Kompositionelle Minimierung endlicher verteilter Systeme

Diplomarbeit
im Fach Informatik
am
Lehrstuhl für Informatik II
der
Rheinisch–Westfälischen Technischen Hochschule Aachen
Prof. Dr. B. Steffen*

vorgelegt von

Gerald Lüttgen
Matrikelnummer 171906

betreut durch

Universitätsprofessor Dr. rer. nat. Bernhard Steffen

Aachen, im Februar 1994

*jetzt: Lehrstuhl für Informatik, Schwerpunkt Programmiersysteme, Universität Passau

Erklärung

Hiermit versichere ich, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, im Februar 1994 .....

Übersicht

In dieser Arbeit wird eine Methode zur kompositionellen Konstruktion eines im Sinne einer Bisimulationssemantik *minimalen Transitionssystems* vorgestellt, welches das Verhalten eines betrachteten *verteilten Systems* charakterisiert. Um die während der Konstruktion i.allg. auftretende *Zustandsexplosion* weitgehend zu vermeiden, werden neben Minimierungsschritten auch Reduktionsschritte durchgeführt. Dabei steht der Begriff eines *Reduktionsoperators* im Vordergrund, welcher im Kontext unerreichbare Zustände und Transitionen eliminiert. Dies geschieht mit Hilfe von *Interfacespezifikationen*, die die möglichen Kommunikationssequenzen zwischen Parallelkomponenten eines Systems beschreiben. Dadurch kann die durch das *Interleaving* von Aktionen kommunizierender Parallelkomponenten verursachte Zustandsexplosion kontrolliert werden. Die Effizienz dieser Methode hängt von der Exaktheit der vom Programmentwickler bereitgestellten Interfacespezifikationen ab. Die Korrektheit der Methode ist jedoch von der Korrektheit der Interfacespezifikationen unabhängig.

Inhaltsverzeichnis

Einleitung

Informelle Beschreibung der Arbeit

Die zentrale Frage bei der *Verifikation* endlicher verteilter Systeme lautet: Gilt $\mathcal{P} \models \Phi$, d.h. erfüllt ein System $\mathcal{P}$ die Eigenschaft Φ? Besonders interessant ist diese Fragestellung für *verteilte Systeme*, da viele Werkzeuge zur automatischen Verifikation auf der Konstruktion des globalen Zustandsgraphen basieren, welcher das betrachtete System darstellt. Der Zustandsraum eines solchen Zustandsgraphen wächst jedoch i.allg. exponentiell in der Anzahl der Parallelkomponenten des verteilten Systems. Diese Tatsache wird als *Zustandsexplosionsproblem* bezeichnet und führt in vielen Fällen dazu, daß der globale Zustandsgraph nicht im Speicher eines Rechners darstellbar ist und dadurch eine automatische Verifikation verhindert wird. Um diesem Problem zu begegnen, gibt es zwei Ansätze, nämlich den der *kompositionellen Verifikation* und den der *kompositionellen Minimierung*.

Kompositionelle Verifikationsmethoden kommen gänzlich ohne die Konstruktion eines globalen Zustandsgraphen aus. Der Beweis $\mathcal{P} \models \Phi$ wird geführt, indem der Prozeß $\mathcal{P}$ und die Eigenschaft Φ gemäß ihres syntaktischen Aufbaus in Beweisziele $\mathcal{P}_i \models \Phi_i$ zerlegt werden, deren Gültigkeit die Gültigkeit von $\mathcal{P} \models \Phi$ impliziert. Der Nachteil dieser Methoden liegt in der Komplexität der Regeln zur Zerlegung von $\mathcal{P}$ und Φ, die für verteilte Systeme notwendig sind. Vereinfachungen ergeben sich, falls für das betrachtete System *Kontextabhängigkeiten* bekannt sind. Die Spezifikation von Kontexten ist aufgrund der Komplexität verteilter Systeme schwierig und oft nur intuitiv möglich. Der Preis für die Vereinfachung der Beweisregeln besteht i.allg. in der Notwendigkeit, die Korrektheit der Kontextspezifikationen separat beweisen zu müssen.

Methoden zur kompositionellen Minimierung haben zum Ziel, den globalen Zustandsgraphen während seiner Konstruktion zu minimieren, indem semantisch äquivalente Zustände zusammengefaßt werden. Mit *Kompositionalität* ist hier die Wiederverwendbarkeit von Zwischenergebnissen der betrachteten Methode gemeint. Zur Definition einer adäquaten *Semantik* ist die folgende Beobachtung wichtig: Gewöhnlich genügt es für die Verifikation eines Systems, eine Abstraktion des globalen Zustandsgraphen zu betrachten, da viele Aktionen des Systems aus der Perspektive des Beobachters unwichtig sind. Solche Abstraktionen erlauben es, die Anzahl der Zustände eines Zustandsgraphen durch Zusammenfassen zu reduzieren, ohne daß sich das beobachtbare Verhalten des Systems ändert.

Als adäquat erweisen sich *Bisimulationssemantiken*, welche kompositionell sind, d.h. solche, die den parallelen Aufbau von Systemen respektieren.

Die in dieser Ausarbeitung vorgestellte Methode, die $\mathcal{RM}$–Methode, folgt den für Methoden zur kompositionellen Minimierung genannten Grundzügen. Ziel der $\mathcal{RM}$–Methode ist es, die *scheinbare Komplexität*, welche die Anzahl der Zustände des globalen Zustandsgraphen eines Systems bezeichnet, zu vermeiden. Ohne den globalen Zustandsgraphen explizit aufzubauen, wird sukzessive eine *minimale* Repräsentation dieses Graphen konstruiert. Die Zustandsanzahl einer solchen Repräsentation gibt die *reale Komplexität* wieder. Für eine Methode fällt als erstes der *naive Ansatz* auf, welcher als Hilfsmittel eine kompositionelle Bisimulationssemantik ausnutzt: „Gehe von einer Parallelkomponente des Systems aus, minimiere diese gemäß einer Bisimulationssemantik wie oben erläutert, nehme die nächste Parallelkomponente hinzu, minimiere wieder, ... und so fort, bis alle Parallelkomponenten berücksichtigt sind“. Als Ergebnis erhält man, unabhängig von der Reihenfolge, in der die Parallelkomponenten berücksichtigt werden, einen zum globalen Zustandsgraphen semantisch äquivalenten Zustandsgraphen mit einer geringeren (bzw. bei optimaler Ausnutzung von semantischen Äquivalenzen mit einer in bezug auf die gewählte kompositionelle Semantik minimalen) Zustandsanzahl. Dieser Ansatz führt während der Konstruktion jedoch i.allg. zu *Transitionssystemen*, wie Zustandsgraphen auch bezeichnet werden, deren Zustandsanzahl die scheinbare Komplexität übersteigt. Die maximale Zustandsanzahl eines Transitionssystems während der Konstruktion wird auch als *algorithmische Komplexität* bezeichnet. Die Ursache des Problems liegt darin, daß die Minimierung bezüglich einer Bisimulationssemantik ausschließlich *lokal* ist, d.h., daß nur die jeweils schon hinzugefügten Parallelkomponenten berücksichtigt werden. *Globale* Kontextabhängigkeiten, also die Interaktion mit den verbliebenen Parallelkomponenten, bleiben hingegen unberücksichtigt. Dadurch werden für die Minimierung Teile des Transitionssystems mitbetrachtet, welche im globalen Kontext unerreichbar sind. Greift man auf die Idee der Kontextspezifikationen, die hier *Interfacespezifikationen* heißen sollen, zurück, so lassen sich solche unerreichbaren Teile eines Transitionssystems jeweils vor der Minimierung entfernen. Formal wird dies mit Hilfe eines *Reduktionsoperators* realisiert.

Die $\mathcal{RM}$–Methode ist auf verteilte Systeme der Form $\mathcal{P} = (p_1 \| \cdots \| p_n)\langle L \rangle$ anwendbar, wobei die p_i (für $1 \leq i \leq n$, $n \in$) bereits als Transitionssysteme gegeben seien, $\|$ den Operator zur parallelen Komposition bezeichnet und $\langle L \rangle$ ein Fensteroperator ist, der nach außen hin nur Aktionen aus L sichtbar werden läßt. Ferner bezeichnet I_i $(1 \leq i \leq n-1)$ die Interfacespezifikation zwischen $R_i =_{df} (p_1 \| \cdots \| p_i)$ und $Q_i =_{df} (p_{i+1} \| \cdots \| p_n)$. Sie ist ebenfalls als Transitionssystem dargestellt und spezifiziert die Menge aller beobachtbaren *Kommunikationssequenzen* zwischen R_i und Q_i.
Die $\mathcal{RM}$–Methode beschreibt die sukzessive Konstruktion *partiell definierter* Transitionssysteme $\mathcal{P}_i$ $(1 \leq i \leq n)$, welche die folgenden Eigenschaften besitzen:

- $\mathcal{P}_i$ ist im Sinne einer Quasiordnung weniger spezifiziert als R_i. Diese Forderung ist die Grundlage für die Korrektheit der $\mathcal{RM}$–Methode.

- $\mathcal{P}_n$ ist semantisch äquivalent zu $\mathcal{P}$.[1]
- $\mathcal{P}_i$ ist ein Repräsentant seiner semantischen Äquivalenzklasse, welcher in dieser die wenigsten Zustände besitzt.

Unter der Voraussetzung, daß die benutzte Semantik die Eigenschaft Φ invariant läßt[2], d.h. für semantisch äquivalente Systeme P und Q gilt $P \models \Phi \iff Q \models \Phi$, genügt der Beweis von $\mathcal{P}_n \models \Phi$, um die Gültigkeit von $\mathcal{P} \models \Phi$ folgern zu können.

Die Korrektheit der $\mathcal{RM}$–Methode basiert dabei *nicht* auf der Korrektheit der vom Systementwickler anzugebenden Interfacespezifikationen: Sind Interfacespezifikationen nicht korrekt, so ist $\mathcal{P}_n$ i.allg. nicht semantisch äquivalent zu $\mathcal{P}$. Läßt sich jedoch $\mathcal{P}_n \models \Phi$ in dieser Situation beweisen, so gilt auch $\mathcal{P} \models \Phi$. Dies bedeutet, daß Interfacespezifikationen sogar erraten werden können. Ist hingegen $\mathcal{P}_n \models \Phi$ nicht beweisbar, so ist offen, ob $\mathcal{P} \models \Phi$ gültig ist oder nicht. Die $\mathcal{RM}$–Methode ist so aufgebaut, daß für ein total definiertes System $\mathcal{P}$ die nur partielle Definiertheit von $\mathcal{P}_n$ die Inkorrektheit einer Interfacespezifikation anzeigt. Im Falle der totalen Definiertheit von $\mathcal{P}_n$ sind $\mathcal{P}$ und $\mathcal{P}_n$ semantisch äquivalent und zwar unabhängig davon, ob die Interfacespezifikationen korrekt sind oder nicht.

Die $\mathcal{RM}$–Methode ist im folgenden Sinne optimal: Sind die anzugebenden Interfacespezifikationen korrekt und ist Φ eine bzgl. der gewählten Semantik konsistente Eigenschaft, dann sind $\mathcal{P}_n$ und $\mathcal{P}$ semantisch äquivalent, $\mathcal{P}_n$ hat in seiner semantischen Äquivalenzklasse eine minimale Zustandsanzahl und $\mathcal{P}_n \models \Phi \iff \mathcal{P} \models \Phi$. Ist also $\mathcal{P}_n \models \Phi$ nicht gültig, so folgt, daß auch $\mathcal{P} \models \Phi$ nicht gilt.

Gliederung der Arbeit

Kapitel 1 beschäftigt sich mit den (*mathematischen*) *Grundlagen* der $\mathcal{RM}$–Methode, insbesondere mit den Begriffen *Prozeß*, *semantische Äquivalenz*, *Quasiordnung* und *Interfacespezifikation*, welche anhand eines begleitenden Beispiels erläutert werden. In Kapitel 2 stehen *Reduktionsoperatoren* im Vordergrund, die sowohl aus der theoretischen, als auch aus der algorithmischen Perspektive betrachtet werden. Die $\mathcal{RM}$–Methode wird in Kapitel 3 vorgestellt, in dem auch ihre Korrektheit und ihre Optimalität nachgewiesen werden. Ein begleitendes Beispiel veranschaulicht die $\mathcal{RM}$–Methode, deren Mächtigkeit an einem weiteren Beispiel demonstriert wird. Kapitel 4 beschreibt Ansätze, wie die Handhabbarkeit der $\mathcal{RM}$–Methode in Spezialfällen verbessert werden kann. Die Hauptresultate dieser Arbeit sind in Kapitel 5 zusammengefaßt, in dem auch Ausblicke auf weitere Forschungsaktivitäten gegeben werden.
Der ausführliche Anhang enthält wichtige Ergänzungen. In Anhang A werden einige Begriffe aus der diskreten Mathematik definiert, die in dieser Arbeit Verwendung finden.

[1] Beachte: Für $1 \leq i \leq n-1$ gilt i.allg. aber nicht, daß $\mathcal{P}_i$ und R_i semantisch äquivalent sind.
[2] Man sagt auch: Φ ist bzgl. der gewählten Semantik konsistent.

Anhang B beinhaltet einige im jeweiligen Kapitel nicht vorgeführte Beweise. Die *Datenflußanalyse* spielt für die algorithmische Umsetzung der Anwendung von Reduktionsoperatoren eine zentrale Rolle. Ihre Grundlagen und ihre Anwendung innerhalb dieser Arbeit werden in Anhang C behandelt.

Literaturübersicht

Viele der in dieser Arbeit benötigten Grundbegriffe sind in [St91, St92, Mi89] zu finden. Der Grundstein der $\mathcal{RM}$–Methode ist von **Clarke**, **Long** und **McMillan** [CLM89] gelegt worden. Sie benutzen Interfacespezifikationen zur Beschreibung von Kontexten und versuchen, Informationen, die für das nach außen sichtbare Verhalten eines Systems unwichtig sind, zu lokalisieren. Diese Idee wird auch hier aufgegriffen. **Larsen** und **Thomsen** [LT87] sowie **Walker** [Wa88] beschreiben Kontextabhängigkeiten durch partielle Spezifikationen. Im Unterschied zu ihrer Arbeit wird hier eine modifizierte Quasiordnung benutzt, welche ausführlich von **Cleaveland** und **Steffen** in [CS] erläutert und dort in einen Zusammenhang mit derjenigen von Walker gestellt wird. Eine konkrete Strategie für (halb–)automatisches Beweisen, die die notwendige Benutzerunterstützung minimiert, ist von **Graf** und **Steffen** [GS91] vorgestellt worden. Der vorliegende Text ist eine Überarbeitung und Ergänzung ihrer Veröffentlichung. Er enthält im Gegensatz zu [GS91] ausführliche Beweise und die dafür notwendige Hilfsaussagen sowie ein anschauliches Beispiel für die $\mathcal{RM}$–Methode. Die Grundlagen der Datenflußanalyse werden von **Knoop** und **Steffen** in [KS91] beschrieben.

Kapitel 1

Grundlagen

In diesem Kapitel werden die grundlegenden Rahmenbedingungen vorgestellt, in die die $\mathcal{RM}$–Methode zur Minimierung endlicher verteilter Systeme eingebettet und innerhalb derer die Methode entwickelt werden soll. Der erste Teil dieses Kapitels beschäftigt sich mit *Prozessen*, mit deren Darstellung durch *erweiterte Transitionssysteme*, mit dem Begriff der *Isomorphie* von Prozessen, mit der Charakterisierung von Prozessen durch ihre *Sprachen* und mit *Operatoren* auf Prozessen, deren Bedeutung wir mittels einer *operationellen Semantik* definieren werden. Danach stehen eine spezielle *semantische Äquivalenz* und eine spezielle *Quasiordnung* auf Prozessen sowie deren Eigenschaften im Vordergrund. Beide werden in der $\mathcal{RM}$–Methode eine wichtige Rolle spielen. Zentral ist jedoch der Begriff einer *Interfacespezifikation*, der zum Abschluß des Kapitels definiert wird. Weiterhin werden die neuen Begriffe an einem Beispiel demonstriert.

1.1 Prozesse

Prozesse sind um ein *Undefiniertheitsprädikat* erweiterte Transitionssysteme, welche einen ausgezeichneten Startzustand besitzen. Das Undefiniertheitsprädikat wird, wie in den Kapiteln 2 und 3 zu sehen ist, ein wichtiges Hilfsprädikat im Hinblick auf die Praktikabilität der $\mathcal{RM}$–Methode sein und gleichzeitig die Grundlage für eine semantische Sichtweise der Methode liefern. Ferner wird auf Prozessen ein Parallel– und ein Fensteroperator definiert.

