{aligned}$$

Die Dualität für Objekte wird trivial definiert: $A^\perp \stackrel{\text{def}}{=} A$. Mit der allgemeinen Definition des Bifunktors $⅋$ folgt $A ⅋ B = A \multimap B = A \otimes B$.

Ein Morphismus $p : A \longrightarrow B$ in $\mathcal{SP}roc$ ist ein Prozeß mit dem Typ $A \multimap B$. Die Komposition zweier Morphismen wird mit Hilfe eines Ableitungsbaumes definiert. Wenn $p : A \multimap B$ und $q : B \multimap C$ dann ist $p\,;\,q : A \multimap C$ definiert mit

$$\frac{p \xrightarrow{(a,b)} p' \qquad q \xrightarrow{(b,c)} q'}{p\,;\,q \xrightarrow{(a,c)} p'\,;\,q'} \tag{2.13}$$

Um die Identität definieren zu können, wird die Einschränkung eines Prozesses auf eine Sicherheitsspezifikation benötigt:

$$\frac{P \xrightarrow{a} Q \qquad a \in S}{P \restriction S \xrightarrow{a} Q \restriction (S/a)} \tag{2.14}$$

wobei $S/a \stackrel{\text{def}}{=} \{\epsilon\} \cup \{\sigma \mid a\sigma \in S\}$. Wenn der Prozeß id durch die folgende Regel definiert wird,

$$\frac{a \in \Sigma_A}{\mathsf{id} \xrightarrow{(a,a)} \mathsf{id}} \tag{2.15}$$

dann ist

$$\mathsf{id}_A \stackrel{\text{def}}{=} \mathsf{id} \restriction S_{A\multimap A} \tag{2.16}$$

Der Beweis, daß die eben definierte Struktur die Axiome einer Kategorie erfüllt, ist in [Gay94] zu finden.

Damit $\mathcal{SP}roc$ den Axiomen einer Linearen Kategorie genügt, müssen die Definitionen für $\otimes$ und $(-)^{\perp}$ zu Funktoren geliftet werden. Wenn $p : A \longrightarrow C$ und $q : B \longrightarrow D$ dann wird $p \otimes q : A \otimes B \longrightarrow C \otimes D$ und $p^{\perp} : C^{\perp} \longrightarrow A^{\perp}$ mit Hilfe der folgenden Transitionsregeln definiert.

$$\frac{p \xrightarrow{(a,c)} p' \qquad q \xrightarrow{(b,d)} q'}{p \otimes q \xrightarrow{((a,b),(c,d))} p' \otimes q'} \qquad\qquad \frac{p \xrightarrow{(a,c)} p'}{p^{\perp} \xrightarrow{(c,a)} p'^{\perp}}$$

Das neutrale Element I des Tensors $\otimes$ ist das Paar (Σ_I, S_I) mit

$$\begin{aligned} \Sigma_I &\stackrel{\text{def}}{=} * \\ S_I &\stackrel{\text{def}}{=} \{*^n \mid n \in \mathbb{N}\} \end{aligned}$$

Die rechte Adjunktion zum Tensor kann mit der folgenden eineindeutigen Beziehung zwischen den Morphismen $p : A \otimes B \longrightarrow C$ und $\Lambda(p) : A \longrightarrow B \multimap C$ konstruiert werden.

$$\frac{p \xrightarrow{((a,b),c)} q}{\Lambda(p) \xrightarrow{(a,(b,c))} \Lambda(q)}$$

Die Definition der anderen Morphismen, zum Beispiel symm oder unitl, kann auf ähnliche Weise mittels Umgruppieren von Tupeln oder Vertauschen von Paaren erfolgen.

Für zwei Objekte A und B aus $\mathcal{SP}roc$ ist das Coprodukt das Paar $(\Sigma_{A\oplus B}, S_{A\oplus B})$, wobei

$$\begin{aligned} \Sigma_{A\oplus B} &\stackrel{\text{def}}{=} \Sigma_A \uplus \Sigma_B \\ S_{A\oplus B} &\stackrel{\text{def}}{=} \{\mathsf{inl}^*(s) \mid s \in S_A\} \uplus \{\mathsf{inr}^*(s) \mid s \in S_B\}. \end{aligned}$$

Die bis auf Eindeutigkeit bestimmten Morphismen $[\_,\_]$ lassen sich leicht durch Umbenennung von Transitionen erhalten. Für die Eindeutigkeit der konstruierten Morphismen spielen die Sicherheitsspezifikationen eine wesentliche Rolle. Ein ausführliche Erläuterung mit Gegenbeispielen ist in [Gay94] enthalten.

Zur Definition des Funktors ! werden zunächst einige Bezeichnungen benötigt. Multimengen sollen mit den Symbolen $\{\!|$ und $|\!\}$ eingeschlossen werden. $\mathcal{M}_n(X)$ bezeichne die Multimengen der Kardinalität n mit Elementen aus X. Die symmetrische Potenz des Tensors, $\otimes_s^n$, wird für Objekte durch

$$\begin{aligned} \Sigma_{\otimes_s^n A} &\stackrel{\text{def}}{=} \mathcal{M}_n(\Sigma_A) \\ S_{\otimes_s^n A} &\stackrel{\text{def}}{=} \{\phi_n^*(s) \mid s \in S_{\otimes^n A}\} \end{aligned}$$

definiert. Die Abbildung $\phi_n : \Sigma_A^n \longrightarrow \mathcal{M}_n(\Sigma_A)$ bildet n-Tupel in Multimengen mit n Elementen ab: $\phi_n(l_1, \ldots, l_n) = \{\!| \, l_1, \ldots, l_n \, |\!\}$. Die Wirkung auf Morphismen soll auch hier durch eine Transitionsregel beschrieben werden.

$$\frac{\otimes^n p \xrightarrow{(l_1,\ldots,l_n)} \otimes^n q}{\otimes_s^n p \xrightarrow{\{\!| l_1,\ldots,l_n |\!\}} \otimes_s^n q}$$

Mit diesen Hilfsmitteln kann ! definiert werden:

$$!A \stackrel{\text{def}}{=} \bigoplus_{n \in \mathbb{N}} \otimes_s^n A$$

In [AGN93] wird ausführlich dargestellt, daß diese Definition die gewünschten Eigenschaften besitzt.

Aufgrund der Definition der Involution $(-)^\perp$ für $\mathcal{SP}roc$ ist die Struktur von $\mathcal{SP}roc$ etwas degeneriert. So gilt $A \otimes B = A \multimap B = A \parr B$ und $A \oplus B = A \,\&\, B$ sowie $!A = ?A$.

3. Lebendigkeits- und Sicherheitseigenschaften

Anhand eines einfachen Beispiels, das sowohl in $\mathcal{SP}roc$, als auch im π-Kalkül modelliert wird, soll überprüft werden, inwieweit sich gewünschte Lebendigkeits- und Sicherheitseigenschaften spezifizieren und auch nachweisen lassen.

Zum Begriff

Stirling unterscheidet in [Sti93] zwischen Sicherheits- und Lebendigkeitseigenschaften. Eine Sicherheitseigenschaft drückt aus, daß ein (ungewünschtes) Ereignis in *keinem* Pfad eines Prozesses auftritt. Für nichtterminierende Prozesse muß die Eigenschaft für einen unendlichen Synchronisationsbaum formuliert werden. Sicherheitseigenschaften werden aus diesen Gründen im μ-Kalkül mittels des größten Fixpunktes formuliert. Typische Beispiele sind die Abwesenheit von Verklemmungen (Deadlock) oder Termination.

Im Unterschied dazu drückt eine Lebendigkeitseigenschaft aus, daß ein (gewünschtes) Ereignis mindestens einmal in *jedem* Pfad des Prozesses eintritt. Im μ-Kalkül werden Lebendigkeitseigenschaften mit Hilfe des kleinsten Fixpunktes formuliert. Eine typische Lebendigkeitseigenschaft ist die Existenz von Normalformen in bestimmten Ableitungssystemen. Die gewünschte Eigenschaft ist hier, daß die Ableitung nach endlich vielen Schritten mit Erreichen der Normalform endet.

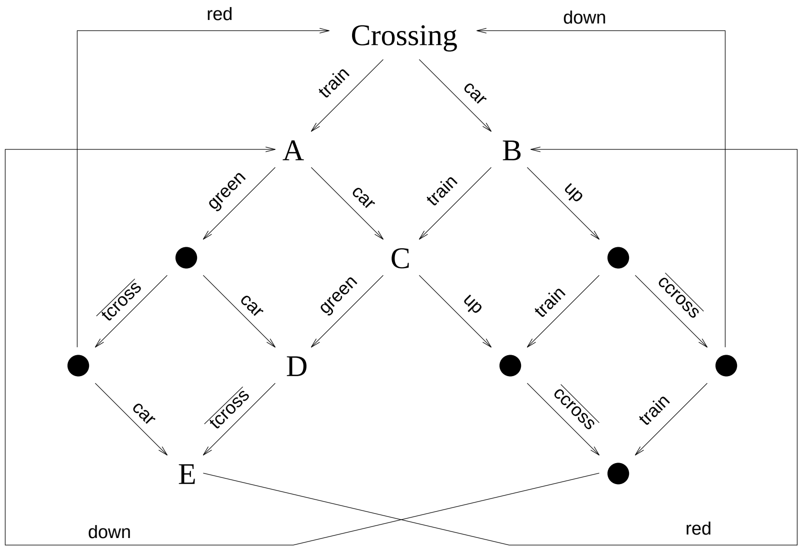
3.1 Modell eines beschrankten Bahnüberganges

Ein beschrankter Bahnübergang kann im π-Kalkül folgendermaßen modelliert werden[3]:

$$\begin{aligned}
\mathsf{Rail} &\stackrel{\text{def}}{=} \mathsf{train}\,.\,\mathsf{green}\,.\,\overline{\mathsf{tcross}}\,.\,\overline{\mathsf{red}}\,.\,\mathsf{Rail} \\
\mathsf{Road} &\stackrel{\text{def}}{=} \mathsf{car}\,.\,\mathsf{up}\,.\,\overline{\mathsf{ccross}}\,.\,\overline{\mathsf{down}}\,.\,\mathsf{Road} \\
\mathsf{Signal} &\stackrel{\text{def}}{=} \overline{\mathsf{green}}\,.\,\mathsf{red}\,.\,\mathsf{Signal} + \overline{\mathsf{up}}\,.\,\mathsf{down}\,.\,\mathsf{Signal} \\
\mathsf{Crossing} &\stackrel{\text{def}}{=} (\mathsf{green})(\mathsf{red})(\mathsf{up})(\mathsf{down})\,(\mathsf{Road} \mid \mathsf{Rail} \mid \mathsf{Signal})
\end{aligned}$$

Das Transitionssystem ist in Abbildung 1 zu sehen. Den meisten existierenden Bahnübergängen unähnlich, öffnet dieser erst die Schranken, wenn sich wirklich ein Auto nähert.