Erweiterte Transitionssysteme und Prozesse

Definition 1.1 (Erweiterte Transitionssysteme)
Ein erweitertes Transitionssystem T ist ein Quadrupel $(S, \mathcal{A} \cup \{\tau\}, \longrightarrow, \uparrow)$ mit:

1. *S ist eine endliche Menge von Zuständen.*
2. *$\mathcal{A}$ ist ein endliches Alphabet beobachtbarer Aktionen. τ repräsentiert eine nicht zu $\mathcal{A}$ gehörige interne, unbeobachtbare Aktion.*
3. *$\longrightarrow$ ist eine Transitionsrelation, d.h. $\longrightarrow \subseteq (S \times \mathcal{A} \cup \{\tau\} \times S)$.*

4. $\uparrow \subseteq S \times 2^{\mathcal{A} \cup \{\tau\}}$ *ist ein Prädikat, welches bewachte Undefiniertheit ausdrückt.*[1]

Statt $(p, L) \in \uparrow$ schreibt man auch $p{\uparrow}a$, $a \in L$, bzw. für $(p, a, q) \in \longrightarrow$ kurz $p \xrightarrow{a} q$ und sagt, daß p a–undefiniert ist bzw. daß p eine a–Transition nach q besitzt. Weiterhin notiert man $p \xrightarrow{a}$, falls ein p' existiert mit $p \xrightarrow{a} p'$. Typischerweise ist S eine Menge von Programmzuständen. Eine Transition $p \xrightarrow{a} q$ drückt aus, daß man vom Zustand p unter Beobachtung der Aktion a in den Zustand q übergehen kann. $p{\uparrow}a$ besagt, daß vom Zustand p aus eine a–Transition in einen undefinierten Zustand führt. In diesem Sinne ist der undefinierte Zustand bewacht.

Ein Prozeß ist ein erweitertes Transitionssystem zusammen mit einem ausgezeichneten Startzustand.

Definition 1.2 (Prozesse)
Gegeben sei ein erweitertes Transitionssystem $T = (S, \mathcal{A} \cup \{\tau\}, \longrightarrow, \uparrow)$. Ein Prozeß ist ein Tupel $((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p)$ für einen Zustand $p \in S$, wobei

- *S_p die Menge aller von p aus erreichbaren Zustände in T bezeichnet,*
- *$\mathcal{A}_p =_{df} \mathcal{A}$ gilt und*
- *$\longrightarrow_p$ und $\uparrow_p$ die auf S_p restringierten Relationen $\longrightarrow$ und $\uparrow$ sind.*

p heißt Startzustand des Prozesses. Die Menge aller Prozesse wird mit <u>Processes</u> bezeichnet.

Im folgenden wird ein Zustand p mit dem Prozeß $((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p)$ identifiziert.[2] Die Indizes werden der Einfachheit halber weggelassen, sofern dies nicht zu Mißverständnissen führt.

Definition 1.3
Ein Prozeß $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p)$ heißt deterministisch, falls für alle $p', p'', p''' \in S_p$ und alle $a \in \mathcal{A}_p \cup \{\tau\}$ gilt:

$$p' \xrightarrow{a}_p p'' \text{ und } p' \xrightarrow{a}_p p''' \text{ impliziert } p'' = p'''.$$

Den Bezug erweiterter Transitionssysteme zu „Standard"–Transitionssystemen stellt die folgende Definition her:

Definition 1.4
Ein Prozeß $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p)$ heißt total definiert, falls $\uparrow = \emptyset$ gilt. Ansonsten heißt er partiell definiert.

[1] 2^M bezeichnet für eine beliebige Menge M die Potenzmenge von M.

[2] Beachte, daß p sowohl einen Zustand als auch einen Prozeß bezeichnet. Dies ist eine in der Literatur übliche Konvention.

Erweiterte Transitionssysteme, die durch dieselben beschrifteten Graphen dargestellt werden und die gleichen Undefiniertheiten besitzen, sollen nun miteinander identifiziert werden. Dazu dient der Begriff der *Isomorphie von erweiterten Transitionssystemen.*

Definition 1.5 (Isomorphie von erweiterten Transitionssystemen)
Zwei erweiterte Transitionssysteme $(S_1, \mathcal{A}_1 \cup \{\tau\}, \longrightarrow_1, \uparrow_1)$ *und* $(S_2, \mathcal{A}_2 \cup \{\tau\}, \longrightarrow_2, \uparrow_2)$ *heißen isomorph, falls* $\mathcal{A}_1 = \mathcal{A}_2$ *gilt und eine bijektive Abbildung* $\nu : S_1 \longrightarrow S_2$ *existiert mit:*

1. $\forall\, p', p'' \in S_1\ \forall\, a \in \mathcal{A}_1 \cup \{\tau\}.\quad p' \xrightarrow{a}_1 p'' \iff \nu(p') \xrightarrow{a}_2 \nu(p'')$ *und*
2. $\forall\, p' \in S_1\ \forall\, a \in \mathcal{A}_1 \cup \{\tau\}.\quad p' \uparrow_1 a \iff \nu(p') \uparrow_2 a.$

Diese Definition läßt sich nun wie folgt auf Prozesse ausdehnen:

Definition 1.6 (Isomorphie von Prozessen)
Zwei Prozesse $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p)$ *und* $q = ((S_q, \mathcal{A}_q \cup \{\tau\}, \longrightarrow_q, \uparrow_q), q)$ *heißen isomorph, in Zeichen* $p \cong q$*, falls* $\mathcal{A}_p = \mathcal{A}_q$ *gilt und eine bijektive Abbildung* $\nu : S_p \longrightarrow S_q$ *existiert mit:*

1. $\nu(p) = q,$
2. $\forall\, p', p'' \in S_p\ \forall\, a \in \mathcal{A}_p \cup \{\tau\}.\quad p' \xrightarrow{a}_p p'' \iff \nu(p') \xrightarrow{a}_q \nu(p'')$ *und*
3. $\forall\, p' \in S_p\ \forall\, a \in \mathcal{A}_p \cup \{\tau\}.\quad p' \uparrow_p a \iff \nu(p') \uparrow_q a.$

Isomorphe Prozesse unterscheiden sich also lediglich in der Benennung ihrer Zustände. Da hier nur Gleichheit von Prozessen bis auf Isomorphie interessiert, wird im folgenden für $\cong$ auch $=$ geschrieben.

Prozesse lassen sich ähnlich wie endliche Automaten auch durch die von ihnen erzeugte bzw. erkannte *Sprache* charakterisieren. Die folgende Definition der Sprache von Prozessen bedient sich bereits der in Abschnitt 1.2 definierten *schwachen* Transitionsrelation $\Longrightarrow$ bzw. des *schwachen* Undefiniertheitsprädikats $\Uparrow$, welche die Relation $\longrightarrow$ bzw. das Prädikat $\uparrow$ aus der Sicht eines Beobachters beschreibt, indem die unbeobachtbare Aktion τ „herausgefiltert“ wird. Ferner bezeichnet „*“ wie gewöhnlich den *Kleeneschen Sternoperator.*

Definition 1.7 (Sprache von Prozessen)
Die Sprache $\mathcal{L}(p)$ *eines partiell definierten Prozesses* p *ist durch den kleinsten Fixpunkt des folgenden Gleichungssystems definiert:*

$$\mathcal{L}(p) = \begin{cases} \mathcal{A}_p^* & \text{, falls } p \Uparrow \varepsilon \\ \bigcup \{\, a \cdot \mathcal{L}_a(p) \mid \mathcal{L}_a(p) \neq \emptyset \,\} \cup \{\varepsilon\} & \text{, sonst} \end{cases}$$

und

$$\mathcal{L}_a(p) = \begin{cases} \mathcal{A}_p^* & \text{, falls } p \Uparrow a \\ \bigcup \{\, \mathcal{L}(p') \mid p \overset{a}{\Rightarrow} p' \,\} & \text{, sonst} \end{cases}$$

für jede Aktion a. Ferner wird zu einer gegebenen Sprache $\mathcal{L}$ die Sprache aller ihrer a–Suffixe, d.h. die Menge $\{w \mid a \cdot w \in \mathcal{L}\}$, mit $\mathcal{L}_a$ bezeichnet.

Die Wohldefiniertheit dieser Definition ergibt sich mit Hilfe der elementaren Fixpunkttheorie. Für total definierte Prozesse entfällt jeweils die erste Alternative aus obiger Definition, die dann mit der in der Automatentheorie üblichen Definition übereinstimmt. Die von einem undefinierten Zustand eines partiell definierten Prozesses aus erkannte Sprache ist dabei die Menge aller Wörter, die aus dem Alphabet des Prozesses gebildet werden können. Entsprechend ist von einem a–undefinierten Zustand aus die Menge aller der Wörter über dem Alphabet erkennbar, welche mit einem a beginnen. Dies entspricht der Intuition, daß man über die Sprache eines undefinierten Zustands nichts weiß und deshalb das unsichere Wissen nur wie oben geschehen ausdrücken kann („worst-case–Annahme").

Parallel– und Fensteroperator

Hier geht es um die Einführung eines binären *Parallel–* und eines unären *Fensteroperators* auf Prozessen. Die Bedeutung beider Operatoren wird mittels einer operationellen Semantik definiert. Intuitiv beschreibt $p \| q$ die parallele Komposition der Prozesse p und q. Die *Synchronisation* von p und q beruht auf gemeinsamen und das sogenannte *Interleaving* auf den restlichen Aktionen. Der Prozeß $p\langle L \rangle$ resultiert aus dem Prozeß p, wobei nur die Aktionen aus L nach außen sichtbar sind.

Definition 1.8 (Operationelle Semantik)
Seien $p = ((S_p,\ \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p,\ \uparrow_p), p)$, $q = ((S_q,\ \mathcal{A}_q \cup \{\tau\}, \longrightarrow_q,\ \uparrow_q), q) \in$ <u>Processes</u>, sowie $p', p'' \in S_p$, $q', q'' \in S_q$ und L eine beliebige Menge sichtbarer Aktionen. Dann sind die Alphabete der Prozesse $p\langle L \rangle$ und $p \| q$ durch $\mathcal{A}_{p\langle L \rangle} =_{df} \mathcal{A}_p \cap L$ bzw. $\mathcal{A}_{p \| q} =_{df} \mathcal{A}_p \cup \mathcal{A}_q$ definiert. Ihre Zustandsmengen sind Teilmengen von $\{\overline{p}\langle L \rangle \mid \overline{p} \in S_p\}$ bzw. $\{\overline{p} \| \overline{q} \mid \overline{p} \in S_p, \overline{q} \in S_q\}$, welche jeweils die Zustände enthalten, die vom Startzustand $p\langle L \rangle$ bzw. $p \| q$ gemäß den nachstehenden Regeln für die Transitionsrelationen erreichbar sind:[3]

[3] Zur Notation:

$$\frac{\text{Prämisse}}{\text{Konklusion}} \quad \text{Nebenbedingungen}$$

1. $$\frac{p' \xrightarrow{a}_p p''}{p'\langle L\rangle \xrightarrow{a}_{p\langle L\rangle} p''\langle L\rangle} \quad a \in L$$

2. $$\frac{p' \xrightarrow{a}_p p''}{p'\langle L\rangle \xrightarrow{\tau}_{p\langle L\rangle} p''\langle L\rangle} \quad a \notin L$$

3. $$\frac{p' \xrightarrow{a}_p p''}{p'\|q' \xrightarrow{a}_{p\|q} p''\|q'} \quad a \notin \mathcal{A}_q$$

4. $$\frac{q' \xrightarrow{a}_q q''}{p'\|q' \xrightarrow{a}_{p\|q} p'\|q''} \quad a \notin \mathcal{A}_p$$

5. $$\frac{p' \xrightarrow{a}_p p'' \quad q' \xrightarrow{a}_q q''}{p'\|q' \xrightarrow{a}_{p\|q} p''\|q''} \quad a \neq \tau$$

Die Undefiniertheitsprädikate von $p\langle L\rangle$ und $p\|q$ werden durch die folgenden Regeln definiert:

6. $$\frac{p'\uparrow_p a}{p'\langle L\rangle\uparrow_{p\langle L\rangle} a} \quad a \in L$$

7. $$\frac{p'\uparrow_p a}{p'\langle L\rangle\uparrow_{p\langle L\rangle} \tau} \quad a \notin L$$

8. $$\frac{p'\uparrow_p a}{(p'\|q')\uparrow_{p\|q} a} \quad a \notin \mathcal{A}_q$$

9. $$\frac{p'\uparrow_p a}{(p'\|q')\uparrow_{p\|q} a} \quad q' \xrightarrow{a}_q q''$$

---

ist wie folgt zu verstehen: Gelten die *Nebenbedingungen*, so läßt sich aus der *Prämisse* die *Konklusion* folgern. Beachte weiterhin in der Definition, daß insbesondere $\tau \notin \mathcal{A}_p$ und $\tau \notin \mathcal{A}_q$ gilt.

10. $\dfrac{q'\uparrow_q a}{(p'\|q')\uparrow_{p\|q} a} \quad a \notin \mathcal{A}_p$

11. $\dfrac{q'\uparrow_q a}{(p'\|q')\uparrow_{p\|q} a} \quad p' \xrightarrow{a}_p p''$

12. $\dfrac{p'\uparrow_p a \quad q'\uparrow_q a}{(p'\|q')\uparrow_{p\|q} a}$.

Wichtig ist die Definition der operationellen Semantik bzgl. des Undefiniertheitprädikats für den Paralleloperator. Sie besagt, daß $p'\uparrow a$ bereits $(p'\|q')\uparrow a$ impliziert, falls der Prozeß q die Ausführung einer a–Transition im Zustand q' nicht verhindert, d.h. falls $a \notin \mathcal{A}_q$ oder $q' \xrightarrow{a} q''$ gilt. q kann in diesem Fall die a–Undefiniertheit von $p'\|q'$ nur verhindern, wenn sich p und q auf der Aktion a synchronisieren müssen (d.h. falls $a \in \mathcal{A}_p \cap \mathcal{A}_q$ gilt), und q im Zustand q' keine a–Transition ausführen kann. Der Nutzen dieser Definition wird, da das Undefiniertheitsprädikat als ein nützliches Hilfsprädikat benutzt werden wird und außerdem eine semantische Sichtweise induziert, erst in Kapitel 2 bzw. mit Definition 1.15 deutlich.

Proposition 1.9 (Assoziativität und Kommutativität)
Der Paralleloperator $\|$ ist in folgendem Sinne assoziativ und kommutativ, wobei ν jeweils eine bijektive Abbildung gemäß Definition 1.6 bezeichnet:

1. $\forall\, p, q, r \in \underline{\textit{Processes}}.\ (p\|q)\|r \cong p\|(q\|r)$
   wobei $((p', q'), r') \xmapsto{\nu} (p', (q', r'))\ \forall\, p' \in S_p,\ q' \in S_q,\ r' \in S_r.$

2. $\forall\, p, q \in \underline{\textit{Processes}}.\ p\|q \cong q\|p$
   wobei $(p', q') \xmapsto{\nu} (q', p')\ \forall\, p' \in S_p,\ q' \in S_q.$

Beweisskizze
Der Beweis ergibt sich sofort aus Definition 1.8 der operationellen Semantik, da die Regeln für die Transitionsrelation bzw. für das Undefiniertheitsprädikat bzgl. des Paralleloperators *symmetrisch* sind. *Symmetrisch* bedeutet, daß es zu jeder Regel für $p\|q$ eine Regel gibt, bei der die Rollen von p' und q' vertauscht sind.

Als Konsequenz dieser Proposition ergibt sich die Wohldefiniertheit von Prozessen der Form $(p_1\| \cdots \|p_n)\langle L\rangle$. Da diese Prozesse für das Problem der Zustandsexplosion verantwortlich sind, wird sich die zu entwickelnde $\mathcal{RM}$–Methode auf Prozesse dieser Form, welche im CCS–Kalkül *Standard Concurrent Form* genannt wird (vgl. [Mi89]), konzentrieren.

Für die $\mathcal{RM}$–Methode wird das folgende Lemma eine zentrale Rolle spielen, welches die Korrespondenz zwischen Parallel– und Fensteroperator widerspiegelt:

Lemma 1.10 (Parallel–/Fensteroperatorkorrespondenz)
Für alle Prozesse $p, q \in$ Processes und alle Mengen L, L' beobachtbarer Aktionen gilt:

$$(p \| q)\langle L \rangle \cong (p\langle L' \rangle \| q)\langle L \rangle \text{ , falls } L' \supseteq L \cup (\mathcal{A}_p \cap \mathcal{A}_q).$$

Der Beweis erfordert umfangreiche Fallunterscheidungen bzgl. der operationellen Semantik und ist deshalb in Anhang B zu finden. Die Bedeutung dieses Lemmas liegt darin, Informationen bezüglich der globalen Unsichtbarkeit zu lokalisieren. Diese einfache Beobachtung wird zur Effizienz der $\mathcal{RM}$–Methode beitragen.

Ein begleitendes Beispiel

Im folgenden wird ein sehr einfaches Beispielsystem vorgestellt, anhand dessen die in dieser Ausarbeitung vorkommenden Begriffe erläutert und Methoden demonstriert werden sollen.
Das Beispielsystem $Sys =_{df} (P_1 \| B \| P_2)\langle \{tk1, tk2\} \rangle$ (vgl. Abbildung 1.1) besteht aus zwei *Prozessen* P_1 und P_2 und einem *Buffer* (vgl. Abbildung 1.2) mit den zugehörigen Alphabeten $\mathcal{A}_{P_1} = \{tk1, tk2, rb1, sb1\}$, $\mathcal{A}_{P_2} = \{tk1, tk2, rb2, sb2\}$ und $\mathcal{A}_B = \{rb1, sb1, rb2, sb2\}$. Der Buffer dient den Prozessen dazu, wechselseitig Daten vom jeweils anderen Prozeß zu empfangen und neue an ihn zu senden. Der *wechselseitige Ausschluß* wird durch ein Token realisiert, welches zwischen den Prozessen hin– und hergeschoben wird und zum Zugriff auf den Buffer berechtigt.

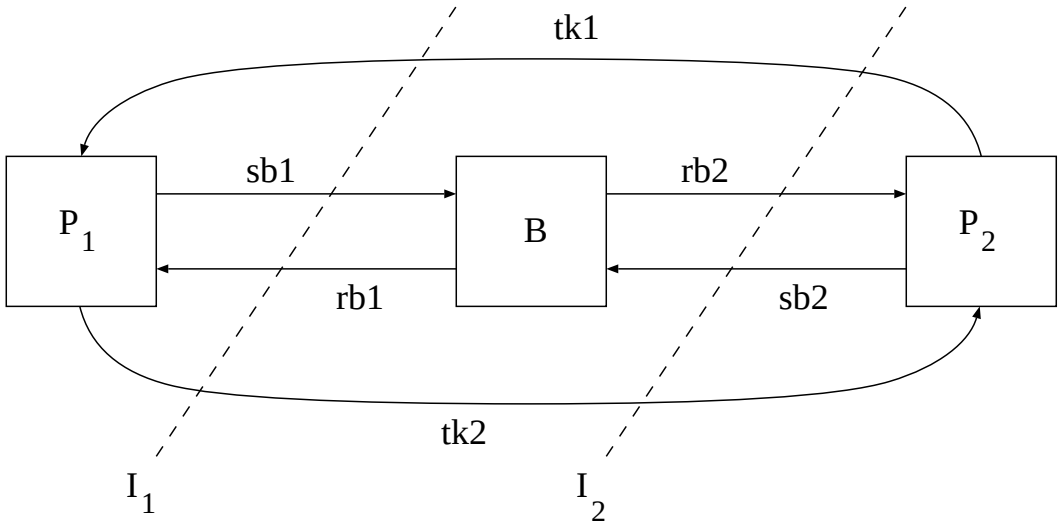


Abbildung 1.1: Beispielsystem

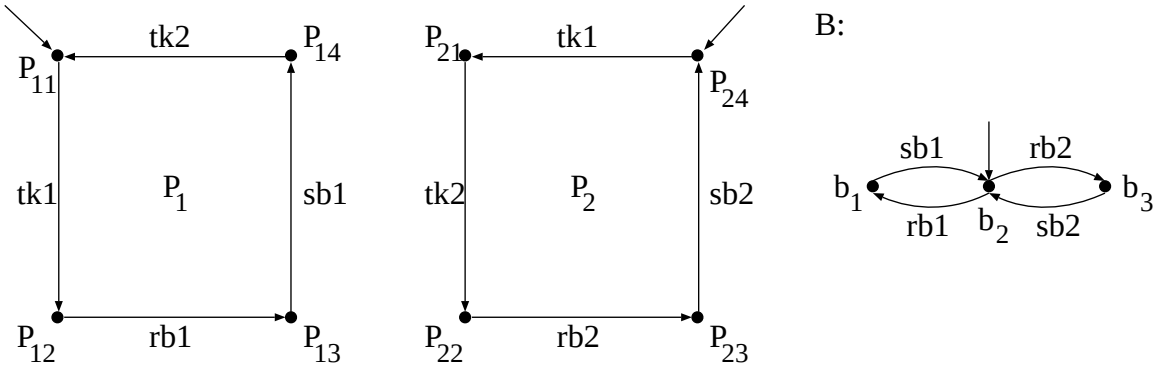


Abbildung 1.2: Spezifikationen für das Beispielsystem

1.2 Semantische Äquivalenz und Quasiordnung

Hier stehen die beiden Begriffe *Semantische Äquivalenz* und *Quasiordnung* im Mittelpunkt. Die semantische Äquivalenz führt zur Definition einer Funktion, welche die eigentliche Minimierung eines Transitionssystems in der $\mathcal{RM}$–Methode bewirkt. Die Quasiordnung stellt den Rahmen dar, in dem die Korrektheit der $\mathcal{RM}$–Methode bewiesen wird. Beide Begriffe benötigen jedoch zunächst die Definition der *schwachen Transitionsrelation* $\Longrightarrow$. Dies ist notwendig, da die (starke) Transitionsrelation $\longrightarrow$ nicht zwischen *beobachtbaren* Aktionen und der für den Beobachter *unsichtbaren* Aktion τ unterscheidet, sondern alle Aktionen gleichwertig behandelt. Daher soll die Relation $\Longrightarrow$ ein Transitionssystem aus der Sicht eines Beobachters beschreiben. Konsequenterweise wird dieses Konzept wegen der hier verwendeten erweiterten Transitionssysteme auch auf das Undefiniertheitsprädikat $\uparrow$ übertragen, so daß man ein schwaches Undefiniertheitsprädikat $\Uparrow$ erhält.

Definition 1.11 (Schwache Transitionsrelation und Undefiniertheitsprädikat) *Sei* $\longrightarrow \subseteq (S \times \mathcal{A} \cup \{\tau\} \times S)$ *die Transitionsrelation und* $\uparrow \subseteq S \times 2^{\mathcal{A} \cup \{\tau\}}$ *das Undefiniertheitsprädikat eines erweiterten Transitionssystems. Dann sind die schwache Transitionsrelation* $\Longrightarrow \subseteq S \times \mathcal{A} \times S$ *und das schwache Undefiniertheitsprädikat* $\Uparrow \subseteq S \times 2^{\mathcal{A}}$ *jeweils als die kleinste Relation mit den folgenden Eigenschaften definiert* ($p, q \in S$, $a \in \mathcal{A}$)*:*

1. $p \xrightarrow{\tau}{}^{*} \xrightarrow{a} \xrightarrow{\tau}{}^{*} q$ *impliziert* $p \overset{a}{\Longrightarrow} q$.
2. $p \xrightarrow{\tau}{}^{*} q$ *impliziert* $p \overset{\epsilon}{\Longrightarrow} q$.
3. $q \uparrow a \ \wedge \ p \overset{\epsilon}{\Longrightarrow} q$ *impliziert* $p \Uparrow a$.
4. $q \uparrow \tau \ \wedge \ p \overset{\epsilon}{\Longrightarrow} q$ *impliziert* $p \Uparrow \epsilon$.
5. $q \Uparrow \epsilon \ \wedge \ p \overset{a}{\Longrightarrow} q$ *impliziert* $p \Uparrow a$.
6. $p \Uparrow \epsilon$ *impliziert* $p \Uparrow a$.

Diese Definition ist mit der operationellen Semantik des Parallel– und des Fensteroperators in dem Sinne konsistent, daß die Regeln aus Definition 1.8 auch gelten, wenn man $\longrightarrow$ durch $\Longrightarrow$ und $\uparrow$ durch $\Uparrow$ ersetzt.

Semantische Äquivalenz

Wie in der Einleitung bereits erwähnt, basiert die Minimierung von Transitionssystemen auf der Idee, aus der Sicht des Beobachters äquivalente Zustände zusammenzufassen. Diese Semantik macht sich daher die Relation bzw. das Prädikat aus Definition 1.11 zunutze.

Definition 1.12 (Semantische Äquivalenzrelation)
$\approx^d$ ist die Vereinigung aller Relationen $R \subseteq S \times S$, für die folgendes gilt:
$p\,R\,q$ impliziert für alle $a \in \mathcal{A}$:

1. *$p \Uparrow a$ genau dann, wenn $q \Uparrow a$.*
2. *$p \stackrel{a}{\Longrightarrow} p'$ impliziert $\exists\, q'.\ q \stackrel{a}{\Longrightarrow} q' \ \wedge\ p'\,R\,q'$.*
3. *$q \stackrel{a}{\Longrightarrow} q'$ impliziert $\exists\, p'.\ p \stackrel{a}{\Longrightarrow} p' \ \wedge\ p'\,R\,q'$.*

Beachte, daß sich die Definition von $\approx^d$ an die der wohlbekannten *Beobachtungsäquivalenz* $\approx$ von Milner (vgl. [Mi89]) anlehnt. Bei obiger Definition kommt lediglich die erste Bedingung bzgl. des Undefiniertheitsprädikats hinzu. Ferner können nur solche Prozesse $\approx^d$–äquivalent sein, die dasselbe Kommunikationspotential (d.h. dasselbe Alphabet) besitzen. Die in dieser Ausarbeitung vorgestellte Methode ist für alle *Äquivalenzrelationen*, welche die Aussagen der Sätze 1.19 und 1.20 erfüllen, korrekt. Daher ist die Äquivalenzrelation in der Methode leicht austauschbar.

Lemma 1.13
Die Relation $\approx^d$ aus Definition 1.12 ist eine Äquivalenzrelation.

Beweis
Es sind die Punkte (1)–(3) aus Definition A.1 einer Äquivalenzrelation zu verifizieren. Seien $p, q, r \in$ <u>Processes</u>.

1. **Transitivität**

   Z.zg.: $p \approx^d q \ \wedge\ q \approx^d r$ impliziert $p \approx^d r$.
   Beweis: Gelte $p \approx^d q$ und $q \approx^d r$. Sei $R =_{df} \{(\overline{p}, \overline{r}) \mid \exists\, \overline{q}.\ \overline{p} \approx^d \overline{q} \ \wedge\ \overline{q} \approx^d \overline{r},\ \ \overline{p} \in S_p,\ \overline{q} \in S_q,\ \overline{r} \in S_r\}$ und $(\overline{p}, \overline{r}) \in R$ sowie $a \in \mathcal{A}$ beliebig. Es ist zu zeigen:

   (a) $\overline{p} \Uparrow a$ genau dann, wenn $\overline{r} \Uparrow a$,

   (b) $\overline{p} \stackrel{a}{\Longrightarrow} \overline{p}'$ impliziert $\exists\, \overline{r}'.\ \overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ (\overline{p}', \overline{r}') \in R$ und

   (c) $\overline{r} \stackrel{a}{\Longrightarrow} \overline{r}'$ impliziert $\exists\, \overline{p}'.\ \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ (\overline{p}', \overline{r}') \in R$.

Dann gilt $R \subseteq \approx^d$, und wegen $(p, r) \in R$ folgt $p \approx^d r$.

ad (a):

$$\begin{array}{lcl} & & \overline{p} \Uparrow a \\ (\overline{p} \approx^d \overline{q}) & \Longleftrightarrow & \overline{q} \Uparrow a \\ (\overline{q} \approx^d \overline{r}) & \Longleftrightarrow & \overline{r} \Uparrow a \end{array}$$

ad (b):

$$\begin{array}{lcl} & & \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \\ (\overline{p} \approx^d \overline{q}) & \Rightarrow & \exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \ \wedge\ \overline{p}' \approx^d \overline{q}' \\ (\overline{q} \approx^d \overline{r}) & \Rightarrow & \exists\, \overline{q}'.\ (\exists\, \overline{r}'.\ (\overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ \overline{q}' \approx^d \overline{r}') \ \wedge\ \overline{p}' \approx^d \overline{q}') \\ & \Rightarrow & \exists\, \overline{r}'.\ (\overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ \exists\, \overline{q}'.\ (\overline{p}' \approx^d \overline{q}' \ \wedge\ \overline{q}' \approx^d \overline{r}')) \\ (\text{Def. von } R) & \Rightarrow & \exists\, \overline{r}'.\ \overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ (\overline{p}', \overline{r}') \in R \end{array}$$

ad (c):

$$\begin{array}{lcl} & & \overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \\ (\overline{q} \approx^d \overline{r}) & \Rightarrow & \exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \ \wedge\ \overline{q}' \approx^d \overline{r}' \\ (\overline{p} \approx^d \overline{q}) & \Rightarrow & \exists\, \overline{q}'.\ (\exists\, \overline{p}'.\ (\overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ \overline{p}' \approx^d \overline{q}') \ \wedge\ \overline{q}' \approx^d \overline{r}') \\ & \Rightarrow & \exists\, \overline{p}'.\ (\overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ \exists\, \overline{q}'.\ (\overline{p}' \approx^d \overline{q}' \ \wedge\ \overline{q}' \approx^d \overline{r}')) \\ (\text{Def. von } R) & \Rightarrow & \exists\, \overline{p}'.\ \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ (\overline{p}', \overline{r}') \in R \end{array}$$

2. **Symmetrie**

Z.zg.: $p \approx^d q$ impliziert $q \approx^d p$.
Beweis: Sei $p \approx^d q$, $R =_{df} \{(\overline{q}, \overline{p}) \mid \overline{p} \approx^d \overline{q},\ \ \overline{p} \in S_p, \overline{q} \in S_q\}$ und $(\overline{q}, \overline{p}) \in R$ beliebig, sowie $a \in \mathcal{A}$ beliebig. Es ist zu zeigen:

(a) $\overline{q} \Uparrow a$ genau dann, wenn $\overline{p} \Uparrow a$,

(b) $\overline{q} \stackrel{a}{\Longrightarrow} \overline{q}'$ impliziert $\exists\, \overline{p}'.\ \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ (\overline{q}', \overline{p}') \in R$ und

(c) $\overline{p} \stackrel{a}{\Longrightarrow} \overline{p}'$ impliziert $\exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \ \wedge\ (\overline{q}', \overline{p}') \in R$.

Dann gilt $R \subseteq \approx^d$, und wegen $(q, p) \in R$ folgt $q \approx^d p$.

ad (a): Klar, da $\overline{p} \approx^d \overline{q}$.

ad (b):

$$\begin{array}{lcl} & & \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \\ (\overline{p} \approx^d \overline{q}) & \Rightarrow & \exists\, \overline{p}'.\ \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ \overline{p}' \approx^d \overline{q}' \\ (\text{Def. von } R) & \Rightarrow & \exists\, \overline{p}'.\ \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ (\overline{q}', \overline{p}') \in R \end{array}$$

ad (c):

$$\begin{array}{lcl} & & \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \\ (\overline{p} \approx^d \overline{q}) & \Rightarrow & \exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \ \wedge\ \overline{p}' \approx^d \overline{q}' \\ (\text{Def. von } R) & \Rightarrow & \exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \ \wedge\ (\overline{q}', \overline{p}') \in R \end{array}$$

3. **Reflexivität**

Z.zg.: $p \approx^d p$.
Beweis: Sei $R =_{df} \{(\overline{p}, \overline{p}) \mid \overline{p} \in S_p\}$ und $(\overline{p}, \overline{p}) \in R$ beliebig, sowie $a \in \mathcal{A}$ beliebig. Trivialerweise gilt:

- $\overline{p} \Uparrow a$ genau dann, wenn $\overline{p} \Uparrow a$, sowie
- $\overline{p} \stackrel{a}{\Longrightarrow} \overline{p}'$ impliziert $\exists\, \overline{\overline{p}} \equiv \overline{p}'.\ \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ (\overline{p}', \overline{p}') \in R$.[4]

Folglich gilt $R \subseteq \approx^d$ und wegen $(p, p) \in R$ auch $p \approx^d p$.

Aus (1)–(3) folgt mit Definition A.1, daß $\approx^d$ eine Äquivalenzrelation ist.

Beispiel 1.14
Für beliebige Prozesse $p, q \in$ <u>Processes</u> gilt:

$$p \cong q \ \ \textit{impliziert} \ \ p \approx^d q\,.$$

[4] Dabei bedeutet $p \equiv q$, daß die Prozesse (bzw. Zustände) p und q identisch sind.

Insbesondere sind syntaktisch gleiche Prozesse $\approx^d$–äquivalent (denn syntaktisch gleiche Prozesse sind vermöge $\nu = id$ isomorph).

Beweisskizze
Mit Hilfe von Definition 1.6 kann man leicht zeigen,[5] daß

$$R =_{df} \{(p, \nu(p)) \mid p \in S_p\}$$

eine Relation im Sinne von Definition 1.12 ist. Wegen $(p, q) = (p, \nu(p)) \in R$ folgt die Behauptung.