[3] Das Beispiel ist [Sti93] entnommen, wurde aber der Syntax des π-Kalkül angepaßt.



Die verwendeten Konstanten bedeuten:

$$
\begin{aligned}
A &\stackrel{\text{def}}{=} (\mathsf{green})(\mathsf{red})(\mathsf{up})(\mathsf{down})(\mathsf{Road} \mid \mathsf{green} \,.\, \overline{\mathsf{tcross}} \,.\, \overline{\mathsf{red}} \,.\, \mathsf{Rail} \mid \mathsf{Signal})\\
B &\stackrel{\text{def}}{=} (\mathsf{green})(\mathsf{red})(\mathsf{up})(\mathsf{down})(\mathsf{up} \,.\, \overline{\mathsf{ccross}} \,.\, \overline{\mathsf{down}} \,.\, \mathsf{Road} \mid \mathsf{Rail} \mid \mathsf{Signal})\\
C &\stackrel{\text{def}}{=} (\mathsf{green})(\mathsf{red})(\mathsf{up})(\mathsf{down})(\mathsf{up} \,.\, \overline{\mathsf{ccross}} \,.\, \overline{\mathsf{down}} \,.\, \mathsf{Road} \mid\\
&\qquad\qquad \mathsf{green} \,.\, \overline{\mathsf{tcross}} \,.\, \overline{\mathsf{red}} \,.\, \mathsf{Rail} \mid \mathsf{Signal})\\
D &\stackrel{\text{def}}{=} (\mathsf{green})(\mathsf{red})(\mathsf{up})(\mathsf{down})(\mathsf{up} \,.\, \overline{\mathsf{ccross}} \,.\, \overline{\mathsf{down}} \,.\, \mathsf{Road} \mid\\
&\qquad\qquad \overline{\mathsf{tcross}} \,.\, \overline{\mathsf{red}} \,.\, \mathsf{Rail} \mid \mathsf{red} \,.\, \mathsf{Signal})\\
E &\stackrel{\text{def}}{=} (\mathsf{green})(\mathsf{red})(\mathsf{up})(\mathsf{down})(\mathsf{up} \,.\, \overline{\mathsf{ccross}} \,.\, \overline{\mathsf{down}} \,.\, \mathsf{Road} \mid \overline{\mathsf{red}} \,.\, \mathsf{Rail} \mid \mathsf{red} \,.\, \mathsf{Signal})
\end{aligned}
$$

Die mit • bezeichneten Zustände spielen im Text keine weitere Rolle.

Weiterhin ist zu beachten, daß die mit red, green, up und down bewerteten Kanten in Wirklichkeit τ-Transitionen sind, die, wie zum Beispiel zwischen den Knoten **C** und **D**, aus einem green - $\overline{\mathsf{green}}$ Paar entstanden sind.

Abbildung 1: Transitionssystem für crossing

Unfallfreiheit

Eine leicht zu formulierdende Sicherheitseigenschaft für den beschrankten Bahnübergang ist die Unmöglichkeit von Zusammenstößen zwischen Zügen und Autos. Das heißt, im unendlichen Synchronisationsbaum des beschrankten Bahnüberganges soll es keinen Zustand geben, von dem aus als nächste Transition sowohl ein $\overline{\mathsf{tcross}}$ als auch ein $\overline{\mathsf{ccross}}$ möglich ist. Das läßt sich im μ-Kalkül formalisieren:

$$\mathsf{NoCrash} \stackrel{\text{def}}{=} \nu Z\,.\,([-]Z \wedge \neg(\langle\mathsf{tcross}\rangle\mathsf{tt} \wedge \langle\mathsf{ccross}\rangle\mathsf{tt})) \tag{3.1}$$

Da das Transitionssystem von Crossing nur endlich viele Zustände hat, ließe sich (3.1) prinzipiell mit einem Tableaubeweis zeigen. Wegen der Fixpunktkonstruktion würde allerdings jeder Zustand des Transitionssystems im Beweis vorkommen. Der Einfachheit halber soll der Beweis $\mathsf{Crossing} \vdash \mathsf{NoCrash}$ hier mittels Approximation geführt werden.

Satz 1 *Das Modell Crossing erfüllt die Eigenschaft NoCrash: $\mathsf{Crossing} \models \mathsf{NoCrash}$.*

Beweis Mittels Approximation der Formel $\nu Z\,.\,[-]Z \wedge \neg(\langle\mathsf{tcross}\rangle\mathsf{tt} \wedge \langle\mathsf{ccross}\rangle\mathsf{tt})$ bezüglich des Transitionssystems Crossing aus Abbildung 1:

$$\begin{aligned}
\nu^0 Z &= \mathsf{tt} \\
\nu^1 Z &= [-]\mathsf{tt} \wedge \neg(\langle\mathsf{tcross}\rangle\mathsf{tt} \wedge \langle\mathsf{ccross}\rangle\mathsf{tt}) \\
&= [\mathsf{tcross}]\mathsf{ff} \vee [\mathsf{ccross}]\mathsf{ff} \\
&= \mathsf{tt}
\end{aligned}$$

Der größte Fixpunkt umfaßt alle Zustände des Transitionssystems und damit folgt $\mathsf{Crossing} \models \mathsf{NoCrash}$ □

Wartezeiten

Eine andere von einem Bahnübergang erhoffte Eigenschaft ist, daß jedem sich nähernden Fahrzeug schließlich die Durchfahrt gewährt wird. Das ist eine typische Lebendigkeitseigenschaft. Auf das formale Modell Crossing übertragen, ergibt sich folgende Formulierung: jeder Transition car oder train in jedem möglichen Pfad im Synchronisationsbaum muß eine entsprechende Transition $\overline{\mathsf{ccross}}$ bzw. $\overline{\mathsf{tcross}}$ folgen. Der Einfachheit halber wird diese Eigenschaft hier nur für Autos im μ-Kalkül definiert.

$$\mathsf{CarWaiting} \stackrel{\text{def}}{=} \nu Z\,.\,[-]Z \wedge [\mathsf{car}]\,(\mu Y\,.\,\langle-\rangle tt \wedge [-\overline{\mathsf{ccross}}]Y) \tag{3.2}$$

Wie man sich leicht überlegen kann, gilt $\mathsf{Crossing} \models \mathsf{CarWaiting}$ *nicht*. Nach der Transition car gibt es einen unendlich langen Pfad $\mathsf{train}, \tau, \overline{\mathsf{tcross}}, \tau, \mathsf{train}, \tau, \overline{\mathsf{tcross}}, \tau, \ldots$. Dementspechend schlägt der Tableaubeweis $\mathsf{Crossing} \vdash \mathsf{CarWaiting}$ in Abbildung 2 fehl.

Satz 2 *Es gilt: $\mathsf{Crossing} \not\models \mathsf{CarWaiting}$*

$$
\begin{prooftree}
\AxiomC{$\vdots$}
\noLine
\UnaryInfC{$\mathsf{Crossing} \vdash [-]\Theta$}
\AxiomC{$\mathbf{C} \vdash \mathsf{tt}$}
\RightLabel{$\langle - \rangle$}
\UnaryInfC{$\mathbf{B} \vdash \langle - \rangle \mathsf{tt}$}
\AxiomC{$\vdots$}
\noLine
\UnaryInfC{$\bullet \vdash \Psi$}
\AxiomC{$\mathbf{D} \vdash \mathsf{tt}$}
\RightLabel{$\langle - \rangle$}
\UnaryInfC{$\mathbf{C} \vdash \langle - \rangle \mathsf{tt}$}
\AxiomC{$\underline{\mathbf{B} \vdash \Psi}$}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\underline{\mathbf{E} \vdash \Psi}$}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\underline{\mathbf{D} \vdash \Psi}$}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\mathbf{C} \vdash [-\overline{\mathsf{ccross}}]\Psi$}
\RightLabel{$\wedge$}
\BinaryInfC{$\mathbf{C} \vdash \langle - \rangle \mathsf{tt} \wedge [-\overline{\mathsf{ccross}}]\Psi$}
\RightLabel{Ψ}
\UnaryInfC{$\mathbf{C} \vdash \Psi$}
\RightLabel{$[-\overline{\mathsf{ccross}}]$}
\BinaryInfC{$\mathbf{B} \vdash [-\overline{\mathsf{ccross}}]\Psi$}
\RightLabel{$\wedge$}
\BinaryInfC{$\mathbf{B} \vdash \langle - \rangle \mathsf{tt} \wedge [-\overline{\mathsf{ccross}}]\Psi$}
\RightLabel{Ψ}
\UnaryInfC{$\mathbf{B} \vdash \Psi$}
\RightLabel{μY}
\UnaryInfC{$\mathbf{B} \vdash \mu Y\,.\,\Phi$}
\RightLabel{$[\mathsf{car}]$}
\UnaryInfC{$\mathsf{Crossing} \vdash [\mathsf{car}](\mu Y\,.\,\Phi)$}
\RightLabel{$\wedge$}
\BinaryInfC{$\mathsf{Crossing} \vdash [-]\Theta \wedge [\mathsf{car}](\mu Y\,.\,\Phi)$}
\RightLabel{Θ}
\UnaryInfC{$\mathsf{Crossing} \vdash \Theta$}
\RightLabel{νZ}
\UnaryInfC{$\mathsf{Crossing} \vdash \nu Z\,.\,[-]Z \wedge [\mathsf{car}](\mu Y\,.\,\Phi)$}
\RightLabel{$\mathsf{CarWaiting}$}
\UnaryInfC{$\mathsf{Crossing} \vdash \mathsf{CarWaiting}$}
\end{prooftree}
$$

Die verwendeten Konstanten bedeuten:

$$
\begin{aligned}
\Phi &\stackrel{\text{def}}{=} \langle - \rangle \mathsf{tt} \wedge [-\overline{\mathsf{ccross}}]Y \\
\Theta &\stackrel{\text{def}}{=} \nu Z\,.\,[-]Z \wedge [\mathsf{car}](\mu Y\,.\,\Phi) \\
\Psi &\stackrel{\text{def}}{=} \mu Y\,.\,\Phi
\end{aligned}
$$

Die Bezeichner **A, B, C, D** und **E** für Prozesse beziehen sich auf Abbildung 1.