Quasiordnung

Die folgende *Quasiordnung* definiert intuitiv eine „weniger definiert als" – Relation zwischen Prozessen.

Definition 1.15 (Spezifikations–Quasiordnung)
Die Spezifikations–Quasiordnung ist die Vereinigung aller Relationen $R \subseteq S \times S$, für die folgendes gilt:
$p\,R\,q$ impliziert für alle $a \in \mathcal{A}$ mit $\neg(p \Uparrow a)$:

1. $\neg(q \Uparrow a)$,
2. $p \stackrel{a}{\Longrightarrow} p'$ *impliziert* $\exists q'.\ q \stackrel{a}{\Longrightarrow} q' \wedge p'\,R\,q'$ *und*
3. $q \stackrel{a}{\Longrightarrow} q'$ *impliziert* $\exists p'.\ p \stackrel{a}{\Longrightarrow} p' \wedge p'\,R\,q'$.

Beachte dabei, daß wie bei Definition 1.12 nur solche Prozesse in Relation zueinander stehen können, die dasselbe Kommunikationspotential besitzen. läßt sich als Spezifikations–Implementations–Quasiordnung verstehen: Eine Implementation q erfüllt die Anforderungen einer partiellen Spezifikation p genau dann, wenn $p \quad q$ gilt. Die in dieser Arbeit vorgestellte Methode ist für alle Quasiordnungen, welche die Sätze 1.19, 1.20 und Definition 2.1 (i) erfüllen, korrekt. Daß hier die Spezifikations–Quasiordnung gewählt wurde, liegt daran, daß eine im Vergleich zu Walkers Quasiordnung $\sqsubseteq$ [Wa88] adäquatere Spezifikations–Implementations–Quasiordnung ist, da weniger Prozesse unterscheidet: Gilt $p \sqsubseteq q$, so impliziert dies bereits $p \quad q$. Der Grund hierfür ist, daß im Falle der a–Undefiniertheit eines Zustands von p die Relation keine Bedingung an das Verhalten des korrespondierenden Zustands von q stellt. Die Spezifikations–Quasiordnung, die insbesondere für eine einfache algorithmische Implementierung geeignet ist, erlaubt somit mehr Freiheiten.
Beachte, daß $p \quad q$ i.allg. nicht impliziert, daß der Prozeß p weniger Zustände als der Prozeß q besitzt, denn:

[5] Man beachte, daß $\mathcal{A}_p = \mathcal{A}_q$ gilt.

Beispiel 1.16
Betrachte für p einen Prozeß, der eine a–Transition in einen a–undefinierten Zustand besitzt, und für q einen Prozeß, der eine a–Transition von q nach q hat, also beliebig viele a–Aktionen hintereinander ausführen kann. Nach Definition von gilt offensichtlich $p \quad q$, aber p besitzt mehr Zustände als q (vgl. Abbildung 1.3).

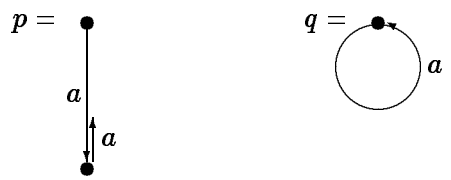


Abbildung 1.3: Gegenbeispiel

Lemma 1.17
Die Relation aus Definition 1.15 ist eine Quasiordnung.

Beweis
Es sind die Punkte (1) und (2) aus der Definition A.2 einer Quasiordnung zu verifizieren. Dazu seien $p, q, r \in$ Processes.

1. **Transitivität**

   Z.zg.: $p \quad q \wedge q \quad r$ impliziert $p \quad r$.
   Beweis: Gelte $p \quad q$ und $q \quad r$. Sei

   $$R =_{df} \{(\overline{p}, \overline{r}) \mid \exists\, \overline{q}.\ \overline{p} \quad \overline{q} \wedge \overline{q} \quad \overline{r},\ \ \overline{p} \in S_p, \overline{q} \in S_q, \overline{r} \in S_r\}$$

   und $(\overline{p}, \overline{r}) \in R$ sowie $a \in \mathcal{A}$ beliebig. Ferner gelte $\neg(\overline{p} \Uparrow a)$. Es ist zu zeigen:

   (a) $\neg(\overline{r} \Uparrow a)$,
   (b) $\overline{p} \stackrel{a}{\Longrightarrow} \overline{p}'$ impliziert $\exists\, \overline{r}'.\ \overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ (\overline{p}', \overline{r}') \in R$ und
   (c) $\overline{r} \stackrel{a}{\Longrightarrow} \overline{r}'$ impliziert $\exists\, \overline{p}'.\ \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ (\overline{p}', \overline{r}') \in R$.

   Dann gilt $R \subseteq$, und wegen $(p, r) \in R$ folgt $p \quad r$.

   ad (a):

   $$\neg(\overline{p} \Uparrow a)$$

   $$(\overline{p} \quad \overline{q}) \quad \Rightarrow \quad \neg(\overline{q} \Uparrow a)$$

   $$(\overline{q} \quad \overline{r}) \quad \Rightarrow \quad \neg(\overline{r} \Uparrow a)$$

ad (b):

$$\overline{p} \stackrel{a}{\Longrightarrow} \overline{p}'$$

$$\begin{array}{lcl}
(\overline{p} \quad \overline{q} \text{ und } \neg(\overline{p} \Uparrow a)) & \Rightarrow & \exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \ \wedge\ \overline{p}' \quad \overline{q}' \\
(\overline{q} \quad \overline{r} \text{ und } \neg(\overline{q} \Uparrow a) \text{ (vgl. ad (a))}) & \Rightarrow & \exists\, \overline{q}'.\ (\exists\, \overline{r}'.\ (\overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ \overline{q}' \quad \overline{r}') \ \wedge\ \overline{p}' \quad \overline{q}') \\
 & \Rightarrow & \exists\, \overline{r}'.\ (\overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ \exists\, \overline{q}'.\ (\overline{p}' \quad \overline{q}' \ \wedge\ \overline{q}' \quad \overline{r}')) \\
(\text{Definition von } R) & \Rightarrow & \exists\, \overline{r}'.\ \overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ (\overline{p}', \overline{r}') \in R
\end{array}$$

ad (c):

$$\overline{r} \stackrel{a}{\Longrightarrow} \overline{r}'$$

$$\begin{array}{lcl}
(\overline{q} \quad \overline{r} \text{ und } \neg(\overline{q} \Uparrow a) \text{ (vgl. ad (a))}) & \Rightarrow & \exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \ \wedge\ \overline{q}' \quad \overline{r}' \\
(\overline{p} \quad \overline{q} \text{ und } \neg(\overline{p} \Uparrow a))) & \Rightarrow & \exists\, \overline{q}'.\ (\exists\, \overline{p}'.\ (\overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ \overline{p}' \quad \overline{q}') \ \wedge\ \overline{q}' \quad \overline{r}') \\
 & \Rightarrow & \exists\, \overline{p}'.\ (\overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ \exists\, \overline{q}'.\ (\overline{p}' \quad \overline{q}' \ \wedge\ \overline{q}' \quad \overline{r}')) \\
(\text{Definition von } R) & \Rightarrow & \exists\, \overline{p}'.\ \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ (\overline{p}', \overline{r}') \in R
\end{array}$$

2. **Reflexivität**

   Z.zg.: $p \quad p$
   Beweis: Sei $R =_{df} \{(\overline{p}, \overline{p}) \mid \overline{p} \in S_p\}$ und $(\overline{p}, \overline{p}) \in R$ sowie $a \in \mathcal{A}$ beliebig. Ferner gelte $\neg(\overline{p} \Uparrow a)$. Trivialerweise gilt:

   - $\neg(\overline{p} \Uparrow a)$ impliziert $\neg(\overline{p} \Uparrow a)$ und
   - $\overline{p} \stackrel{a}{\Longrightarrow} \overline{p}'$ impliziert $\exists\, \overline{\overline{p}} \equiv \overline{p}'.\ \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ (\overline{p}', \overline{p}') \in R$.

   Folglich gilt $R \subseteq$ und wegen $(p, p) \in R$ auch $p \quad p$.

Aus (1) und (2) folgt mit Definition A.2, daß eine Quasiordnung ist.

Beachte, daß keine Halbordnung ist, denn:

Beispiel 1.18

ist nicht antisymmetrisch und daher keine Halbordnung. Betrachte dazu die Prozesse p und q mit $\mathcal{A}_p = \mathcal{A}_q = \{a\}$ aus Abbildung 1.4.

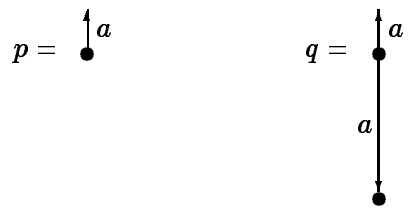


Abbildung 1.4: (Gegen–)Beispiel

Offensichtlich gilt $p \quad q$ und $q \quad p$, da beide Prozesse a–undefiniert sind. Jedoch gilt nicht $p \approx^d q$, weil q eine a–Transition besitzt, die von p nicht „gematcht" (simuliert) werden kann.

Eigenschaften von Äquivalenzrelation und Quasiordnung

Vergleicht man die Definitionen von $\approx^d$, $\approx$ und , so stellt man fest, daß $\approx$ und ähnliche Semantiken induzieren und $\approx^d$ eine Verfeinerung von beiden darstellt.

Proposition 1.19
Es gilt für alle Prozesse $p, q \in$ Processes:

1. *$p \approx^d q$ impliziert $p \approx q$ und $p \quad q$.*
2. *Für total definierte Prozesse stimmen $\approx^d$, $\approx$ und überein.*

Von großer Wichtigkeit ist die Kompositionalität von $\approx^d$ und , d.h., daß die Operatoren $\|$ und $\langle\cdot\rangle$ diese Relationen respektieren.

Satz 1.20 (Kompositionalität)
Für alle Prozesse $p, q, r \in$ Processes und alle Mengen L beobachtbarer Aktionen gilt:

1. *$p \quad q$ impliziert $p\|r \quad q\|r$.*
2. *$p \approx^d q$ impliziert $p\|r \approx^d q\|r$.*
3. *$p \quad q$ impliziert $p\langle L\rangle \quad q\langle L\rangle$.*
4. *$p \approx^d q$ impliziert $p\langle L\rangle \approx^d q\langle L\rangle$.*

Beweis
Seien $p, q, r \in$ Processes beliebig und L eine beliebige Menge beobachtbarer Aktionen, sowie $\mathcal{A} =_{df} \mathcal{A}_p = \mathcal{A}_q$.[6]

[6] Beachte dabei, daß $p \quad q$ bzw. $p \approx^d q$ impliziert, daß die Prozesse p und q dasselbe Kommunikationspotential ($\mathcal{A}_p = \mathcal{A}_q$) besitzen.

1. Gelte $p \quad q$ und $R =_{df} \{(\overline{p}\|\overline{r}, \overline{q}\|\overline{r}) \mid \overline{p} \quad \overline{q},\ \overline{p}\|\overline{r} \in S_{p\|r},\ \overline{q}\|\overline{r} \in S_{q\|r}\}$. Es genügt nach Definition 1.15 für $a \in \mathcal{A}$ und $(\overline{p}\|\overline{r}, \overline{q}\|\overline{r}) \in R$ unter der Voraussetzung $\neg((\overline{p}\|\overline{r}) \Uparrow a)$ folgendes zu zeigen:

   (a) $\neg((\overline{q}\|\overline{r}) \Uparrow a)$,

   (b) $\overline{p}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{p}'\|\overline{r}'$ impliziert $\exists\, \overline{q}'\|\overline{r}'.\ \overline{q}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{q}'\|\overline{r}' \ \wedge\ (\overline{p}'\|\overline{r}', \overline{q}'\|\overline{r}') \in R$ und

   (c) $\overline{q}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{q}'\|\overline{r}'$ impliziert $\exists\, \overline{p}'\|\overline{r}'.\ \overline{p}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{p}'\|\overline{r}' \ \wedge\ (\overline{p}'\|\overline{r}', \overline{q}'\|\overline{r}') \in R$.[7]

   Dann gilt $R \subseteq \quad$ und folglich wegen $(p\|r, q\|r) \in R$ auch $p\|r \quad q\|r$.

   ad (a): Es gilt $\neg((\overline{p}\|\overline{r}) \Uparrow a)$.

   Betrachte nun eine vollständige Fallunterscheidung gemäß der operationellen Semantik (vgl. Definition 1.8):

Fall 1:		$\neg(\overline{p} \Uparrow a) \ \wedge\ \neg(\overline{r} \Uparrow a)$
$(\overline{p} \quad \overline{q})$	$\Rightarrow$	$\neg(\overline{q} \Uparrow a) \ \wedge\ \neg(\overline{r} \Uparrow a)$
(vgl. Definition 1.8)	$\Rightarrow$	$\neg((\overline{q}\|\overline{r}) \Uparrow a)$

Fall 2:[8]		$\neg(\overline{p} \Uparrow a) \ \wedge\ \neg(\overline{p} \stackrel{a}{\Longrightarrow}) \ \wedge\ a \in \mathcal{A}_p \ \wedge\ \overline{r} \Uparrow a$
$(\overline{p} \quad \overline{q})$	$\Rightarrow$	$\neg(\overline{q} \Uparrow a) \ \wedge\ \neg(\overline{q} \stackrel{a}{\Longrightarrow}) \ \wedge\ a \in \mathcal{A}_q \ \wedge\ \overline{r} \Uparrow a$
(vgl. Definition 1.8)	$\Rightarrow$	$\neg((\overline{q}\|\overline{r}) \Uparrow a)$

Fall 3:[9]		$\neg(\overline{r} \Uparrow a) \ \wedge\ \neg(\overline{r} \stackrel{a}{\Longrightarrow}) \ \wedge\ a \in \mathcal{A}_r \ \wedge\ \overline{p} \Uparrow a$
Fall 3a: $\overline{q} \Uparrow a$	$\Rightarrow$	$\neg(\overline{r} \Uparrow a) \ \wedge\ \neg(\overline{r} \stackrel{a}{\Longrightarrow}) \ \wedge\ a \in \mathcal{A}_r \ \wedge\ \overline{q} \Uparrow a$
(vgl. Definition 1.8)	$\Rightarrow$	$\neg((\overline{q}\|\overline{r}) \Uparrow a)$
Fall 3b: $\neg(\overline{q} \Uparrow a)$	$\Rightarrow$	$\neg(\overline{r} \Uparrow a) \ \wedge\ \neg(\overline{q} \Uparrow a)$
(vgl. Definition 1.8)	$\Rightarrow$	$\neg((\overline{q}\|\overline{r}) \Uparrow a)$

[7] Beachte, daß $\exists\, \overline{q}'\|\overline{r}'$ bzw. $\exists\, \overline{p}'\|\overline{r}'$ als Metaschreibweise zu verstehen ist, da $\overline{r}'$ jeweils auf den linken Seiten der Implikationen festgelegt wurde.

[8] $\overline{p}$ verhindert die Undefiniertheit von $\overline{p}\|\overline{r}$.

ad (b): Gelte $\overline{p}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{p}'\|\overline{r}'$.

Fallunterscheidung gemäß der operationellen Semantik (vgl. Definition 1.8):

Regel (3):		$\overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ \overline{r} = \overline{r}' \ \wedge\ a \in (\mathcal{A}_p \setminus \mathcal{A}_r) \cup \{\tau\}$
$(\overline{p} \quad \overline{q})$	$\Rightarrow$	$(\exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \ \wedge\ \overline{p}' \quad \overline{q}') \ \wedge\ \overline{r} = \overline{r}' \ \wedge\ a \in (\mathcal{A}_q \setminus \mathcal{A}_r) \cup \{\tau\}$
(Regel (3) & Def. R)	$\Rightarrow$	$\exists\, \overline{q}'\|\overline{r}'.\ \overline{q}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{q}'\|\overline{r}' \ \wedge\ (\overline{p}'\|\overline{r}', \overline{q}'\|\overline{r}') \in R$

Regel (4):		$\overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ \overline{p} = \overline{p}' \ \wedge\ a \in (\mathcal{A}_r \setminus \mathcal{A}_p) \cup \{\tau\}$
$(\overline{p} \quad \overline{q})$	$\Rightarrow$	$\overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ \overline{q} = \overline{q}' \ \wedge\ a \in (\mathcal{A}_r \setminus \mathcal{A}_q) \cup \{\tau\}$
(Regel (4) & Def. von R)	$\Rightarrow$	$\exists\, \overline{q}'\|\overline{r}'.\ \overline{q}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{q}'\|\overline{r}' \ \wedge\ (\overline{p}'\|\overline{r}', \overline{q}'\|\overline{r}') \in R$

Regel (5):		$\overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ \overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ a \in (\mathcal{A}_p \cap \mathcal{A}_r)$
$(\overline{p} \quad \overline{q})$	$\Rightarrow$	$(\exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \ \wedge\ \overline{p}' \quad \overline{q}') \ \wedge\ \overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ a \in (\mathcal{A}_q \cap \mathcal{A}_r)$
(Regel (5) & Def. von R)	$\Rightarrow$	$\exists\, \overline{q}'\|\overline{r}'.\ \overline{q}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{q}'\|\overline{r}' \ \wedge\ (\overline{p}'\|\overline{r}', \overline{q}'\|\overline{r}') \in R$

ad (c): Analog zu (b) mit vertauschten Rollen von $\overline{p}$ und $\overline{q}$.

2. Gelte $p \approx^d q$ und $R =_{df} \{(\overline{p}\|\overline{r}, \overline{q}\|\overline{r}) \mid \overline{p} \approx^d \overline{q},\ \ \overline{p}\|\overline{r} \in S_{p\|r},\ \overline{q}\|\overline{r} \in S_{q\|r}\}$. Es genügt nach Definition 1.12, für $a \in \mathcal{A}$ und $(\overline{p}\|\overline{r}, \overline{q}\|\overline{r}) \in R$ beliebig zu zeigen:

   (a) $(\overline{p}\|\overline{r}) \Uparrow a \iff (\overline{q}\|\overline{r}) \Uparrow a$,

   (b) $\overline{p}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{p}'\|\overline{r}'$ impliziert $\exists\, \overline{q}'\|\overline{r}'.\ \overline{q}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{q}'\|\overline{r}' \ \wedge\ (\overline{p}'\|\overline{r}', \overline{q}'\|\overline{r}') \in R$ und

   (c) $\overline{q}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{q}'\|\overline{r}'$ impliziert $\exists\, \overline{p}'\|\overline{r}'.\ \overline{p}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{p}'\|\overline{r}' \ \wedge\ (\overline{p}'\|\overline{r}', \overline{q}'\|\overline{r}') \in R$.

   Dann gilt $R \subseteq \approx^d$ und folglich wegen $(p\|r, q\|r) \in R$ auch $p\|r \approx^d q\|r$.

   ad (a): Beweis der Richtung „ $\Rightarrow$ “. Die andere Beweisrichtung ist analog mit vertauschten Rollen von $\overline{p}$ und $\overline{q}$.

[9] $\overline{r}$ verhindert die Undefiniertheit von $\overline{p}\|\overline{r}$.

Gelte nun $(\overline{p}\|\overline{r}) \Uparrow a$. Fallunterscheidung gemäß der operationellen Semantik (vgl. Definition 1.8):

$$
\begin{array}{lll}
\text{Regel (8):} & & \overline{p} \Uparrow a \ \wedge \ a \notin \mathcal{A}_r \\
(\overline{p} \approx^d \overline{q}) & \Rightarrow & \overline{q} \Uparrow a \ \wedge \ a \notin \mathcal{A}_r \\
(\text{Regel (8)}) & \Rightarrow & (\overline{q}\|\overline{r}) \Uparrow a
\end{array}
$$

$$
\begin{array}{lll}
\text{Regel (9):} & & \overline{p} \Uparrow a \ \wedge \ \overline{r} \stackrel{a}{\Longrightarrow} \\
(\overline{p} \approx^d \overline{q}) & \Rightarrow & \overline{q} \Uparrow a \ \wedge \ \overline{r} \stackrel{a}{\Longrightarrow} \\
(\text{Regel (9)}) & \Rightarrow & (\overline{q}\|\overline{r}) \Uparrow a
\end{array}
$$

$$
\begin{array}{lll}
\text{Regel (10):} & & \overline{r} \Uparrow a \ \wedge \ a \notin \mathcal{A}_p \\
(\mathcal{A}_p = \mathcal{A}_q) & \Rightarrow & \overline{r} \Uparrow a \ \wedge \ a \notin \mathcal{A}_q \\
(\text{Regel (10)}) & \Rightarrow & (\overline{q}\|\overline{r}) \Uparrow a
\end{array}
$$

$$
\begin{array}{lll}
\text{Regel (11):} & & \overline{r} \Uparrow a \ \wedge \ \overline{p} \stackrel{a}{\Longrightarrow} \\
(\overline{p} \approx^d \overline{q}) & \Rightarrow & \overline{r} \Uparrow a \ \wedge \ \overline{q} \stackrel{a}{\Longrightarrow} \\
(\text{Regel (11)}) & \Rightarrow & (\overline{q}\|\overline{r}) \Uparrow a
\end{array}
$$

$$
\begin{array}{lll}
\text{Regel (12):} & & \overline{p} \Uparrow a \ \wedge \ \overline{r} \Uparrow a \\
(\overline{p} \approx^d \overline{q}) & \Rightarrow & \overline{q} \Uparrow a \ \wedge \ \overline{r} \Uparrow a \\
(\text{Regel (12)}) & \Rightarrow & (\overline{q}\|\overline{r}) \Uparrow a
\end{array}
$$

ad (b): Gelte $\overline{p}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{p}'\|\overline{r}'$.

Fallunterscheidung gemäß der operationellen Semantik (vgl. Definition 1.8):

Regel (3):		$\overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ \overline{r} = \overline{r}' \ \wedge\ a \in (\mathcal{A}_p \setminus \mathcal{A}_r) \cup \{\tau\}$
$(\overline{p} \approx^d \overline{q})$	$\Rightarrow$	$(\exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \wedge \overline{p}' \approx^d \overline{q}') \wedge \overline{r} = \overline{r}' \ \wedge\ a \in (\mathcal{A}_q \setminus \mathcal{A}_r) \cup \{\tau\}$
(Regel (3) & Def. R)	$\Rightarrow$	$\exists\, \overline{q}'\|\overline{r}'.\ \overline{q}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{q}'\|\overline{r}' \ \wedge\ (\overline{p}'\|\overline{r}', \overline{q}'\|\overline{r}') \in R$
Regel (4):		$\overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ \overline{p} = \overline{p}' \ \wedge\ a \in (\mathcal{A}_r \setminus \mathcal{A}_p) \cup \{\tau\}$
$(\overline{p} \approx^d \overline{q})$	$\Rightarrow$	$\overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ \overline{q} = \overline{q}' \ \wedge\ a \in (\mathcal{A}_r \setminus \mathcal{A}_q) \cup \{\tau\}$
(Regel (4) & Def. R)	$\Rightarrow$	$\exists\, \overline{q}'\|\overline{r}'.\ \overline{q}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{q}'\|\overline{r}' \ \wedge\ (\overline{p}'\|\overline{r}', \overline{q}'\|\overline{r}') \in R$
Regel (5):		$\overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ \overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ a \in (\mathcal{A}_p \cap \mathcal{A}_r)$
$(\overline{p} \approx^d \overline{q})$	$\Rightarrow$	$(\exists \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \ \wedge\ \overline{p}' \approx^d \overline{q}') \ \wedge\ \overline{r} \stackrel{a}{\Longrightarrow} \overline{r}' \ \wedge\ a \in (\mathcal{A}_q \cap \mathcal{A}_r)$
(Regel (5) & Def. R)	$\Rightarrow$	$\exists\, \overline{q}'\|\overline{r}'.\ \overline{q}\|\overline{r} \stackrel{a}{\Longrightarrow} \overline{q}'\|\overline{r}' \ \wedge\ (\overline{p}'\|\overline{r}', \overline{q}'\|\overline{r}') \in R$

ad (c): Analog zu (b) mit vertauschten Rollen von $\overline{p}$ und $\overline{q}$.

3. Gelte $p \quad q$ und $R =_{df} \{(\overline{p}\langle L\rangle, \overline{q}\langle L\rangle) \mid \overline{p} \quad \overline{q}, \overline{p} \in S_p, \overline{q} \in S_q\}$. Sei $(\overline{p}\langle L\rangle, \overline{q}\langle L\rangle) \in R$ und $a \in \mathcal{A}$ beliebig. Unter der Voraussetzung $\neg(\overline{p}\langle L\rangle \Uparrow a)$ ist gemäß Definition 1.15 zu zeigen:

   (a) $\neg(\overline{q}\langle L\rangle \Uparrow a)$,

   (b) $\overline{p}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{p}'\langle L\rangle$ impliziert $\exists\, \overline{q}'\langle L\rangle.\ \overline{q}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{q}'\langle L\rangle \ \wedge\ (\overline{p}'\langle L\rangle, \overline{q}'\langle L\rangle) \in R$ und

   (c) $\overline{q}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{q}'\langle L\rangle$ impliziert $\exists\, \overline{p}'\langle L\rangle.\ \overline{p}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{p}'\langle L\rangle \ \wedge\ (\overline{p}'\langle L\rangle, \overline{q}'\langle L\rangle) \in R$.

   Dann gilt $R \subseteq \quad$ und folglich wegen $(p\langle L\rangle, q\langle L\rangle) \in R$ auch $p\langle L\rangle \quad q\langle L\rangle$.

ad (a):

$$\begin{array}{lcl} & & \neg(\overline{p}\langle L\rangle \Uparrow a) \\ (\text{insbesondere } a \in L, \text{ Regel (6)}) & \Rightarrow & \neg(\overline{p} \Uparrow a) \\ (\overline{p} \quad \overline{q}) & \Rightarrow & \neg(\overline{q} \Uparrow a) \\ (a \in L, \text{ Regel (6)}) & \Rightarrow & \neg(\overline{q}\langle L\rangle \Uparrow a) \end{array}$$

ad (b):

$$\begin{array}{lcl} & & \overline{p}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{p}'\langle L\rangle \\ (\text{insbesondere } a \in L, \text{ Regel (1)}) & \Rightarrow & \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \\ (\overline{p} \quad \overline{q} \text{ und } \neg(\overline{p}\langle L\rangle \Uparrow a)) & \Rightarrow & \exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \ \wedge\ \overline{p}' \quad \overline{q}' \\ (a \in L, \text{ Regel (1), Def. } R) & \Rightarrow & \exists\, \overline{q}'\langle L\rangle.\ \overline{q}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{q}'\langle L\rangle \ \wedge\ (\overline{p}'\langle L\rangle, \overline{q}'\langle L\rangle) \in R \end{array}$$

ad (c):

$$\begin{array}{lcl} & & \overline{q}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{q}'\langle L\rangle \\ (\text{insbesondere } a \in L, \text{ Regel (1)}) & \Rightarrow & \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \\ (\overline{p} \quad \overline{q} \text{ und } \neg(\overline{p}\langle L\rangle \Uparrow a)) & \Rightarrow & \exists\, \overline{p}'.\ \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ \overline{p}' \quad \overline{q}' \\ (a \in L, \text{ Regel (1), Def. } R) & \Rightarrow & \exists\, \overline{p}'\langle L\rangle.\ \overline{p}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{p}'\langle L\rangle \ \wedge\ (\overline{p}'\langle L\rangle, \overline{q}'\langle L\rangle) \in R \end{array}$$

4. Gelte $p \approx^d q$ und $R =_{df} \{(\overline{p}\langle L\rangle, \overline{q}\langle L\rangle) \mid \overline{p} \approx^d \overline{q},\ \ \overline{p} \in S_p,\ \overline{q} \in S_q\}$. Dann genügt es nach Definition 1.12, für $a \in \mathcal{A}$ und $(\overline{p}\langle L\rangle, \overline{q}\langle L\rangle) \in R$ beliebig zu zeigen:

   (a) $\overline{p}\langle L\rangle \Uparrow a \iff \overline{q}\langle L\rangle \Uparrow a$,

   (b) $\overline{p}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{p}'\langle L\rangle$ impliziert $\exists\, \overline{q}'\langle L\rangle.\ \overline{q}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{q}'\langle L\rangle \ \wedge\ (\overline{p}'\langle L\rangle, \overline{q}'\langle L\rangle) \in R$ und

   (c) $\overline{q}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{q}'\langle L\rangle$ impliziert $\exists\, \overline{p}'\langle L\rangle.\ \overline{p}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{p}'\langle L\rangle \ \wedge\ (\overline{p}'\langle L\rangle, \overline{q}'\langle L\rangle) \in R$.

   Dann gilt $R \subseteq \approx^d$ und folglich wegen $(p\langle L\rangle, q\langle L\rangle) \in R$ auch $p\langle L\rangle \approx^d q\langle L\rangle$.

ad (a):

$$\begin{array}{lcl} & & \overline{p}\langle L\rangle \Uparrow a \\ (a \in L, \text{ Regel } (6)) & \Longleftrightarrow & \overline{p} \Uparrow a \\ (\overline{p} \approx^d \overline{q}) & \Longleftrightarrow & \overline{q} \Uparrow a \\ (a \in L, \text{ Regel } (6)) & \Longleftrightarrow & \overline{q}\langle L\rangle \Uparrow a \end{array}$$

ad (b):

$$\begin{array}{lcl} & & \overline{p}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{p}'\langle L\rangle \\ (\text{insbesondere } a \in L, \text{ Regel } (1)) & \Rightarrow & \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \\ (\overline{p} \approx^d \overline{q}) & \Rightarrow & \exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \ \wedge\ \overline{p}' \approx^d \overline{q}' \\ (a \in L, \text{ Regel } (1), \text{ Def. } R) & \Rightarrow & \exists\, \overline{q}'\langle L\rangle.\ \overline{q}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{q}'\langle L\rangle \ \wedge\ (\overline{p}'\langle L\rangle, \overline{q}'\langle L\rangle) \in R \end{array}$$

ad (c):

$$\begin{array}{lcl} & & \overline{q}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{q}'\langle L\rangle \\ (\text{insbesondere } a \in L, \text{ Regel } (1)) & \Rightarrow & \overline{q} \stackrel{a}{\Longrightarrow} \overline{q}' \\ (\overline{p} \approx^d \overline{q}) & \Rightarrow & \exists\, \overline{p}'.\ \overline{p} \stackrel{a}{\Longrightarrow} \overline{p}' \ \wedge\ \overline{p}' \approx^d \overline{q}' \\ (a \in L, \text{ Regel } (1), \text{ Def. } R) & \Rightarrow & \exists\, \overline{p}'\langle L\rangle.\ \overline{p}\langle L\rangle \stackrel{a}{\Longrightarrow} \overline{p}'\langle L\rangle \ \wedge\ (\overline{p}'\langle L\rangle, \overline{q}'\langle L\rangle) \in R \end{array}$$

Quasiordnung bzw. semantische Äquivalenz und Sprachen

Den Zusammenhang zwischen den Begriffen *Quasiordnung* bzw. semantischer Äquivalenz mit *Sprachen von Prozessen* stellen die beiden folgenden Lemmata her.

Lemma 1.21
Für beliebige Prozesse $p, q \in$ Processes gilt:

$$p \quad q \text{ impliziert } \mathcal{L}(p) \supseteq \mathcal{L}(q).$$

Gilt insbesondere $p \quad q$ und $q \quad p$, so folgt $\mathcal{L}(p) = \mathcal{L}(q)$.

Beweisskizze
Gilt $p \quad q$, so ist p intuitiv „mehr undefiniert" als q. Von einem undefinierten Zustand aus ist aber nach Definition 1.7 die ganze Sprache $\mathcal{A}_p^*$ erkennbar. Daher gilt $\mathcal{L}(p) \supseteq \mathcal{L}(q)$.

Lemma 1.22
Für beliebige Prozesse $p, q \in$ Processes gilt:

$$p \approx^d q \text{ impliziert } \mathcal{L}(p) = \mathcal{L}(q).$$

Beweis
Seien $p, q \in$ Processes mit $p \approx^d q$ beliebig. Gemäß Proposition 1.19 gilt dann $p \quad q$ sowie $q \quad p$. Mit Lemma 1.21 folgt schließlich die Behauptung $\mathcal{L}(p) = \mathcal{L}(q)$.

1.3 Schnittstellen

In diesem Abschnitt steht der Begriff *Interfacespezifikation* im Vordergrund. Interfacespezifikationen sind für die $\mathcal{RM}$–Methode das zentrale Hilfsmittel. Mit ihrer Hilfe kann man von einer konkreten Darstellung von Parallelkontexten abstrahieren und die Charakteristika der betrachteten Parallelkontexte hervorheben.

Interfacespezifikationen

Eine Interfacespezifikation ist ein total definierter Prozeß, der den *Kommunikationsfluß* zwischen zwei Prozessen beschreibt. Der Kommunikationsfluß zwischen zwei Prozessen ist die Menge aller beobachtbarer Aktionensequenzen, die die betrachtete Schnittstelle passieren können. Deswegen wird eine *exakte Interfacespezifikation* zwischen den Prozessen p und q als ein Prozeß definiert, dessen Sprache mit der des Prozesses $(p \| q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle$ übereinstimmt; denn im Parallelkontext kommunizieren die Prozesse, wie in Definition 1.8 festgelegt, über gemeinsame Aktionen. Eine *korrekte Interfacespezifikation* ist dann ein total definierter Prozeß, dessen Sprache die der exakten Interfacespezifikation umfaßt.