Das Blatt $\mathbf{B} \vdash \Psi$ ist erfolglos, da Ψ einen kleinsten Fixpunkt bezeichnet. Da im Pfad zwischen der Wurzel $\mathsf{Crossing} \vdash \mathsf{CarWaiting}$ und diesem Blatt keine Alternativen möglich sind, ist der Pfad, der zum Blatt $\mathbf{B} \vdash \Psi$ führt, in jedem Beweisbaum für die Formel $\mathsf{Crossing} \vdash \mathsf{CarWaiting}$ enhalten und somit existiert für diese Formel kein Beweisbaum.

Abbildung 2: Fehlschlagender Tableaubeweis für $\mathsf{Crossing} \vdash \mathsf{CarWaiting}$

Beweis Der Beweis wird in Abbildung 2 mittels des Tableausystems aus [Rei95] geführt. □

3.2 Der beschrankte Bahnübergang in $\mathcal{SProc}$

Die Straße sowie die Eisenbahn sollen als Prozesse mit zwei Kanälen modelliert werden. Jeweils ein Kanal steht für die Kommunikation mit der Umwelt (Signale train, tcross, car, ccross) zur Verfügung, während der andere zur Synchronisation mit der Signalanlage benutzt wird (Signale green, red, up, down).

Die Signalanlage soll auch zwei Kanäle haben, jeweils einen für Ampel und Schranke. Mit $\Sigma_{\mathsf{Lights}} \stackrel{\text{def}}{=} \{\mathsf{red},\ \mathsf{green},\ \mathsf{wait}\}$ und $\Sigma_{\mathsf{Barrier}} \stackrel{\text{def}}{=} \{\mathsf{up},\ \mathsf{down},\ \mathsf{wait}\}$ und geeigneten Sicherheitsspezifikationen S_{Lights} und S_{Barrier}[4] könnte ein die Signalanlage realisierender Prozeß s getypt werden: $s : \mathsf{Lights}^{\perp} \parr \mathsf{Barrier}^{\perp}$. Um die Darstellung in Grenzen zu halten, wird solch ein Prozeß nicht konstruiert. Dagegen wird ein Typ Signal mit $\Sigma_{\mathsf{Signal}} = \Sigma_{\mathsf{Lights}^{\perp} \parr \mathsf{Barrier}^{\perp}}$ mit einer nicht trivialen Sicherheitsspezifikation angegeben. Damit ist Signal ein Teiltyp von $\mathsf{Lights}^{\perp} \parr \mathsf{Barrier}^{\perp}$ in dem Sinne, daß für jeden Prozeß p mit $p : \mathsf{Signal}$ auch $p : \mathsf{Lights}^{\perp} \parr \mathsf{Barrier}^{\perp}$ gilt.

Die eben am Beispiel der Signalanlage dargestellten Überlegungen sollen sinngemäß auf die anderen beiden am System beteiligten Prozesse Road und Rail übertragen werden. Damit ergeben sich die folgenden Definitionen:

$$
\begin{array}{lcl}
\Sigma_{\mathsf{Rail}} & \stackrel{\text{def}}{=} & \{\mathsf{train}, \mathsf{tcross}, \mathsf{wait}\} \times \{\mathsf{green}, \mathsf{red}, \mathsf{wait}\} & = & \Sigma_{\mathsf{Train}^{\perp} \parr \mathsf{Lights}} \\
\Sigma_{\mathsf{Signal}} & \stackrel{\text{def}}{=} & \{\mathsf{green}, \mathsf{red}, \mathsf{wait}\} \times \{\mathsf{up}, \mathsf{down}, \mathsf{wait}\} & = & \Sigma_{\mathsf{Lights}^{\perp} \parr \mathsf{Barrier}^{\perp}} \\
\Sigma_{\mathsf{Road}} & \stackrel{\text{def}}{=} & \{\mathsf{up}, \mathsf{down}, \mathsf{wait}\} \times \{\mathsf{car}, \mathsf{tcross}, \mathsf{wait}\} & = & \Sigma_{\mathsf{Barrier} \parr \mathsf{Car}}
\end{array}
$$

Alle am Modell beteiligten Prozesse sollen sequentiell in dem Sinne sein, daß immer nur an einem ihrer Kanäle eine von wait unterschiedliche Aktion auftreten kann. Deshalb wird von nun an abkürzend green, up und so weiter für die Paare (green, wait), (wait, up), ... geschrieben werden. Das Paar (wait,wait) wird mit wait abgekürzt.

Die Sicherheitsspezifikationen werden mit Hilfe regulärer Ausdrücke angegeben. Wie in Abschnitt 2.3 erörtert, sollen nur an den erforderlichen Stellen wait-Aktionen eingefügt werden.

$$
\begin{array}{rcl}
S_{\mathsf{Rail}} & \stackrel{\text{def}}{=} & (\mathsf{wait}^* \ \mathsf{train} \ \mathsf{wait}^* \ \mathsf{green} \ [\mathsf{wait}] \ \mathsf{tcross} \ [\mathsf{wait}] \ \mathsf{red})^* \\
S_{\mathsf{Signal}} & \stackrel{\text{def}}{=} & ([\mathsf{wait}] \ [\mathsf{wait}] \ ((\mathsf{up} \ \mathsf{wait} \ [\mathsf{wait}] \ \mathsf{down}) \mid (\mathsf{green} \ \mathsf{wait} \ [\mathsf{wait}] \ \mathsf{red})))^* \\
S_{\mathsf{Road}} & \stackrel{\text{def}}{=} & (\mathsf{wait}^* \ \mathsf{car} \ \mathsf{wait}^* \ \mathsf{up} \ [\mathsf{wait}] \ \mathsf{ccross} \ [\mathsf{wait}] \ \mathsf{down})^*
\end{array}
\tag{3.3}
$$

Ein Ausschnitt der Synchronisationsbäume von Rail und Signal ist in den Abbildungen 3 und 4 dargestellt. Prozesse des Types Rail und Road können beliebig viele wait-Aktionen hintereinander an einem ihrer Ports ausführen. Prozesse des Types Signal haben diese Eigenschaft nicht. Dadurch gibt es im Synchronisationsbaum der Komposition dreier Prozesse dieses Typs keine beliebig langen Folgen von wait-Aktionen. Überlegungen

[4] Der an technischen Details interessierte Leser stelle sich vor, diese Sicherheitsspezifikationen seien trivial und würden alle unter dem entsprechenden Alphabet zulässigen Pfade enthalten

bezüglich Divergenz, wie in [Gay94] für den allgemeinen Fall geführt, müssen deshalb hier nicht vorgenommen werden.

Im folgenden sollen t, s und c drei anonyme, jeweils den Zug, die Signalanlage und das Auto implementierende Prozesse bezeichnen. Es soll gelten s : Signal, t : Rail und c : Road. Offensichtlich gilt damit auch $t : \mathsf{Train} \multimap \mathsf{Lights}$, $s : \mathsf{Lights} \multimap \mathsf{Barrier}^{\perp}$ und $c : \mathsf{Barrier}^{\perp} \multimap \mathsf{Car}$, und die Komposition der drei Prozesse $\mathsf{crossing} \stackrel{\text{def}}{=} t\,;\, s\,;\, c$ hat den Typ $\mathsf{Train} \multimap \mathsf{Car}$. Für diese Komposition gelten die obigen Bemerkungen über Sequentialität nicht mehr. Zum Beispiel kann die Transition $\mathsf{crossing} \xrightarrow{(\mathsf{train},\mathsf{car})} \mathsf{crossing}'$ auftreten, wenn $t \xrightarrow{(\mathsf{train},\mathsf{wait})} t'$, $s \xrightarrow{(\mathsf{wait},\mathsf{wait})} s'$ und $c \xrightarrow{(\mathsf{wait},\mathsf{car})} c'$ stattfinden und $\mathsf{crossing}' = t'\,;\, s'\,;\, c'$ ist.

Spezifikation von Eigenschaften in $\mathcal{SP}roc$

Wie schon im Abschnitt 3.1 soll nun versucht werden, einige kritische Eigenschaften nachzuweisen. Im Gegensatz zum π-Kalkül, wo drei konkrete Prozeßausdrücke zur Verfügung standen, sollen die Eigenschaften jetzt für eine ganze Menge von Prozessen, nämlich für alle Prozesse eines bestimmten Typs, gezeigt werden.