Definition 1.23 (Interfacespezifikationen)
Seien $p, q \in$ Processes. Dann definiere:

1. *Ein total definierter Prozeß I heißt Interfacespezifikation für p genau dann, wenn $\mathcal{A}_I \subseteq \mathcal{A}_p$ und $\tau \notin \mathcal{A}_I$ gilt. I heißt Interfacespezifikation für p und q genau dann, wenn $\mathcal{A}_I = \mathcal{A}_p \cap \mathcal{A}_q$ und $\tau \notin \mathcal{A}_I$.*

2. *Eine Interfacespezifikation I für p und q heißt exakte Interfacespezifikation für p und q, falls $\mathcal{L}(I) = \mathcal{L}((p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle)$*

3. *Eine Interfacespezifikation I für p und q heißt korrekt für p und q genau dann, wenn $\mathcal{L}((p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle) \subseteq \mathcal{L}(I)$ gilt.*

Die Menge aller Schnittstellen wird mit Interfaces, die aller Interfacespezifikationen für p mit $\mathcal{I}(p)$ und die aller korrekten Interfacespezifikationen für p und q mit $\mathcal{I}(p,q)$ bezeichnet.

Insbesondere ist nach dieser Definition eine exakte Interfacespezifikation auch korrekt. Daß es im Rahmen dieser Arbeit sinnvoll ist, die Korrektheit einer Interfacespezifikation über ihre Sprache zu definieren, wird im Kapitel 2 am theoretischen Resultat der *Repräsentationsunabhängigkeit* deutlich.

Interfacespezifikationen für das begleitende Beispiel

Die Interfacespezifikationen I_1 und I_2 für das begleitende Beispiel sind gemäß Figur 1.5 definiert.

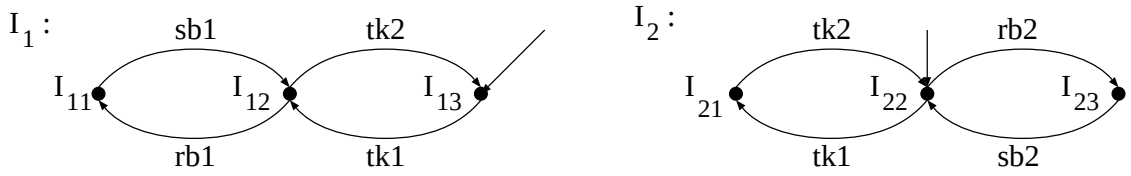


Abbildung 1.5: Interfacespezifikationen

Die mit Definition von I_1 verbundene Intuition ergibt sich durch die Vorstellung der Kommunikationssequenzen auf den die Schnittstelle betreffenden Kanälen (am Beispiel von I_1 sind dies $tk1, tk2, rb1, sb1$): Zunächst wird über $tk1$ das Token erwartet, bevor über $rb1$ bzw. $sb1$ Daten empfangen bzw. gesendet werden und schließlich das Token über $tk2$ weitergegeben wird. Damit würde man als Interfacespezifikation I_1 einen Prozeß erhalten, der mit P_1 identisch ist und eine exakte Interfacespezifikation wäre. Da eine korrekte Interfacespezifikation auch eine Obermenge der entsprechenden Sprache beschreiben darf, ist die gegebene Definition von I_1, die quasi durch das Zusammenfassen der Zustände p_{12} und p_{14} von P_1 entsteht, korrekt. Die zu I_1 gehörige Sprache ist damit größer als diejenige, die die Intuition vorgibt. Eine analoge Argumentation gilt für die Definition von I_2.

I_1 ist eine Interfacespezifikation für P_1 und $B\|P_2$, denn es gilt:

$$\mathcal{A}_{I_1} = \{tk1, tk2, rb1, sb1\} = \mathcal{A}_{P_1} \cap \mathcal{A}_{B\|P_2} \text{ und } \tau \notin \mathcal{A}_{I_1}.$$

Ebenso ist I_2 eine Interfacespezifikation für $P_1 \| B$ und P_2, denn es gilt:

$$\mathcal{A}_{I_2} = \{tk1, tk2, rb2, sb2\} = \mathcal{A}_{P_1 \| B} \cap \mathcal{A}_{P_2} \text{ und } \tau \notin \mathcal{A}_{I_2}.$$

Beide Interfacespezifikationen sind auch korrekt.

Die Sprache der Interfacespezifikationen I_1 und I_2 erhält man gemäß Definition 1.7 aus folgenden Gleichungssystemen:

$$\begin{array}{lcl}
\mathcal{L}(I_1) & = & \mathcal{L}(I_{13}) = tk1 \cdot \mathcal{L}(I_{12}) \cup \{\epsilon\} \\
\mathcal{L}(I_{12}) & = & rb1 \cdot \mathcal{L}(I_{11}) \cup tk2 \cdot \mathcal{L}(I_{13}) \cup \{\epsilon\} \\
\mathcal{L}(I_{11}) & = & sb1 \cdot \mathcal{L}(I_{12}) \cup \{\epsilon\} \\
 & & \\
\mathcal{L}(I_2) & = & \mathcal{L}(I_{22}) = tk1 \cdot tk2 \cdot \mathcal{L}(I_{22}) \cup rb2 \cdot sb2 \cdot \mathcal{L}(I_{22}) \cup \{tk1 \cdot \epsilon\} \cup \{rb2 \cdot \epsilon\} \cup \{\epsilon\}
\end{array}$$

Da später jeweils nur die Präfixe der Länge eins der Sprachen von I_1 und I_2 benötigt werden, ist ein explizites Ausrechnen von $\mathcal{L}(I_1)$ bzw. $\mathcal{L}(I_2)$ nicht erforderlich.

Eigenschaften von Interfacespezifikationen

Die Frage, wie sich die durch die Quasiordnung induzierte Semantik auf korrekte Interfacespezifikationen auswirkt, beantwortet das folgende Lemma:

Lemma 1.24
Für beliebige Prozesse $p, p', q \in$ Processes gilt:

$$p \quad p' \text{ impliziert } \mathcal{I}(p,q) \subseteq \mathcal{I}(p',q).$$

Beweis
Seien $p, p', q \in$ Processes mit $p \quad p'$ beliebig, d.h. insbesondere $\mathcal{A}_p = \mathcal{A}_{p'}$. Wegen Satz 1.20 gilt

$$(p \| q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle \quad (p' \| q)\langle \mathcal{A}'_p \cap \mathcal{A}_q \rangle.$$

Mit Lemma 1.21 folgt

$$(*) \qquad \mathcal{L}((p \| q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle) \supseteq \mathcal{L}((p' \| q)\langle \mathcal{A}'_p \cap \mathcal{A}_q \rangle).$$

Sei nun $I \in \mathcal{I}(p,q)$ beliebig, aber fest. Dann folgt die Behauptung mit

$$
\begin{array}{lll}
 & & I \in \mathcal{I}(p,q) \\
(\text{Definition 1.23}) & \Rightarrow & \mathcal{L}(I) \supseteq \mathcal{L}((p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle) \\
(*) & \Rightarrow & \mathcal{L}(I) \supseteq \mathcal{L}((p'\|q)\langle \mathcal{A}_{p'} \cap \mathcal{A}_q \rangle) \\
(\text{Definition 1.23}) & \Rightarrow & I \in \mathcal{I}(p',q).
\end{array}
$$

Lemma 1.25
Für beliebige Prozesse $p, p', q \in$ <u>Processes</u> mit $p \approx^d p'$ gilt: $\mathcal{I}(p,q) = \mathcal{I}(p',q)$.

Beweis
Seien $p, p', q \in$ <u>Processes</u> mit $p \approx^d p'$ beliebig. Gemäß Proposition 1.19 gilt dann $p \quad p'$ sowie $p' \quad p$. Mit Lemma 1.24 folgt unmittelbar die Behauptung $\mathcal{I}(p,q) = \mathcal{I}(p',q)$.

Für den Beweis der Korrektheit der $\mathcal{RM}$–Methode zur kompositionellen Minimierung endlicher verteilter Systeme wird die nachstehende Beziehung wichtig sein.

Lemma 1.26
Seien $p, q \in$ <u>Processes</u> und L eine beliebige Menge von Aktionen. Dann gilt:

$$\mathcal{I}(p,q) = \mathcal{I}(p\langle (\mathcal{A}_p \cap \mathcal{A}_q) \cup L \rangle, q).$$

Beweis
Seien $p, q \in$ <u>Processes</u> beliebig und L eine beliebige Menge von Aktionen. Die Behauptung folgt mit:

$$
\begin{array}{lll}
 & & I \in \mathcal{I}(p,q) \\
(\text{Definition 1.23}) & \Longleftrightarrow & \mathcal{L}(I) \supseteq \mathcal{L}((p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle) \\
(\text{Lemmata 1.10 \& 1.22}) & \Longleftrightarrow & \mathcal{L}(I) \supseteq \mathcal{L}((p\langle (\mathcal{A}_p \cap \mathcal{A}_q) \cup L \rangle \| q)\langle \mathcal{A}_p \cap \mathcal{A}_q \rangle) \\
(\text{Definition 1.23}) & \Longleftrightarrow & I \in \mathcal{I}(p\langle (\mathcal{A}_p \cap \mathcal{A}_q) \cup L \rangle, q).
\end{array}
$$

Kapitel 2

Der Reduktionsoperator

In diesem Kapitel wird das Kernstück der in dieser Arbeit zu entwickelnden $\mathcal{RM}$–Methode zur kompositionellen Minimierung endlicher verteilter Systeme vorgestellt, nämlich der Begriff des *Reduktionsoperators*.

Nach einer kurzen Motivation und Analyse der Problematik wird in Abschnitt 2.1 der Begriff eines *Reduktionsoperators* definiert. Anschließend wird ein spezieller Reduktionsoperator, bezeichnet mit $\overline{\Pi}$, vorgestellt und anhand eines kurzen Beispiels vorgeführt. $\overline{\Pi}$ wird aus zwei unterschiedlichen Perspektiven betrachtet: zum einen aus der theoretischen und zum anderen aus der algorithmischen Perspektive. In Abschnitt 2.2 stehen die theoretischen Eigenschaften von $\overline{\Pi}$ im Vordergrund, welche hauptsächlich beim Beweis, daß es sich bei $\overline{\Pi}$ tatsächlich um einen Reduktionsoperator handelt, behilflich sind. In Abschnitt 2.3 geht es um die Entwicklung eines korrekten und effektiven Algorithmus, der die Anwendung von $\overline{\Pi}$ realisiert. Dabei wird implizit das wichtige theoretische Resultat der *Repräsentationsunabhängigkeit* bewiesen.

In der Einleitung wurde bereits bemerkt, daß der naive Ansatz zur kompositionellen Minimierung von Transitionssystemen unzureichend ist. Dabei wird sukzessive die Funktion $\mathcal{M}$, welche beobachtungsäquivalente Zustände eines Transitionssystems zu jeweils einem Zustand verschmilzt, angewandt. Der Nachteil liegt darin, daß $\mathcal{M}$ nur den lokalen Kontext berücksichtigt. Dieser ist jeweils durch das momentan konstruierte Zwischentransitionssystem gegeben, welches die parallele Komposition der bereits betrachteten Parallelkomponenten darstellt. Unberücksichtigt hingegen bleiben globale Kontextabhängigkeiten, die Aufschluß darüber geben können, welche Zustände und Transitionen des bis jetzt konstruierten Zwischentransitionssystems bei Hinzunahme der restlichen Parallelkomponenten nicht mehr erreicht werden können. Solche Zustände und Transitionen kann man deshalb aus diesem Zwischentransitionssystem eliminieren. Dies wird die Aufgabe von Reduktionsoperatoren sein. Es verbleibt die Frage, wie man globale Kontextabhängigkeiten erkennen kann, ohne das gesamte Transitionssystem konstruieren zu müssen. Als Hilfsmittel dazu werden die im vorangegangenen Kapitel vorgestellten *Interfacespezifikationen*, genauer gesagt die Sprache dieser Prozesse, dienen. Dabei interessiert jeweils eine

Interfacespezifikation, welche die Schnittstelle zwischen den bereits bei der Konstruktion des Zwischentransitionssystems berücksichtigten und den restlichen Parallelkomponenten beschreibt. Zustände und Transitionen des momentanen Zwischentransitionssystems, die bzgl. der Schnittstellensprache unerreichbar sind, bleiben auch im globalen Kontext unerreichbar, sofern die zu Hilfe genommene Interfacespezifikation korrekt ist.

2.1 Grundlagen

Zunächst ist zu definieren, was unter einem Reduktionsoperator verstanden werden soll.

Definition 2.1 (Reduktionsoperatoren)
Eine Abbildung $\Pi : \underline{Interfaces} \times \underline{Processes} \dashrightarrow \underline{Processes}$ *heißt Reduktionsoperator, falls* Π *die folgenden Eigenschaften erfüllt:*[1]

(i) $\forall\, p \in \underline{Processes}\ \forall\, I \in \mathcal{I}(p).\quad \Pi(I,p) \quad p.$

(ii) $\forall\, p,q \in \underline{Processes}\ \forall\, I \in \mathcal{I}(p,q).\quad \Pi(I,p)\|q \approx^d p\|q.$

Statt $\Pi(I,p)$ *schreibt man auch* $\Pi_I(p)$.

Die Intuition für diese Definition ist die folgende:
Gemäß den einleitenden Überlegungen soll die Anwendung eines Reduktionsoperators Π auf einen gegebenen Prozeß p und eine Interfacespezifikation $I \in \mathcal{I}(p)$ die bzgl. des durch I beschriebenen Kontextes unerreichbaren Zustände und Transitionen von p eliminieren, so daß man den reduzierten Prozeß $\Pi_I(p)$ erhält. Forderung (i) stellt die Korrektheit einer solchen Reduktion sicher. Sie besagt, daß der reduzierte Prozeß $\Pi_I(p)$ in der Quasiordnung kleiner als p sein soll. Dadurch wird garantiert, daß die Argumente I und p durch Π nicht auf irgendeinen Prozeß abgebildet werden dürfen, sondern, daß sich der Prozeß $\Pi_I(p)$ an den Zuständen, an denen er definiert ist, genauso verhält wie p. Folglich ist es also insbesondere möglich, daß ein Reduktionsoperator Π neue Undefiniertheiten bzgl. eines Zustands hinzufügen darf, um beispielsweise zu markieren, welche vom betrachteten Zustand ausgehenden Transitionen eliminiert worden sind. Diese Technik führt offensichtlich zu einem bzgl. weniger definierten Prozeß und wird im Zusammenhang mit der Definition des speziellen Reduktionsoperators $\overline{\Pi}$ noch näher erläutert. Die Forderung (ii) obiger Definition betrifft die *Kontexttreue* der Reduktion: Die Anwendung von Π bzgl. eines Prozesses p und einer Interfacespezifikation $I \in \mathcal{I}(p)$, d.h. die Reduktion von p bzgl. I, im Parallelkontext mit einem Prozeß q heißt *kontexttreu*, falls $\Pi(I,p)\|q \approx^d p\|q$ gilt. Ist insbesondere $I \in \mathcal{I}(p,q)$, also eine korrekte Interfacespezifikation für die Prozesse p und q, so ist die Reduktion von p bzgl. I im Parallelkontext mit q wegen Forderung (ii) kontexttreu. Forderung (ii) ist sinnvoll, da ein Reduktionsoperator nur solche Zustände und Transitionen aus p verhindern soll, die im Parallelkontext mit q nicht erreicht werden können. Dort soll es nämlich bzgl. des Verhaltens des Gesamtsystems keinen Unterschied

[1] Das Zeichen $\dashrightarrow$ verdeutlicht, daß es sich bei Π um eine *partielle* Abbildung handelt.

machen, ob p oder $\Pi(I,p)$ berücksichtigt wird. Dies drückt die Forderung $\Pi(I,p)\|q \approx^d p\|q$ aus.

Es fällt auf, daß die Forderungen (i) und (ii) aus obiger Definition eher semantischer als technischer Natur sind. So ist der Begriff *Reduktions*operator auch zu verstehen: Er soll einen gegebenen Prozeß zu einem semantisch kleineren, aber im Kontext korrekten Prozeß reduzieren. Diese Sichtweise verdeutlicht nochmals die Notwendigkeit und den Sinn einer Quasiordnung in der vorliegenden Arbeit.

Um einen Reduktionsoperator im Rahmen einer praktischen Methode zur Vermeidung der Zustandsexplosion, wie der $\mathcal{RM}$–Methode, sinnvoll verwenden zu können, wird in dieser Arbeit folgende technische Bedingung in Definition 2.1 zusätzlich aufgenommen:

(iii) $\forall\, p \in \underline{\text{Processes}}\ \forall\, I \in \mathcal{I}(p). \quad |\, S_{\Pi(I,p)} \,| \leq |\, S_p \,| \quad \wedge \quad |\, \longrightarrow_{\Pi(I,p)} \,| \leq |\, \longrightarrow_p \,|.$

Der reduzierte Prozeß $\Pi_I(p)$ soll höchstens soviele Zustände und Transitionen wie der Prozeß p besitzen. Dies ergibt sich, wie in Beispiel 1.16 gesehen, *nicht* als Konsequenz aus Definition 2.1 (i).

Die nachstehende Proposition ist eine leichte Folgerung der Eigenschaften aus Definition 2.1:

Proposition 2.2
Sei Π ein beliebiger Reduktionsoperator. Dann gilt für alle $p, p', q \in \underline{\text{Processes}}$ und $I \in \mathcal{I}(p,q)$:

$$p \approx^d p' \ \textit{ impliziert } \ \Pi(I,p)\|q \approx^d \Pi(I,p')\|q$$

Beweis
Sei Π ein beliebiger Reduktionsoperator, sowie $p, p', q \in \underline{\text{Processes}}$ und $I \in \mathcal{I}(p,q)$ beliebig. Dann folgt die Behauptung mit:

$$\begin{array}{lcl} & & \Pi_I(p)\|q \\ (\text{Definition 2.1 (ii)}) & \approx^d & p\|q \\ (p \approx^d p' \ \&\ \text{Satz 1.20}) & \approx^d & p'\|q \\ (\text{Lemma 1.25 \& Definition 2.1 (i)}) & \approx^d & \Pi_I(p')\|q. \end{array}$$

Nun sind die Grundlagen gelegt, um einen konkreten Reduktionsoperator vorstellen zu können, welcher direkt der einleitend erwähnten Intuition entspricht und mit $\overline{\Pi}$ bezeichnet wird.
Die zentrale Idee der Definition von $\overline{\Pi}$ ist, für einen Prozeß $p \in \underline{\text{Processes}}$ und eine Interfacespezifikation $I \in \mathcal{I}(p)$ die Reduktion $\overline{\Pi}(I,p)$ durch die Projektion von $p\|I$ auf die erste Komponente zu definieren. Durch die Betrachtung von $p\|I$ kann $\overline{\Pi}(I,p)$ nur solche Transitionen von p ausführen bzw. solche Zustände von p erreichen, die die durch I spezifizierte Schnittstelle auch erlaubt. Die Projektion sichert, daß $\overline{\Pi}$ zusätzlich die technische Forderung (iii) aus Definition 2.1 erfüllt. Formal ist der Reduktionsoperator $\overline{\Pi}$ wie folgt definiert:

Definition 2.3 (Der Reduktionsoperator $\overline{\Pi}$)
Sei $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p) \in \underline{\text{Processes}}$ *und* $I \in \mathcal{I}(p)$.
Dann ist $\overline{\Pi} : \underline{\text{Interfaces}} \times \underline{\text{Processes}} \dashrightarrow \underline{\text{Processes}}$ *definiert durch*

$$(I,p) \longmapsto \overline{\Pi}(I,p) = \overline{\Pi}_I(p) = ((S, \longrightarrow, \mathcal{A} \cup \{\tau\}, \uparrow), p),$$

wobei

1. $S = \{q \in S_p \mid \exists\, i \in S_I.\ q\|i \in S_{p\|I}\}$,
2. $\mathcal{A} = \mathcal{A}_p$,
3. $\forall\, q, q' \in S\ \forall\, a \in \mathcal{A} \cup \{\tau\}.\ q \xrightarrow{a} q'$ *genau dann, wenn* $\exists\, i, i' \in S_I.\ q\|i \xrightarrow{a}_{p\|I} q'\|i'$,[2]
4. $\forall\, q \in S.\ q\uparrow\tau$ *genau dann, wenn* $q\uparrow_p\tau$ *und*
5. $\forall\, q \in S\ \forall\, a \in \mathcal{A}.\ q\uparrow a$ *genau dann, wenn eine der beiden folgenden Bedingungen gilt:*
   (a) $q\uparrow_p a$ *oder*
   (b) $\exists\, q' \in S_p.\ q \xrightarrow{a}_p q'\ \wedge\ \nexists\, q' \in S.\ q \xrightarrow{a} q'$.

Es bleibt natürlich zu zeigen, daß $\overline{\Pi}$ ein Reduktionsoperator im Sinne von Definition 2.1 ist. Der Beweis dazu steht im Mittelpunkt des nächsten Abschnitts. Doch zuvor sei der Blick noch einmal auf Definition 2.3 gerichtet.
Der einzige Unterschied zwischen $\overline{\Pi}_I(p)$ und der Projektion von $p\|I$ auf die erste Komponente betrifft die Undefiniertheiten: $\overline{\Pi}_I(p)$ erbt alle Undefiniertheiten von p und erhält an solchen Zuständen zusätzliche a–Undefiniertheiten, an denen eine a–Transition aufgrund der Interfacespezifikation I „abgeschnitten“ (eliminiert) worden ist. Zentral hierbei ist, daß letztere Undefiniertheiten im globalen Kontext $\overline{\Pi}_I(p)\|q$ für eine korrekte Interfacespezifikation I für die Prozesse p und q wieder verschwinden, denn: Ist eine a–Transition von p durch eine a–Undefiniertheit $\uparrow a$ ersetzt worden, so verschwindet diese im globalen Kontext $\overline{\Pi}_I(p)\|q$ genau dann, wenn q bzgl. des korrespondierenden Zustands die Ausführung einer

[2] Beachte: Diese Forderung impliziert bereits, daß $q\|i$ in $p\|I$ erreichbar ist.

a–Transition verhindert, was bei einer korrekten Interfacespezifikation offensichtlich der Fall ist. Für die praktische Anwendung der hier entwickelten $\mathcal{RM}$–Methode ist diese Beobachtung von entscheidender Bedeutung. Wie in Kapitel 1 bereits erwähnt, sind die meisten Systeme bzw. Prozesse in der Praxis total definiert und $\uparrow$ lediglich ein für theoretische (besser: semantikorientierte) Betrachtungen wichtiges Hilfsprädikat. Die $\mathcal{RM}$–Methode basiert auf Interfacespezifikationen, die in der Praxis aufgrund der Komplexität der Probleme oft nur erraten werden können. Wie aber stellt man nun fest (ohne die Korrektheit einer Interfacespezifikation explizit beweisen zu müssen), ob tatsächlich $\overline{\Pi}_I(p)\|q \approx^d p\|q$ gilt, d.h. ob die durchgeführte Reduktion kontexttreu ist? Wie noch implizit bewiesen wird, ist das für total definierte Prozesse genau dann der Fall, wenn $\overline{\Pi}_I(p)\|q$ total definiert ist. Dies ist ein technischer Aspekt der Undefiniertheiten im Zusammenhang mit $\overline{\Pi}$. Dabei ist zu beachten, daß die totale Definiertheit von $\overline{\Pi}_I(p)\|q$ i.allg. nicht bedeutet, daß I eine korrekte Interfacespezifikation für p und q ist, sondern nur, daß die Reduktion bzgl. I kontexttreu ist. Es kann nämlich vorkommen, daß eine Reduktion mit einer inkorrekten Interfacespezifikation kontexttreu ist, falls diejenigen Teile der Interfacespezifikation, die eine kontexttreue Reduktion verhindern, für die Reduktion nicht betrachtet werden müssen.

Zur Veranschaulichung wird eine Reduktion vorgeführt, die auch bei der Anwendung der $\mathcal{RM}$–Methode auf das Beispielsystem benötigt wird.

Beispiel 2.4 (Beispiel einer Reduktion mit $\overline{\Pi}$)
Betrachte den Prozeß p aus Abbildung 2.1 mit $\mathcal{A}_p = \{tk1, tk2, rb2, sb2\}$. Es sei $\overline{\Pi}_{I_2}(p)$

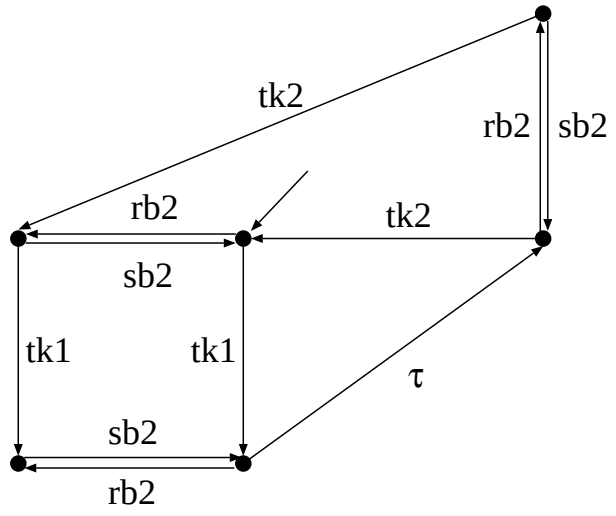


Abbildung 2.1: Beispielprozeß p

zu bestimmen, wobei I_2 die in Abbildung 1.5 definierte Interfacespezifikation ist. Gemäß Definition 2.3 ist der reduzierte Prozeß $\overline{\Pi}_{I_2}(p)$ im wesentlichen die Projektion von $p\|I_2$ auf p. Als Ergebnis der Konstruktion von $p\|I_2$ mit Hilfe von Definition 1.8 erhält man den Prozeß aus Abbildung 2.2, jedoch ohne die dort eingezeichneten Undefiniertheiten.

Dieser stimmt schon fast mit $\overline{\Pi}_{I_2}(p)$ überein, da nur Punkt (5b) aus Definition 2.3 noch

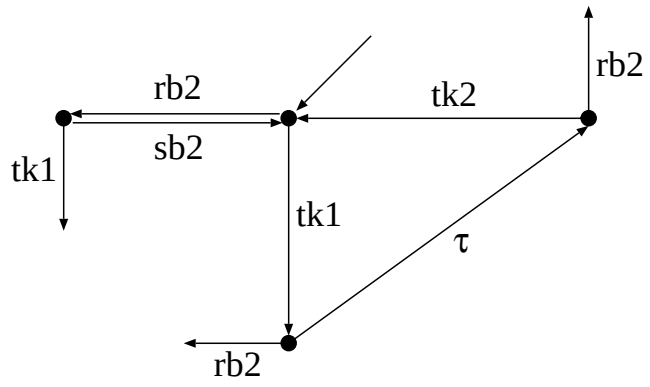


Abbildung 2.2: Der reduzierte Prozeß $\overline{\Pi}_{I_2}(p)$

nicht berücksichtigt wurde. Abbildung 2.2 zeigt, daß bei der Reduktion Zustände und Transitionen, die in durch I_2 spezifizierten Kontexten nicht erreichbar wären, weggefallen sind. Zur Kennzeichnung von solchen weggefallenen Transitionen fügt man gemäß Definition 2.3 (5b) an den Zuständen, von denen in p die jeweilige Transition ausging, eine entsprechende Undefiniertheit ein. Somit ist der Prozeß aus Abbildung 2.2 (mit Undefiniertheiten) der reduzierte Prozeß $\overline{\Pi}_{I_2}(p)$.

2.2 Theoretische Sicht

In Definition 2.3 wurde ein spezieller Reduktionsoperator vorgestellt, von dem in diesem Abschnitt gezeigt wird, daß er tatsächlich ein Reduktionsoperator im Sinne von Definition 2.1 ist und zusätzlich Bedingung (iii) dieser Definition erfüllt.

Satz 2.5
Die Abbildung $\overline{\Pi}$: $\underline{\text{Interfaces}} \times \underline{\text{Processes}} \dashrightarrow \underline{\text{Processes}}$ aus Definition 2.3 definiert einen Reduktionsoperator.

Zum Beweis der Eigenschaften (i) und (ii) aus Definition 2.1 werden Hilfsaussagen benötigt, die auch zum Verständnis der Definition 2.3 beitragen.

Betrachtung von $\overline{\Pi}$ bzgl. der Quasiordnung

Hier geht es um Eigenschaften des Reduktionsoperators $\overline{\Pi}$, die nicht nur zum Beweis der Eigenschaft (i) aus Definition 2.3 nützlich sind, sondern auch eine wichtige Rolle in Kapitel 4 spielen. Grundlegend dabei ist die folgende Definition:

Definition 2.6 (Halbordnung auf Prozessen)
Seien $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p), q = ((S_q, \mathcal{A}_q \cup \{\tau\}, \longrightarrow_q, \uparrow_q), q) \in \underline{\text{Processes}}$. Dann gelte $p \leq q$ genau dann, wenn folgende Bedingungen erfüllt sind:

1. $\mathcal{A}_p = \mathcal{A}_q$.
2. $S_p \subseteq S_q$ *und die Startzustände* p *und* q *sind identisch.*
3. $\forall\, p', p'' \in S_p.\ \ p' \xrightarrow{\tau}_p p''$ *genau dann, wenn* $p' \xrightarrow{\tau}_q p''$.
4. $\forall\, p', p'' \in S_p\, \forall\, a \in \mathcal{A}_p.\ \ p' \xrightarrow{a}_p p''$ *impliziert* $p' \xrightarrow{a}_q p''$.
5. $\forall\, p' \in S_p.\ p'\uparrow_p \tau$ *genau dann, wenn* $p'\uparrow_q \tau$.
6. $\forall\, p' \in S_p\, \forall\, a \in \mathcal{A}_p.\ \ p'\uparrow_p a$ *genau dann, wenn eine der beiden folgenden Bedingungen gilt:*
   (a) $p'\uparrow_q a$ *oder*
   (b) $\exists\, p'' \in S_q.\ p' \xrightarrow{a}_q p'' \ \wedge\ \ \nexists\, p'' \in S_p.\ p' \xrightarrow{a}_p p''$.

Es ist zu beweisen, daß durch $\leq$ tatsächlich eine *Halbordnung* auf Prozessen definiert wird. Diese Eigenschaft von $\leq$ spielt in diesem Kapitel jedoch keine Rolle, so daß auf den Beweis verzichtet wird.
Vergleicht man Definition 2.6 mit Definition 2.3, so stellt man fest, daß erstere von der letzteren „abstammt", denn die Eigenschaften (1), (2), (5) und (6) aus Definition 2.6 werden sofort durch die Punkte (1), (2), (4) und (5) von Definition 2.3 impliziert. Für die Punkte (3) und (4) muß man noch Lemma 2.11 zu Hilfe nehmen. Dieser Vergleich liefert die Aussage und den Beweis des nächsten Lemmas.

Lemma 2.7
Seien $p \in$ *Processes und* $I \in \mathcal{I}(p)$. *Dann gilt* $\overline{\Pi}_I(p) \leq p$.

Den Zusammenhang zwischen der Halbordnung $\leq$ und der Quasiordnung beschreibt das folgende Lemma:

Lemma 2.8
Für alle Prozesse $p, q \in$ *Processes mit* $p \leq q$ *gilt* $p \quad q$.

Beweis
Seien $p, q \in$ Processes beliebig. Definiere $R =_{df} \{(p', p') \mid p' \in S_p\}$. Es ist zu zeigen, daß R eine Relation im Sinne von Definition 1.15 ist, so daß $R \subseteq$ gilt. Da beide Prozesse nach Definition 2.6 (2) denselben Startzustand besitzen, gilt insbesondere $(p, q) \in R$ und damit auch $p \quad q$.

Seien also $(\overline{p}, \overline{p}) \in R$, $a \in \mathcal{A}_p \cup \{\tau\}$ beliebig und gelte $\neg(\overline{p}\uparrow_p a)$. Verifiziere nun die Eigenschaften (1) – (3) aus Definition 1.15.

1. Angenommen es gilt $\overline{p} \Uparrow_q a$, also nach Definition 1.11 für ein $k \geq 1$:

$$\overline{p} \equiv \overline{p}_1 \xrightarrow{\tau}_q \cdots \xrightarrow{\tau}_q \overline{p}_k \uparrow_q a.$$

   Dann gilt nach Definition 2.6 (1), (2), (3) und (5) bzw. (6a):

$$\overline{p} \equiv \overline{p}_1 \xrightarrow{\tau}_p \cdots \xrightarrow{\tau}_p \overline{p}_k \uparrow_p a.$$

   Es folgt mit Definition 1.11 $\overline{p} \Uparrow_p a$, im Widerspruch zur Voraussetzung $\neg(\overline{p} \Uparrow_p a)$. Somit gilt $\neg(\overline{p} \Uparrow_q a)$, also Bedingung (1) aus Definition 1.15.