Abramsky schlägt in [AGN93] zum Nachweis von Eigenschaften P in Prozeßkategorien folgende Verfahrensweise vor: Zuerst wird eine Einschränkung $\mathsf{SafeCrossing}$ des interessierenden Typs $\mathsf{Train} \multimap \mathsf{Car}$ konstruiert, so daß

$$\begin{aligned} \Sigma_{\mathsf{SafeCrossing}} &= \Sigma_{\mathsf{Train} \multimap \mathsf{Car}} \\ S_{\mathsf{SafeCrossing}} &\subseteq S_{\mathsf{Train} \multimap \mathsf{Car}} \end{aligned}$$

ist. Die Sicherheitsspezifikation des eingeschränkten Types soll die gewünschte Eigenschaft respektieren, in dem Sinne, daß gilt:

$$p : \mathsf{SafeCrossing} \quad \Longrightarrow \quad p \text{ hat die Eigenschaft } P$$

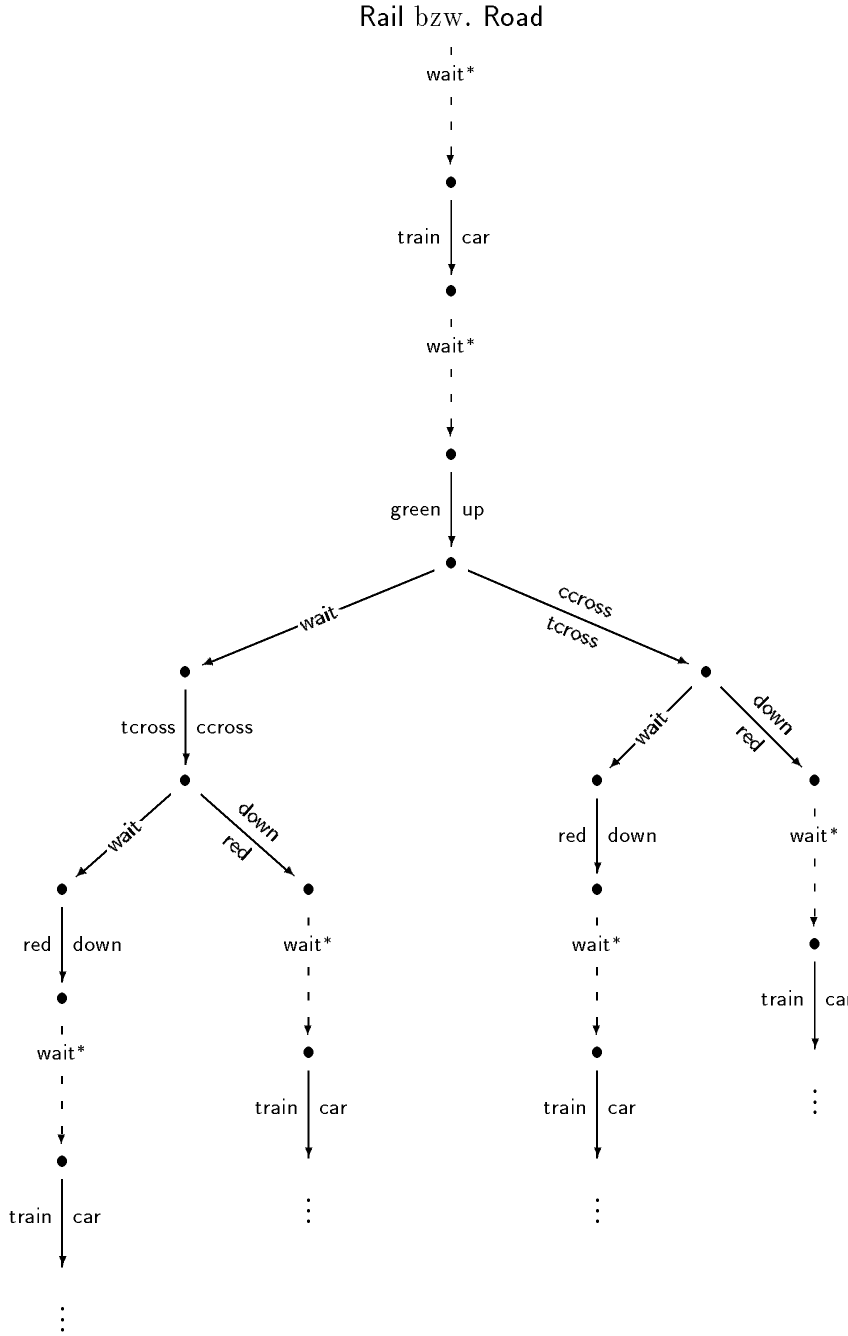
Die Identität $\mathsf{id}_{\mathsf{SafeCrossing}}$ dieses eingeschränkten Types ist gleichzeitig auch ein Morphismus vom Typ $(\mathsf{Train} \multimap \mathsf{Car}) \multimap \mathsf{SafeCrossing}$. Für alle Prozesse p, für die man $p^{\#}\,;\, \mathsf{id}_{\mathsf{SafeCrossing}} = p^{\#}$[5] nachweisen kann, gilt offensichtlich die Eigenschaft P.

Unfallfreiheit

Dieser Idee folgend, muß zuerst die Eigenschaft $\mathsf{NoCrash}$, die in Gleichung (3.1) im μ-Kalkül formalisiert wurde, auf Synchronisationsbäume übertragen werden: Ein Bahnübergang $b : \mathsf{Train} \multimap \mathsf{Car}$ wird als *sicher* bezeichnet, wenn der Synchronisationsbaum von b keine Transaktion $(\mathsf{tcross}, \mathsf{ccross})$ enthält.

Satz 3 *Wenn t : Rail, s : Signal und c : Road dann ist $t\,;\, s\,;\, c$ sicher.*

[5] Wenn $f : A \longrightarrow B$, dann ist $f^{\#} : I \longrightarrow A \multimap B$, siehe Regel (2.7)



Die Synchronisationsbäume von Rail und Road unterscheiden sich nur in einigen Transaktionen. Links beziehungsweise unten stehen die zu Rail gehörenden Transaktionen; rechts beziehungsweise oben, die zu Road gehörenden.

Abbildung 3: Synchronisationsbaum von Rail und Road

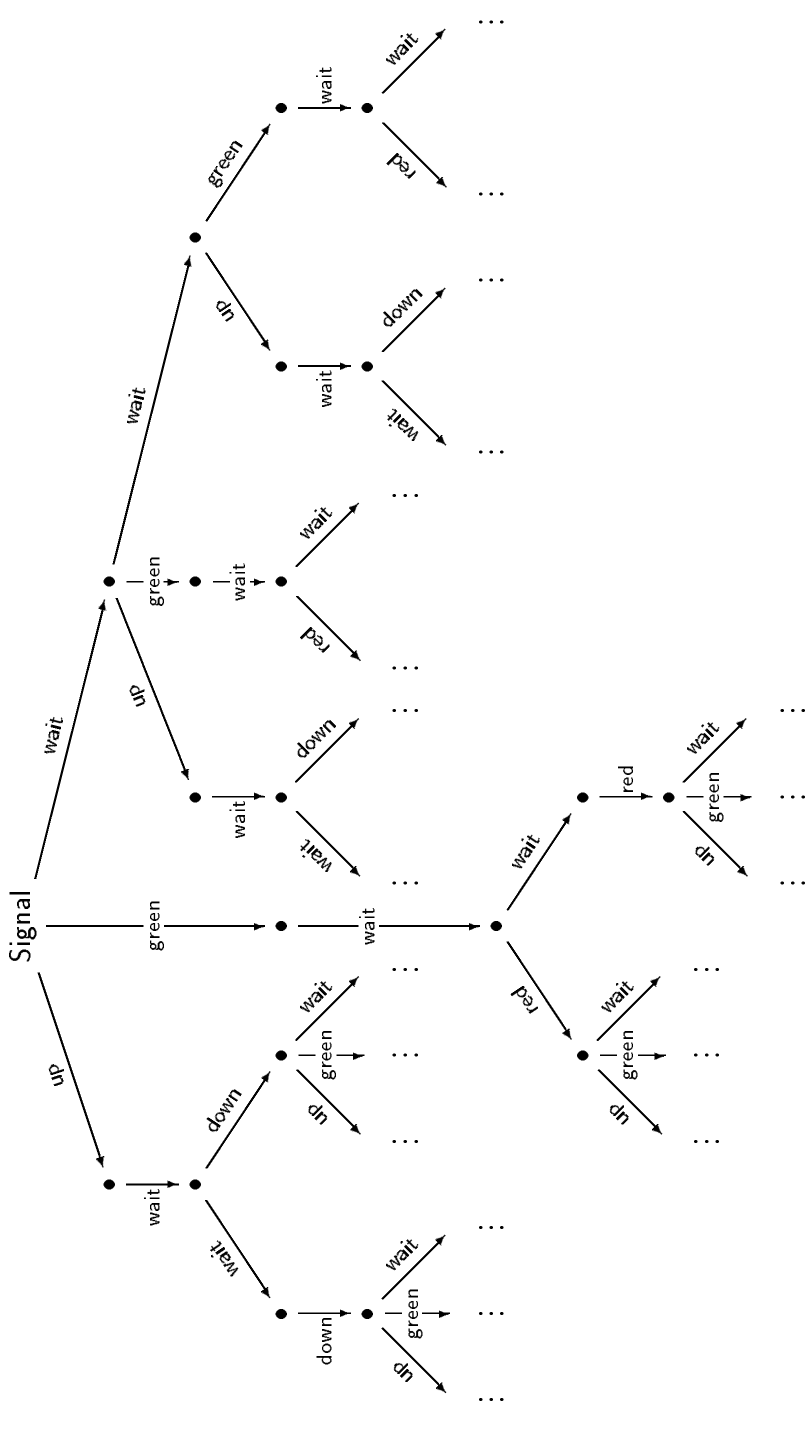


Abbildung 4: Synchronisationsbaum von **Signal**

$$
\begin{array}{llllllll}
t\,;\,s\,;\,c: & ? & \xrightarrow{?} & p & \xrightarrow{(\mathsf{wait},\mathsf{wait})} & q & \xrightarrow{(\mathsf{tcross},\mathsf{ccross})} & r \\[2em]
t: & o_t & \xrightarrow{(\mathsf{wait},\mathsf{wait})} & p_t & \xrightarrow{(\mathsf{wait},\mathsf{green})} & q_t & \xrightarrow{(\mathsf{tcross},\mathsf{wait})} & r_t \\[1em]
s: & o_s & \xrightarrow{(\mathsf{wait},\mathsf{wait})} & p_s & \xrightarrow{(\mathsf{green},\mathsf{wait})} & q_s & \xrightarrow{(\mathsf{wait},\mathsf{wait})} & r_s \\[1em]
c: & ? & \xrightarrow{?} & p_c & \xrightarrow{(\mathsf{wait},\mathsf{wait})} & q_c & \xrightarrow{(\mathsf{wait},\mathsf{ccross})} & r_c
\end{array}
$$

Abbildung 5: Illustration zum Beweis von Lemma 4

Den Vorüberlegungen entsprechend, wird die Sicherheitsspezifikation des Types SafeCrossing als die Menge aller Pfade definiert, die die Transition (tcross, ccross) *nicht* enthalten. Es ist leicht einzusehen, daß diese Menge präfixabgeschlossen ist und daß ein Bahnübergang des Types SafeCrossing sicher ist.

Aus der Definition der Identitäten in $\mathcal{SP}roc$ (2.16) folgt, daß der Prozeß $\mathsf{id}_{\mathsf{SafeCrossing}}$ nur Transitionen mit Markierungen der Form (a, a) besitzt.

Aus Regel (2.13) und (2.7) ist ersichtlich, daß $(t\,;\,s\,;\,c)^{\#}\,;\,\mathsf{id}_{\mathsf{SafeCrossing}} = (t\,;\,s\,;\,c)^{\#}$ gilt, wenn es für jede Transition a im Synchronisationsbaum von $t\,;\,s\,;\,c$ eine Transition (a, a) an entsprechender Stelle im Synchronisationsbaum von $\mathsf{id}_{\mathsf{SafeCrossing}}$ gibt. Damit reduziert sich das Problem darauf, nachzuweisen, daß es im Synchronisationsbaum des Prozesses $t\,;\,s\,;\,c$ keine Transition (tcross, ccross) gibt.

Lemma 4 *Wenn t : Rail, s : Signal und c : Road dann enthält der Synchronisationsbaum des Prozesses $t\,;\,s\,;\,c$ keine Transition (tcross, ccross).*

Beweis von Lemma 4 Der Beweis wird indirekt geführt. Seien t : Rail, s : Signal und c : Road beliebige Prozesse.

Angenommen es gäbe einen Zustand q im Synchronisationsbaum von $t\,;\,s\,;\,c$ mit einer Transition $q \xrightarrow{(\mathsf{tcross},\mathsf{ccross})} r$. Dann müßte es im Synchronisationsbaum von t eine Transition $q_t \xrightarrow{(\mathsf{tcross},\mathsf{wait})} r_t$, im Synchronisationsbaum von s eine Transition $q_s \xrightarrow{(\mathsf{wait},\mathsf{wait})} r_s$ und im Synchronisationsbaum von c eine Transition $q_c \xrightarrow{(\mathsf{wait},\mathsf{ccross})} r_c$ geben. Mit der Sicherheitsspezifikation der Typen Rail, Signal und Road (3.3) folgt, daß die zum Zustand q_s führende Transition entweder (green, wait), (wait, up) oder (wait, wait) gewesen sein müßte.

Im folgenden wird nur auf den ersten Fall eingegangen, die anderen beiden sind ebenso einfach auf einen Widerspruch zu führen. Wie in Abbildung 5 zu sehen ist, gibt es aufgrund der Sicherheitsspezifikation von Road nur eine Möglichkeit: die vor (wait, ccross) ausgeführte Transition ist (wait, wait). Aufgrund der Sicherheitsspezifikation von Road

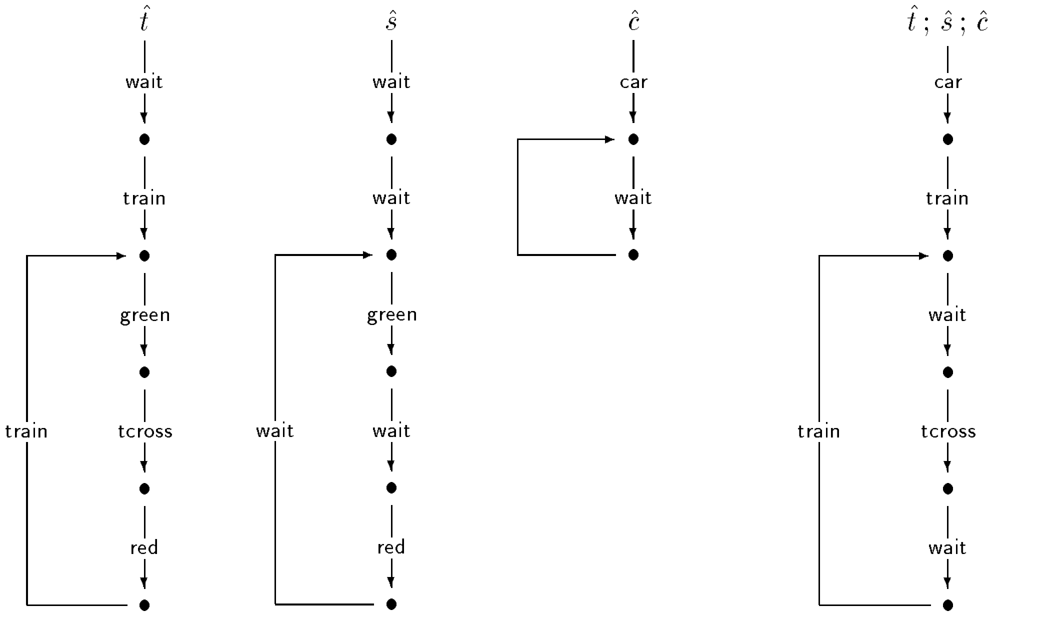


Abbildung 6: Transitionssysteme von $\hat{t}$, $\hat{s}$, $\hat{c}$ und $\hat{t}\,;\,\hat{s}\,;\,\hat{c}$

hätte davor eine Transition up erfolgen müssen, aber die Sicherheitsspezifikation von Signal macht nur eine weitere wait-Transition möglich. Folglich gibt es keinen Pfad, über den der Zustand $q = q_t\,;\,q_s\,;\,q_r$ hätte erreicht werden können. □

Beweis von Satz 3 Folgt direkt aus Lemma 4. □

Wartezeiten

Ein Bahnübergang soll als *fair* bezeichnet werden, wenn nach jeder Transition car jeder Pfad mindestens eine Transition ccross enthält.

Satz 5 *Es gibt Prozesse t : Rail, s : Signal und c : Road, so daß $t\,;\,s\,;\,c$ nicht fair ist.*

Beweis Zum Beweis genügt die Konstruktion eines Gegenbeispiels. Seien $\hat{t}$: train, $\hat{s}$: Signal und $\hat{c}$: car wie in Abbildung 6 definiert. Es ist leicht zu sehen, daß der Synchronisationsbaum von $\hat{t}\,;\,\hat{s}\,;\,\hat{c}$ nur aus einem unendlich langen Pfad besteht, in dem keine Transition ccross vorkommt. □

3.3 Bewertung

Wie schon in Abschnitt 2.3 erläutert, sollen die, auf die Modellierung ein und desselben Beispiels in einem synchronen und einem asynchronen Kalkül zurückzuführenden

Unterschiede hier unberücksichtigt bleiben.

In beiden Kalkülen erfüllen die Modelle die Lebendigkeitseigenschaft CarWaiting nicht. Im π-Kalkül ist eine Verfeinerung des Modells, so daß CarWaiting erfüllt wäre, zwar komplizierter, aber prinzipiell kein Problem.

Das Modell in $\mathcal{SP}roc$ betreffend, könnte die Sicherheitsspezifikation von Signal auf Fairnessbedingungen respektierende Pfade eingeschränkt werden. Als eine geeignete Bedingung käme in Betracht: Ein Pfad, der unendlich viele Aktionen green enthält, muß auch unendlich viele Aktionen up enthalten. Der Prozeß $\hat{s}$ erfüllt diese Bedingung nicht, und Satz 5 wäre zunächst wieder unbewiesen.

Es gibt aber noch eine zweite Klasse von Gegenbeispielen: Prozesse, die nach einer car-Transition endliche Pfade enthalten. Inwieweit es in $\mathcal{SP}roc$ möglich ist, dieses (unerwünschte) Terminieren vor Beendigung der Aufgabe zu verhindern, wird in Abschnitt 5 untersucht werden.

Es soll nun noch auf die verwendeten Beweistechniken eingegangen werden. Das im Beweis zu Satz 2 verwandte Tableausystem ist für Prozesse mit endlichem Transitionssystem korrekt und vollständig [Rei95]. Es gibt darüber hinaus ein weiteres System, das korrekt und vollständig für beliebige Prozesse ist [BS92]. Die Korrektheit der hier geführten Beweise hängt allerdings in kritischer Weise von der Richtigkeit des in Abbildung 1 dargestellten Transitionssytems ab. Mit einem Werkzeug wie der „Mobility Workbench“ [VM94] ließe sich diese Lücke in der Beweisführung schließen.

Zum Nachweis von Eigenschaften in $\mathcal{SP}roc$ steht (zur Zeit noch) nichts Vergleichbares zur Verfügung. Der von Abramsky vorgeschlagene und hier auch vorgestellte Ansatz erwies sich im konkreten Beispiel als nicht hilfreich. Es war nicht möglich, $(t\,;\,s\,;\,c)^{\#}\,;\,\mathsf{id}_{\mathsf{SafeCrossing}} = (t\,;\,s\,;\,c)^{\#}$ auf kategorientheoretischer Ebene zu zeigen. Letztlich führte kein Weg daran vorbei, für alle interessierenden Prozesse direkt anhand ihres Synchronisationsbaumes zu zeigen, daß die kritische Transition (tcross, ccross) unmöglich ist. Als Ursache wird vermutet, daß sich die Eigenschaft NoCrash im Gegensatz zur Sicherheitsspezifikation des Schedulers in [AGN93] nicht in für die drei Komponenten zu zeigende (Teil-)Eigenschaften zerlegen läßt.

Selbst mit Abramsky's Methode, muß für einen formalen Beweis die Sicherheitsspezifikation des eingeschränkten Types konstruiert werden. Schon beim vergleichsweise einfachen Beweis zu Lemma 4 wurde des Aufwandes wegen nur eine informale Beschreibung der Sicherheitsspezifikation angegeben.

Der μ-Kalkül kann zur Charakterisierung der zulässigen Pfade nur in begrenztem Maße herangezogen werden. Zum einen ist nicht klar, auf welchem Wege die Bedingung der Präfixabgeschlossenheit erfüllt werden kann. Zum anderen sind Synchronisationsbäume potentiell unendlich, so daß auf das komplexere, im allgemeinen Fall nicht mehr entscheidbare Tableausystem zurückgegriffen werden müßte.

Abschließend kann gesagt werden, daß sowohl im π-Kalkül als auch in $\mathcal{SP}roc$ der Bahnübergang und die beiden interessierenden Eigenschaften zufriedenstellend modelliert werden konnten.

4. Mobilität

Ein entscheidender Unterschied zwischen CCS[6] [Mil89] und dem π-Kalkül ist die Möglichkeit, im π-Kalkül sich dynamisch verändernde Prozeßstrukturen auf einfache Art modellieren zu können. Zum Beispiel läßt sich ein Server, der eine vorher nicht bekannte Anzahl von Dienstnutzern[7] bedient, wie folgt modellieren:

$$\mathsf{Server} \stackrel{\text{def}}{=} !\,(\mathsf{request}(c)\,.\,\mathsf{ServerProcess}(c)\,)$$

Dabei ist request ein allen Dienstnutzern bekannter Kanal. Falls vom Nutzer gewünscht, kann aber mit c ein privater Kanal an den, den eigentlichen Dienst implementierenden Prozeß ServerProcess, weitergegeben werden.

In diesem Abschnitt soll Mobilität an einem einfacheren Beispiel untersucht werden: einem Semaphor und einer (prinzipiell beliebigen) Menge kooperierender Prozesse, die den Semaphor benutzen, um zu sichern, daß sich jeweils nur ein Prozeß in einem kritischen Abschnitt befindet. Das Gewicht der Darstellung liegt auf der Modellierung des Aufbaus einer privaten Verbindung über einen öffentlich bekannten Kanal.

4.1 Modell eines Semaphors

Ein einfacher Semaphor ist ein Prozeß mit zwei Zuständen.

$$\begin{aligned} S_{\mathsf{free}} &\stackrel{\text{def}}{=} (\mathsf{release})\,\overline{\mathsf{request}}\;\mathsf{release}\,.\,S_{busy}(\mathsf{release}) \\ S_{\mathsf{busy}}(\mathsf{release}) &\stackrel{\text{def}}{=} \mathsf{release}\,.\,S_{\mathsf{free}} \end{aligned}$$

Die Ausgabe des privaten Kanals release über den öffentlichen Kanal request soll sichern, daß nur der Prozeß, dem der Semaphor zugesprochen wurde, ihn wieder freigeben kann. Die Nutzerprozesse werden durch einfache Zyklen modelliert.

$$\mathsf{Client}_i \stackrel{\text{def}}{=} \mathsf{request}(c)\,.\,\overline{\mathsf{entry}_i}\,.\,\overline{\mathsf{exit}_i}\,.\,\overline{c}\,.\,\mathsf{Client}_i$$

Das Gesamtsystem besteht aus der Parallelkomposition des Semaphors mit beliebig vielen Nutzern.

$$\mathsf{Demo} \stackrel{\text{def}}{=} S_{\mathsf{free}} \mid \mathsf{Client}_1 \mid \cdots \mid \mathsf{Client}_n$$

Zu Demonstrationszwecken wird ein nicht funktionierender Dienstnutzer definiert:

$$\begin{aligned} \mathsf{BrokenCient} &\stackrel{\text{def}}{=} \mathsf{request}(c)\,.\,\overline{\mathsf{entry}_{\mathsf{Broken}}}\,.\,\mathbf{0} \\ \mathsf{Demo}' &\stackrel{\text{def}}{=} S_{\mathsf{free}} \mid \mathsf{Client}_1 \mid \cdots \mid \mathsf{Client}_n \mid \mathsf{BrokenCient} \end{aligned}$$

Im System Demo' kann eine Verklemmung auftreten. Nach der Transition $\overline{\mathsf{entry}_{\mathsf{Broken}}}$ wird der Semaphor nicht mehr in den Zustand S_{free} zurückkehren, kein weiterer Dienstnutzer kann einen kritischen Abschnitt betreten.

[6] Calculus of Communicating Systems

[7] Das Wort Dienstnutzer soll dem Anglizismus „Client“ vorgezogen werden.

$$
\dfrac{\dfrac{\dfrac{}{S_1 \xrightarrow{\overline{\mathsf{request}}\ \mathsf{release}} S_2}\ \textsc{Act}}{S_{\mathsf{free}} \xrightarrow{\overline{\mathsf{request}}\ (\mathsf{release})} S_2}\ \textsc{Open} \qquad \dfrac{}{\mathsf{request}(c)\,.\,\overline{\mathsf{entry}_{\mathsf{Broken}}}\,.\,\mathbf{0} \xrightarrow{\mathsf{request}\ (\mathsf{release})} \overline{\mathsf{entry}_{\mathsf{Broken}}}\,.\,\mathbf{0}}\ \textsc{Act}}{\dfrac{S_{\mathsf{free}} \mid \mathsf{BrokenClient} \xrightarrow{\tau} (\mathsf{release})(S_2 \mid \overline{\mathsf{entry}_{\mathsf{Broken}}}\,.\,\mathbf{0})}{S_{\mathsf{free}} \mid \mathsf{BrokenClient} \xrightarrow{\tau\cdot\overline{\mathsf{entry}_{\mathsf{Broken}}}} (\mathsf{release})(S_2 \mid \mathbf{0}) \approx (\mathsf{release})(\mathsf{release}\,.\,S_{\mathsf{free}})}\ \textsc{Act}}\ \textsc{Close}
$$

Zur besseren Übersicht wurden folgende Konstanten verwendet:

$$
\begin{aligned}
S_1 &\stackrel{\text{def}}{=} \overline{\mathsf{request}}\ \mathsf{release}\,.\,\mathsf{release}\,.\,S_{\mathsf{free}} \\
S_2 &\stackrel{\text{def}}{=} \mathsf{release}\,.\,S_{\mathsf{free}}
\end{aligned}
$$

Die Bezeichnungen der Regeln beziehen sich auf Tabelle 2 in [MPW92b].

Abbildung 7: Beispielhafte Ableitung zum Beweis von Satz 6

Satz 6 *Sei P ein beliebiger Prozeßausdruck, in dem die Aktion $\overline{\mathsf{request}}$ nicht frei in Subjektposition auftritt, t eine beliebige Aktionsfolge und Q ein Prozeß, so daß*

$$
\mathit{Demo}' \xrightarrow{t\,\cdot\,\overline{\mathit{entry}_{\mathit{Broken}}}} Q
$$

gilt. Dann ist $P \mid Q \approx P$.

Beweisskizze Mittels der in [MPW92b] gegebenen Transitionssemantik läßt sich ableiten, daß Q einen Prozeßausdruck der Form $(\mathsf{release})(\mathsf{release}\,.\,S_{\mathsf{free}})$ enthalten muß. Eine Beispielableitung, die sich leicht verallgemeinern läßt, ist in Abbildung 7 angegeben. Für einen Beweis ist nun noch zu zeigen, daß sich dieser Prozeß im Deadlock befindet. Zwar ist eine Transition $\mathsf{release}\,.\,S_{\mathsf{free}} \xrightarrow{\mathsf{release}} S_{\mathsf{free}}$ mittels der Regel Input-Act ableitbar. Aber es kann keine der Regeln Res, Open oder Close angewandt werden, um eine Aktion des Prozesses $(\mathsf{release})(\mathsf{release}\,.\,S_{\mathsf{free}})$ zu erhalten. □

Damit wird nicht nur bewiesen, daß es unmöglich ist, daß ein bestimmtes Protokoll einhaltende Dienstnutzer den Semaphor „unbeabsichtigt“ für einen anderen Prozeß freigeben. Sogar beliebig konstruierte Eindringlinge können das Protokoll nicht stören.

4.2 Der Semaphor in *SProc*

Bei einer Modellierung in *SProc* müssen zunächst die Typen der beteiligten Prozesse so festgelegt werden, daß die gewünschte Kommunikation möglich wird. Die degenerierte Struktur von *SProc* soll hierbei nicht ausgenutzt werden. Nach der Komposition mit dem Semaphor sollen von jedem beteiligten Dienstnutzer i die Kanäle entry_i und exit_i für den äußeren Beobachter sichtbar bleiben. Die interne Kommunikation soll ebenfalls über zwei Kanäle $\mathsf{request}$ und $\mathsf{release}$ erfolgen. Eine Variante, die Typen S für den Semaphor und Cient_i für die Dienstnutzer festzulegen, ist:

$$\begin{aligned} S &\stackrel{\text{def}}{=} (\mathsf{request}^{\perp} \parr \mathsf{release}^{\perp}) \\ \mathsf{Cient}_i &\stackrel{\text{def}}{=} (\mathsf{request} \otimes \mathsf{release}) \parr (\mathsf{entry}_n \parr \mathsf{exit}_n) \end{aligned}$$

Mobilität in $\mathcal{SProc}$

Das entscheidende Problem ist, *einen* Prozeß mit zwei Kanälen mit n Prozessen mit insgesamt $2n$ Kanälen so zu verbinden, daß der Semaphor immer nur mit einem Dienstnutzer auf einmal kommuniziert. Der Autor sieht in $\mathcal{SProc}$ zwei Möglichkeiten, die Konstruktion fortzusetzen:

1. Es wird für jedes n ein Prozeß vom Typ

   $$A \stackrel{\text{def}}{=} S^{\perp} \multimap (\underbrace{S \parr \cdots \parr S}_{n \text{ mal}})$$

   konstruiert, der als „Auffächerer“ dient.

2. Mittels des Funktors ! kann jeder Semaphor mit beliebig vielen Dienstnutzern komponiert werden.

Variante 1 könnte ernsthaft in Betracht gezogen werden, wenn es möglich wäre, den Auffächerungsprozeß so allgemein zu konstruieren, daß er auch in anderen Anwendungen verwendet werden könnte. Eine der Voraussetzungen[8] für eine allgemeine Verwendbarkeit von A ist, daß A im wesentlichen als Verbindungskanal agiert: Jedes Signal auf der linken Seite wird einem Prozeß auf der rechten Seite angeboten. So ist A aber zusammen mit dem Semaphor nicht verwendbar, weil dann ein beliebiger Prozeß den Semaphor freigeben könnte.

Die Sicherheitsspezifikation von A so einzuschränken, daß ein zu Satz 6 äquivalentes Ergebnis bewiesen werden könnte, würde die Wiederverwendbarkeit von A stark einschränken. Deshalb soll hier Variante 2 etwas ausführlicher diskutiert werden.

Konstruktion des Semaphors mittels des Funktors !

Eine Ableitung in Linearer Logik (Abbildung 8) zeigt, daß der Prozeß $!S$ mit beliebig vielen Dienstnutzern komponiert werden kann und daß das Resultat den Typ $(\mathsf{entry}_1 \parr \mathsf{exit}_1) \otimes \cdots \otimes (\mathsf{entry}_n \parr \mathsf{exit}_n)$ hat.

Aber auch bei dieser Variante bleibt ein Problem ungelöst: Der Funktor ! auf $\mathcal{SProc}$ erzeugt eine beliebige Anzahl von Kopien des eigentlichen Semaphors. Das heißt, jeder Dienstnutzer wird mit einem privaten Semaphor verbunden, eine Synchronisation der kritischen Abschnitte findet *nicht* statt.

[8] Diese Argumentation erhebt keinen Anspruch auf Vollständigkeit. Es geht nur darum, zu zeigen, daß ein Prozeß vom Typ A keine Alternative ist.

$$
\begin{array}{c}
\vdash \mathsf{Client}_{n-1} \;\cdots\; \vdash \mathsf{Cient}_1 \\
\vdots \\
\dfrac{\vdash \overbrace{?V,\ldots,?V}^{(n-1)\ \mathrm{mal}}, U_{n-1} \qquad \dfrac{\vdash \mathsf{Cient}_n \stackrel{\mathrm{def}}{=} (\mathsf{request} \otimes \mathsf{release}) \parr (\mathsf{entry}_n \parr \mathsf{exit}_n)}{\vdash ?V, (\mathsf{entry}_n \parr \mathsf{exit}_n)}\ \mathsf{Dereliction}}{\vdash \overbrace{?V,\ldots,?V}^{n\ \mathrm{mal}}, U_n}\ \otimes
\end{array}
$$

$$
\begin{array}{c}
\vdots\ (n-2)\ \mathrm{mal}\ \mathsf{Contraction}
\end{array}
$$

$$
\dfrac{\vdash !S \;=\; !\,(\mathsf{request}^{\perp} \parr \mathsf{release}^{\perp}) \qquad \dfrac{\vdash ?V, ?V, U_n}{\vdash ?V, U_n}\ \mathsf{Contraction}}{U_n}\ \mathsf{Cut}
$$

Aus drucktechnischen Gründen mußten zwei Abkürzungen eingeführt werden:

$$
\begin{aligned}
V &\stackrel{\mathrm{def}}{=} \mathsf{request} \otimes \mathsf{release} \\
U_n &\stackrel{\mathrm{def}}{=} (\mathsf{entry}_1 \parr \mathsf{exit}_1) \otimes \cdots \otimes (\mathsf{entry}_n \parr \mathsf{exit}_n)
\end{aligned}
$$

Abbildung 8: Typableitung mittels Linearer Logik

Als Lösung des Problems könnte wie in Abschnitt 3 eine Einschränkung S' des Types $!S$ konstruiert werden, so daß bei einer Komposition von S' mit n Prozessen des Types Client eine Synchronisation stattfindet und sich nur jeweils ein Prozeß im kritischen Abschnitt befindet. In der Sicherheitsspezifikation von S' dürften keine Pfade enthalten sein, in denen eine zweite Aktion auf einem request-Kanal stattfindet, bevor nicht eine Aktion auf den zum ersten request-Kanal gehörenden release-Kanal erfolgt ist. Eine solche Sicherheitsspezifikation kann jedoch nur konstruiert werden, wenn schon eine Lösung des in diesem Abschnitt diskutierten Problems — der Synchronisation von beliebig vielen Prozessen bezüglich eines kritischen Abschnitts — vorhanden ist.

4.3 Bewertung

Für ein Standardproblem der Informatik konnte innerhalb von $\mathcal{SP}roc$ keine Lösung gefunden werden. Es ist zwar möglich, einen Semaphor mit den gewünschten Eigenschaften zu entwerfen. Aber die wesentliche Aussage eines Modells in $\mathcal{SP}roc$, die Charakterisierung aller möglichen Pfade, ist wertlos, weil sie vorher in das Modell eingebracht werden mußte.

Das verdeutlicht zwei Probleme: Zum einen ist es mit $\mathcal{SP}roc$ aufgrund der statischen Typisierung nur sehr umständlich möglich, dynamische Prozeßstrukturen zu modellieren. Zum anderen kann die Konstruktion der für das Modell notwendigen Sicherheitsspezifikationen eine auf anderen Wegen erbrachte Lösung des zu modellierenden Problems erfordern.

Zur Typableitung in $\mathcal{SP}roc$

Die Typableitung für das Modell in $\mathcal{SP}roc$ wurde ohne Ausnutzung der degenerierten Struktur von $\mathcal{SP}roc$ gemacht. Die Festlegung der Typen für den Semaphor S und die Dienstnutzer erscheint willkürlich. Die Schnittstelle zum äußeren Beobachter könnte zum Beispiel in $(\mathsf{entry}_i^\perp \otimes \mathsf{exit}_i^\perp)$ geändert werden. Allerdings gibt es auch Restriktionen. Zum Beispiel:

- Um mit den Regeln Contraction und Dereliction genau die richtige Anzahl von Kopien der internen Schnittstelle $(\mathsf{request} \otimes \mathsf{release})$ zu erzeugen, muß der Funktor ! auf den Semaphor angewendet werden. Die Alternative ? ist nicht möglich.
- Die beiden Kanäle der internen Schnittstelle müssen im Typ des Semaphors und des Dienstnutzers durch zueinander duale Konnektoren (das heißt auf der einen Seite durch $⅋$ und auf der anderen Seite durch $\otimes$) verbunden werden.

Der letzte Punkt widerspricht der Aussage Abramsky's in [AGN93], daß verschiedene Kanäle eines Prozesses mit $⅋$ verbunden werden sollten. Es ist nicht klar, wie zu verfahren ist, wenn durch Verkettung einer Reihe von Prozessen sich widersprechende Anforderungen an den Typ eines Prozesses ergeben. Zumindest muß dann der Grundsatz aufgegeben werden, von der degenerierten Struktur $\mathcal{SP}roc$'s keinen Gebrauch zu machen. Die Absicht, Interaktionskategorien als nicht degenerierte Modelle der klassischen Linearen Logik zu definieren, erscheint vor diesem Hintergrund fraglich.

Es muß allerdings darauf hingewiesen werden, daß weniger degenerierte Beispiele für Interaktionskategorien existieren, siehe zum Beispiel [Gay94].

5. Erfolgreiche Termination

In verschiedenen Anwendungen ist es wichtig, abzusichern, daß bestimmte Abläufe vollständig ausgeführt werden. Man denke etwa an ein Betriebssystem: Falls es angehalten wird, dann müssen, bevor das Betriebssystem terminiert, modifizierte Puffer gesichert werden. Auf keinen Fall darf es passieren, daß die Modifikation eines im Hauptspeicher liegenden Puffers die letzte Aktion des Betriebssystems ist.

In diesem Abschnitt soll untersucht werden, inwieweit es in $\mathcal{SP}roc$ möglich ist, erfolgreich terminierende Prozesse von erfolglosen zu unterscheiden. Zuerst soll das Problem an einem einfachen Beispiel mit Hilfe des π-Kalküls vorgestellt werden.

5.1 Zwei tick-tack Uhren

Zwei einfache Uhren[9], die abwechselnd tick und tack Geräusche machen können und eventuell stehen bleiben, seien wie folgt definiert:

[9] Die Anregung zu diesem Beispiel erhielt ich durch [Sti93], wo eine große Menge an verschiedenartigen Uhren zu finden ist.

$$\begin{array}{lcl} \mathsf{clock} & \stackrel{\text{def}}{=} & \mathsf{tick} \,.\, \mathsf{clock}' + \mathbf{0} \\ \mathsf{clock}' & \stackrel{\text{def}}{=} & \mathsf{tack} \,.\, \mathsf{clock} + \mathbf{0} \end{array} \qquad \begin{array}{lcl} \mathsf{clock}_e & \stackrel{\text{def}}{=} & \mathsf{tick} \,.\, \mathsf{clock}'_e + \mathbf{0} \\ \mathsf{clock}'_e & \stackrel{\text{def}}{=} & \mathsf{tack} \,.\, \mathsf{clock}_e \end{array}$$

Der wesentliche Unterschied zwischen clock und clock_e besteht darin, daß die Uhr clock_e immer eine gerade Anzahl von Geräuschen von sich gibt, falls sie stehenbleibt. Falls es ein letztes Geräusch gibt, ist es immer ein tack, niemals ein tick. Das läßt sich leicht formalisieren:

$$\begin{array}{rcl} \mathsf{Even} & \stackrel{\text{def}}{=} & \nu Z \,.\, [-][-]Z \wedge [-]\langle - \rangle \mathsf{tt} \\ \mathsf{Lastnoise} & \stackrel{\text{def}}{=} & \mu Z \,.\, [-]Z \vee \langle \mathsf{tack} \rangle [-]\mathsf{ff} \end{array} \tag{5.1}$$

Es gilt

$$\begin{array}{lcl} \mathsf{clock} & \not\models & \mathsf{Even} \\ \mathsf{clock} & \models & \mathsf{Lastnoise} \end{array} \qquad \begin{array}{lcl} \mathsf{clock}_e & \models & \mathsf{Even} \\ \mathsf{clock}_e & \not\models & \mathsf{Lastnoise} \end{array}$$

Der Nachweis mit dem μ-Kalkül ist trivial.

5.2 Die Uhren in $\mathcal{SP}roc$

Unter der Annahme, daß die Uhr clock wegen ihres dissonaten Verhaltens unerwünscht ist, soll untersucht werden, ob es ein Objekt X in $\mathcal{SP}roc$ gibt, so daß clock nicht den Typ X hat, wohingegen $\mathsf{clock}_e : X$ gilt.

Satz 7 *Für alle Objekte X aus $\mathcal{SP}roc$ gilt:*

$$\mathit{clock} : X \iff \mathit{clock}_e : X$$

Beweis Da die Menge der möglichen Aktionen von clock und clock_e identisch ist, bleibt zu zeigen, daß das auch für die kleinstmöglichen Sicherheitsspezifikationen gilt. Da die Sicherheitsspezifikationen präfixabgeschlossen sein müssen, genügt die Beobachtung, daß alle mit tick endenden Pfade von clock den Anfang mindestens eines Pfades von clock_e bilden. □

Anpassung des Modells

Eine Variante, die beiden Uhren in $\mathcal{SP}roc$ zu unterscheiden, besteht darin, erfolgreiches und erfolgloses Terminieren durch extra Aktionen zu markieren. In Abbildung 9 sind die beiden Uhren zu diesem Zweck angepaßt worden.

Es ist jetzt kein Problem mehr, ein Objekt Clock_e in $\mathcal{SP}roc$ zu konstruieren, so daß $\mathsf{cclock}_e : \mathsf{Clock}_e$ gilt, aber $\mathsf{cclock} : \mathsf{Clock}_e$ nicht. Es würde zum Beispiel ausreichen, das Alphabet $\Sigma_{\mathsf{Clock}_e}$ auf die Aktionen tick, tack und $\mathsf{success}$ zu begrenzen. Allerdings würde immer noch $\mathsf{clock} : \mathsf{Clock}_e$ gelten.

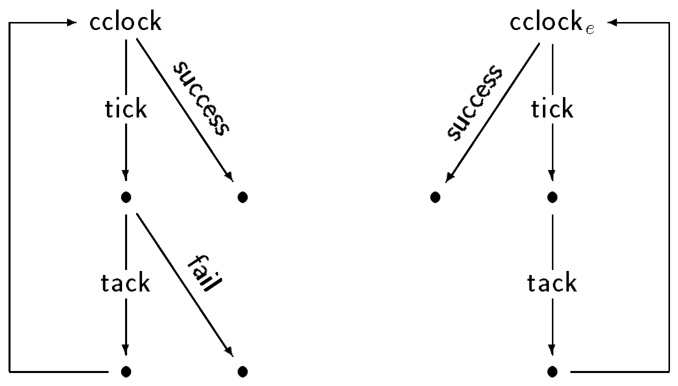


Abbildung 9: Veränderte Uhrmodelle

5.3 Bewertung

Das Beispiel der tick-tack Uhren offenbart eine erhebliche Schwäche von $\mathcal{SP}roc$. Lebendigkeitseigenschaften lassen sich nur in einer sehr abgeschwächten Form formulieren: Das gewünschte Ereignis tritt nur unter der Bedingung ein, daß der Prozeß nicht vorzeitig terminiert.

Inwieweit dies ein spezifisches Problem von $\mathcal{SP}roc$ ist, konnte im Rahmen dieser Arbeit nicht untersucht werden. In [Gay94] wird die Kategorie $\mathcal{SP}roc_D$, der Deadlock-freien Prozesse definiert. Allerdings unterscheidet Gay auch hier nur zwischen terminierenden und nicht terminierenden Prozessen. Erfolgreiches Terminieren wird nicht betrachtet.

6. Schlußbemerkung

Anhand dreier Beispiele wurden Probleme bei der Modellierung von Prozessen mittels $\mathcal{SP}roc$ und Beschränkungen in der Ausdrucksstärke verdeutlicht. Zur richtigen Einordnung der Ergebnisse muß allerdings vermerkt werden, daß erstens bewußt Beispiele ausgewählt wurden, bei denen die Grenzen von $\mathcal{SP}roc$ deutlich wurden. Zweitens ist $\mathcal{SP}roc$ nur ein Beispiel einer Interaktionskategorie. Ergebnisse, die mit $\mathcal{SP}roc$ erzielt wurden, können auf keinen Fall auf Interaktionskategorien im allgemeinen übertragen werden. Und schließlich sind Prozeßkategorien gegenwärtig Gegenstand intensiver Forschung, so daß neue Ergebnisse erwartet werden können, insbesondere was die Modellierung von Mobilität und asynchronem Verhalten betrifft.

Ein wesentlicher Vorteil von Interaktionskategorien besteht im Typsystem, das sie für Prozesse bereitstellen. Der π-Kalkül hat in seiner ursprünglichen Form kein Typsystem. Für den polyadischen π-Kalkül wird von Milner in [Mil91] eine Sortendisziplin eingeführt, die sichert, daß Sender und Empfänger die gleiche Anzahl von Namen über einen Kanal senden, beziehungsweise empfangen. Darauf aufbauend ist es ohne weiteres möglich, ein Typsystem für den π-Kalkül zu entwickeln, das Polymorphismus, Typoperatoren höherer

Ordnung und existentiell quantifizierte Typen unterstützt [Pie94].

In Typsystemen für den π-Kalkül werden jedoch immer nur die Kanäle getypt. Ein Konzept zur Typisierung von Prozessen existiert nicht. So können auch im getypten π-Kalkül Prozesse parallel komponiert werden, die gar nicht in der Lage sind, miteinander zu kommunizieren. Das ist innerhalb von Interaktionskategorien nicht möglich.

Allerdings wurde in Abschnitt 4 dieser Arbeit deutlich, daß das Typsystem für eine Reihe von Beschränkungen, denen Interaktionskategorien unterliegen, verantwortlich ist. Gerade bei der Modellierung von komplexen Systemen erscheint dem Autor das Kreieren und Weitergeben von Kanälen als ein wichtiges Mittel, um das Gesamtsystem überschaubar zu halten. In diesem Zusammenhang hält der Autor die Entscheidung Abramsky's, die Anzahl und Art der Kanäle über die ein Prozeß verfügt, als dessen Typ anzusehen, für falsch.

Literatur

[Abr94] Samson Abramsky. Interaction Categories and communicating sequential processes. In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C. A. R. Hoare*, pages 1–15. Prentice Hall International, 1994.

[AGN93] Samson Abramsky, Simon J. Gay, and Rajagopal Nagarajan. Interaction categories and the foundations of typed concurrent programming. In M. Broy, editor, *Deductive Program Design: Proceedings of the 1994 Marktoberdorf Summer School*, NATO ASI Series F: Computer and System Sciences. Springer-Verlag, 1993. to appear.

[Bar92] Henk P. Barendregt. Lambda calculi with types. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2. Oxford Science Publications, 1992.

[BS92] Bradfield and Stirling. Local model checking for infinite state spaces. *Theoretical Computer Science*, 96, 1992.

[EK65] Samuel Eilenberg and G. Max Kelly. Closed categories. In S. Eilenberg, D. K. Harrison, S. MacLane, and H. Röhrl, editors, *Proceedings of the Conference on Categorical Algebra, La Jolla, 1965*, pages 421–562. Springer-Verlag, 1965.

[FKKK94] Berndt Farwer, Peter-Alexis Kasten, Andreas Köster, and Michael Kopka. Nichtklassische Logiken. Diplomarbeit, Universität Hamburg, Dezember 1994.

[Gay94] Simon J. Gay. *Linear Types for Communicating Processes.* PhD thesis, University of London, 1994. To appear.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir89] Jean-Yves Girard. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[GN94] Simon J. Gay and Rajagopal Nagarajan. A typed calculus of synchronous processes. Submitted for publication, 1994.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil91] Robin Milner. The polyadic π calculus: a tutorial. Technical Report ECS-LFCS-91-180, Computer Science Dept., University of Edinburgh, October 1991.

[ML71] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.

[MOM90] N. Martí-Oliet and J. Meseguer. Duality in closed and linear categories. Technical Report SRI-CSL-90-01, SRI International, Computer Science Laboratory, February 1990.

[MPW92a] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I. *Information and Computation*, 100(1):1–40, 1992.

[MPW92b] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part II. *Information and Computation*, 100(1):41–77, 1992.

[Pie94] Benjamin C. Pierce. Programming in the pi-calculus: An experiment in programming language design. Tutorial notes on the PICT language. Available electronically, 1994.

[Poi92] Axel Poigné. Basic category theory. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, chapter Basic Category Theory. Oxford Science Publications, 1992.

[Rei95] Horst Reichel. Formale Modelle der Parallelität. Vorlesungsskript, Sommersemester 1994/95.

[See89] Robert A. G. Seely. Linear logic, $*$-autonomous categories and cofree coalgebras. In John W. Gray and Andre Scedrov, editors, *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 371–382, Providence, Rhode Island, 1989. American Mathematical Society.

[Sti92] Colin Stirling. Modal and temporal logics. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, chapter Modal and Temporal Logics. Oxford Science Publications, 1992.

[Sti93] Colin Stirling. Modal and temporal logics for processes. Notes for TEMPUS Summer School for Algebraic and Categorical Methods in Computer Science, Masaryk University, Brno, 1993.

[VM94] B. Victor and F. Moller. The Mobility Workbench — A tool for the π-calculus. In D. Dill, editor, *Proceedings of CAV'94*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.