2. Gelte $\overline{p} \stackrel{a}{\Longrightarrow}_p \overline{p}'$, also nach Definition 1.11 für ein l und k mit $l > k \geq 1$:

$$\overline{p} \equiv \overline{p}_1 \xrightarrow{\tau}_p \cdots \xrightarrow{\tau}_p \overline{p}_k \xrightarrow{a}_p \overline{p}_{k+1} \xrightarrow{\tau}_p \cdots \xrightarrow{\tau}_p \overline{p}_l \equiv \overline{p}'.$$

   Dann gilt nach Definition 2.6 (1), (2), (3) und (4):

$$\overline{p} \equiv \overline{p}_1 \xrightarrow{\tau}_q \cdots \xrightarrow{\tau}_q \overline{p}_k \xrightarrow{a}_q \overline{p}_{k+1} \xrightarrow{\tau}_q \cdots \xrightarrow{\tau}_q \overline{p}_l \equiv \overline{p}'.$$

   Mit Definition 1.11 ergibt dies $\overline{p} \stackrel{a}{\Longrightarrow}_q \overline{p}'$, und wegen $(\overline{p}', \overline{p}') \in R$ gilt damit Eigenschaft (2) aus Definition 1.15.

3. Gelte $\overline{p} \stackrel{a}{\Longrightarrow}_q \overline{p}'$, also nach Definition 1.11 für ein l und k mit $l > k \geq 1$:

$$\overline{p} \equiv \overline{p}_1 \xrightarrow{\tau}_q \cdots \xrightarrow{\tau}_q \overline{p}_k \xrightarrow{a}_q \overline{p}_{k+1} \xrightarrow{\tau}_q \cdots \xrightarrow{\tau}_q \overline{p}_l \equiv \overline{p}'.$$

   Dann folgt $\overline{p} \stackrel{a}{\Longrightarrow}_p \overline{p}'$, denn sonst müßte nach Definition 2.6 (3) und (6b) gelten:

$$\overline{p} \equiv \overline{p}_1 \xrightarrow{\tau}_p \cdots \xrightarrow{\tau}_p \overline{p}_k \uparrow_p a.$$

   Mit Definition 1.11 ergibt sich $\overline{p} \Uparrow_p a$, im Widerspruch zur Voraussetzung $\neg(\overline{p} \Uparrow_p a)$. Zusammen mit $(\overline{p}', \overline{p}') \in R$ folgt Eigenschaft (3) aus Definition 1.15.

Also ist R eine Relation gemäß Definition 1.15, womit die Behauptung folgt.

Kontexttreue der Reduktion

Um die Kontexttreue der Reduktion, d.h. $\overline{\Pi}_I(p) \| q \approx^d p \| q$, für beliebige Prozesse $p, q \in$ <u>Processes</u> und beliebige Interfacespezifikationen $I \in \mathcal{I}(p, q)$ zu beweisen, zeigen wir eine schärfere Aussage:

Satz 2.9 (Kontexttreue der Reduktion)
Seien $p, q \in$ <u>Processes</u> und $I \in \mathcal{I}(p, q)$. Dann gilt $p \| q \;=\; \overline{\Pi}_I(p) \| q$.

Für den Beweis dieser Aussage, aus welcher wegen „$r = s$ impliziert $r \approx^d s$" für beliebige Prozesse r und s und unter Voraussetzung korrekter Interfacespezifikationen die Kontexttreue der Reduktion bzgl. des speziellen Reduktionsoperators $\overline{\Pi}$ folgt, sind die beiden folgenden Lemmata von zentraler Bedeutung.

Intuitiv besagt das erste, daß der Reduktionsoperator $\overline{\Pi}$ nicht „zuviel abschneidet" (also im Parallelkontext q erreichbare Zustände bzw. Transitionen von p auch nach Anwendung von $\overline{\Pi}$ bzgl. eines Interfaces $I \in \mathcal{I}(p,q)$ erhalten bleiben), falls die zugrundeliegende Interfacespezifikation I für p und q korrekt ist. Formal bedeutet dies:

Lemma 2.10
Seien $p, q \in$ <u>Processes</u>, $I \in \mathcal{I}(p,q)$, $a \in \mathcal{A}_p \cup \{\tau, \epsilon\}$ und $p\|q \longrightarrow^*_{p\|q} p'\|q' \xrightarrow{a}_{p\|q} p''\|q''$. Dann gilt:

1. *$\exists\, I'' \in S_I \,.\, p''\|I''$ ist in $p\|I$ erreichbar.*
2. *$p' \xrightarrow{a}_{\overline{\Pi}(I,p)} p''$.*

Beweis
Nach Satz 2.17, der das zentrale Resultat des Abschnitts 2.3 sein wird, darf man o.B.d.A. davon ausgehen, daß I eine deterministische Interfacespezifikation ist.

Beweis mittels Induktion nach der Pfadlänge n von $p\|q \longrightarrow^n_{p\|q} p''\|q''$:

Induktionsanfang ($n = 0$):

D.h. $a = \epsilon$ und $p\|q = p'\|q' = p''\|q''$, also $p = p' = p''$ und $q = q' = q''$. Wähle $I'' =_{df} I$, dann ist $p''\|I'' = p\|I$ trivialerweise in $p\|I$ erreichbar, also gilt (1). Teil (2) ist wegen $p' \xrightarrow{\epsilon}_{\overline{\Pi}(I,p)} p''$ trivial.

Induktionsschluß ($n \longrightarrow n+1$): ($p\|q \longrightarrow^n_{p\|q} p'\|q' \xrightarrow{a}_{p\|q} p''\|q''$)

Nach Induktionsannahme existiert ein $I' \in S_I$ mit:

$$(*) \qquad p'\|I' \text{ ist in } p\|I \text{ erreichbar.}$$

Durch Anwendung des Fensteroperators bzgl. $\mathcal{A}_p \cap \mathcal{A}_q$ erhält man:

$$(p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q\rangle \longrightarrow^n_{p\|q} (p'\|q')\langle \mathcal{A}_p \cap \mathcal{A}_q\rangle \xrightarrow{b}_{p\|q} (p''\|q'')\langle \mathcal{A}_p \cap \mathcal{A}_q\rangle$$

mit $b = \begin{cases} a & \text{falls } a \in \mathcal{A}_q \\ \tau & \text{sonst} \end{cases}$. Im folgenden sei $b' = \begin{cases} \epsilon & \text{falls } b = \tau \\ b & \text{sonst} \end{cases}$.

Nach Induktionsannahme und mit der Voraussetzung (vgl. Definition 1.23) $\mathcal{L}((p\|q)\langle \mathcal{A}_p \cap \mathcal{A}_q\rangle) \subseteq \mathcal{L}(I)$ folgt mit Hilfe von Definition 1.7 und aufgrund des deterministischen I die Existenz von einem $I'' \in S_I$ mit $I' \overset{b'}{\Longrightarrow}_I I''$. Dies bedeutet:

$$\exists i', i'' \,.\, I' \xrightarrow{\tau}_I \ldots \xrightarrow{\tau}_I i' \xrightarrow{b'}_I i'' \xrightarrow{\tau}_I \ldots \xrightarrow{\tau}_I I'' \;,$$

also mit Definition 1.8 Regel (3) und evtl. (5) (falls $b' = a$)

$$p' \| I' \xrightarrow{\tau}_{p\|I} \ldots \xrightarrow{\tau}_{p\|I} p' \| i' \xrightarrow{a}_{p\|I} p'' \| i'' \xrightarrow{\tau}_{p\|I} \ldots \xrightarrow{\tau}_{p\|I} p'' \| I''.$$

Zusammen mit $(*)$ ist daher auch $p''\|I''$ in $p\|I$ erreichbar, d.h. es gilt (1).
Teil (2) ergibt sich aus der Existenz von obigen i' und i'', der Erreichbarkeit von $p''\|i''$ in $p\|I$, sowie $p'\|i' \xrightarrow{a}_{p\|I} p''\|i''$ mit Hilfe von Definition 2.3 (3).

Per Induktionsprinzip folgt die Behauptung.

Daß $\overline{\Pi}_I(p)$ zu einem Prozeß p keine Transitionen „hinzuerfindet" (also nur Transitionen enthält, die bereits in p enthalten sind), besagt das folgende Lemma:

Lemma 2.11
Seien $p \in$ Processes und $I \in \mathcal{I}(p)$ beliebig. Dann gilt:

$$\forall\, p', p'' \in S_{\overline{\Pi}(I,p)}\ \forall\, a \in \mathcal{A}_p \cup \{\tau\} \,.\; p' \xrightarrow{a}_{\overline{\Pi}(I,p)} p'' \textit{ impliziert } p' \xrightarrow{a}_p p''$$

Beweis
Seien $p', p'' \in S_{\overline{\Pi}(I,p)}$ und $a \in \mathcal{A}_p \cup \{\tau\}$ beliebig.[3] Dann impliziert $p' \xrightarrow{a}_{\overline{\Pi}(I,p)} p''$ wegen Definition 2.3 (3) $\exists i', i'' \in S_I \,.\, p'\|i' \xrightarrow{a}_{p\|I} p''\|i''$.
1. Fall: $a \notin \mathcal{A}_I \quad \Rightarrow$ (Definition 1.8 (3)) $\quad i' = i'' \quad \wedge\, p' \xrightarrow{a}_p p''$.
2. Fall: $a \in \mathcal{A}_I \quad \Rightarrow$ (Definition 1.8 (5)) $\quad i' \xrightarrow{a}_I i'' \wedge\, p' \xrightarrow{a}_p p''$.

Nun zum eigentlichen Beweis von Satz 2.9:

Beweis
Seien $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p), q = ((S_q, \mathcal{A}_q \cup \{\tau\}, \longrightarrow_q, \uparrow_q), q) \in$ Processes, $I \in \mathcal{I}(p,q)$ und $\mathcal{A} =_{df} \mathcal{A}_I = \mathcal{A}_p \cap \mathcal{A}_q$. Weiterhin gelten die folgenden Bezeichnungen:

$$\begin{aligned} p\|q &= ((S_1, \mathcal{A}_p \cup \mathcal{A}_q \cup \{\tau\}, \longrightarrow_1, \uparrow_1), p\|q) \\ \Pi_I(p) &= ((S_{p_I}, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_{p_I}, \uparrow_{p_I}), p) \\ \Pi_I(p)\|q &= ((S_2, \mathcal{A}_p \cup \mathcal{A}_q \cup \{\tau\}, \longrightarrow_2, \uparrow_2), p\|q). \end{aligned}$$

Da S_1 und S_2 beides Teilmengen der Menge $\{\,(p'\|q') \mid p' \in S_p, q' \in S_q\,\}$ sind, bleibt zu zeigen, daß $S_1 = S_2$, $\longrightarrow_1 = \longrightarrow_2$ und $\uparrow_1 = \uparrow_2$ gilt. Dazu definiere für $i = 1, 2$:

- S_i^n, die Teilmenge aller in n Schritten vom Startzustand aus erreichbaren Zustände,
- $\longrightarrow_i^n$, die Menge aller von einem Zustand in S_i^n ausgehenden Transitionen, und

[3] Beachte $\tau \notin \mathcal{A}_I$.

- $\uparrow_i^n$, das Undefiniertheitsprädikat der Zustände in S_i^n.

Führe nun eine Induktion über n durch[4]. Wegen $S_1^0 = S_2^0 = \{p\|q\}$ ist der Induktionsanfang trivial. Um den Beweis zu vervollständigen, genügt es, den Induktionsschritt für $n \geq 1$ unter der Induktionsannahme $S_1^{n-1} = S_2^{n-1}$ durchzuführen:

1. $\longrightarrow_1^{n-1} = \longrightarrow_2^{n-1}$
2. $S_1^n = S_2^n$
3. $\uparrow_1^{n-1} = \uparrow_2^{n-1}$.

Zunächst ist Punkt (1) des Induktionsschritts zu verifizieren. Beweise dazu die Äquivalenz $p'\|q' \xrightarrow{a}_1 p''\|q'' \iff p'\|q' \xrightarrow{a}_2 p''\|q''$ anhand der drei anwendbaren Regeln aus Definition 1.8:

Regel 3: Hier gilt $a \in (\mathcal{A}_p \setminus \mathcal{A}_q) \cup \{\tau\}$ und $q' = q''$, so daß man wie folgt schließen kann:

$$
\begin{array}{lcl}
 & & p'\|q' \xrightarrow{a}_1 p''\|q'' \\
(\text{Regel 3}) & \iff & p' \xrightarrow{a}_p p'' \\
(\text{Lemma 2.10 bzw. 2.11}^5) & \iff & p' \xrightarrow{a}_{pI} p'' \\
(\text{Regel 3}) & \iff & p'\|q' \xrightarrow{a}_2 p''\|q''
\end{array}
$$

Regel 4: Hier gilt $a \in (\mathcal{A}_q \setminus \mathcal{A}_p) \cup \{\tau\}$ und $p' = p''$, und deswegen:

$$
\begin{array}{lcl}
 & & p'\|q' \xrightarrow{a}_1 p''\|q'' \\
(\text{Regel 4}) & \iff & q' \xrightarrow{a}_q q'' \\
(\text{Regel 4}) & \iff & p'\|q' \xrightarrow{a}_2 p''\|q''
\end{array}
$$

[4]Beachte, daß hier ausschließlich endliche Transitionssysteme betrachtet werden.

[5]Der Kommentar „Lemma 2.10 bzw. 2.11“ ist beim Beweis dieses Lemmas wie folgt zu verstehen: Für die Beweisrichtung „ $\Rightarrow$ “ wird Lemma 2.10 benötigt und für die Rückrichtung „ $\Leftarrow$ “ Lemma 2.11.

Regel 5: Hier gilt $a \in \mathcal{A}$, $p' \xrightarrow{a}_p p''$ und $q' \xrightarrow{a}_q q''$:

$$
\begin{array}{lcl}
 & & p' \| q' \xrightarrow{a}_1 p'' \| q'' \\
(\text{Regel } 5) & \Longleftrightarrow & p' \xrightarrow{a}_p p'' \wedge q' \xrightarrow{a}_q q'' \\
(\text{Lemma 2.10 bzw. 2.11}) & \Longleftrightarrow & p' \xrightarrow{a}_{p_I} p'' \wedge q' \xrightarrow{a}_q q'' \\
(\text{Regel } 5) & \Longleftrightarrow & p' \| q' \xrightarrow{a}_2 p'' \| q''
\end{array}
$$

Also gilt Punkt (1) des Induktionsschritts. Punkt (2) ist eine unmittelbare Konsequenz aus Punkt (1). Es verbleibt Punkt (3) zu zeigen. Da die τ–Undefiniertheit eines Prozesses durch den Reduktionsoperator $\overline{\Pi}$ (vgl. Definition 2.3 (4)) unberührt bleibt, ist der Fall der a–Undefiniertheit für $a \neq \tau$ zu betrachten. Beweise dazu die Äquivalenz $(p' \| q') \uparrow_1 a \iff (p' \| q') \uparrow_2 a$ anhand der fünf anwendbaren Regeln aus Definition 1.8:

Regel 8: Hier gilt $p' \uparrow_p a$ und $a \notin \mathcal{A}_q$, und deswegen:

$$
\begin{array}{lcl}
 & & (p' \| q') \uparrow_1 a \\
(\text{Regel } 8) & \Longleftrightarrow & p' \uparrow_p a \\
(\text{Definition 2.3 (5) (a)}^6) & \Longleftrightarrow & p' \uparrow_{p_I} a \\
(\text{Regel } 8) & \Longleftrightarrow & (p' \| q') \uparrow_2 a
\end{array}
$$

Regel 9: Hier gilt $p' \uparrow_p a$ und $q' \xrightarrow{a}_q$, also $a \in \mathcal{A}_q$:

$$
\begin{array}{lcl}
 & & (p' \| q') \uparrow_1 a \\
(\text{Regel } 9) & \Longleftrightarrow & p' \uparrow_p a \wedge q' \xrightarrow{a}_q \\
(\text{Definition 2.3 (5) (a)}^7) & \Longleftrightarrow & p' \uparrow_{p_I} a \wedge q' \xrightarrow{a}_q \\
(\text{Regel } 9) & \Longleftrightarrow & (p' \| q') \uparrow_2 a
\end{array}
$$

[6] ad " $\Leftarrow$ ": Die Anwendung von Definition 2.3 (5) (b) ist nicht möglich, denn sonst folgt mit Regel (3) $p' \| q' \xrightarrow{a}_1$ und damit (s.o. bzgl. Regel (3)) $p' \| q' \xrightarrow{a}_2$ (Widerspruch zur Konstruktion $(p' \| q') \uparrow_2 a$).

Regel 10: Hier gilt $q'\uparrow_q a$ und $a \notin \mathcal{A}_p$:

$$
\begin{aligned}
& & & (p'\|q')\uparrow_1 a \\
& (\text{Regel 10}) & \Longleftrightarrow\ & q'\uparrow_q a \\
& (\text{Regel 10}) & \Longleftrightarrow\ & (p'\|q')\uparrow_2 a
\end{aligned}
$$

Regel 11: Hier gilt $q'\uparrow_q a$ und $p' \xrightarrow{a}_p$, also $a \in \mathcal{A}_p$:

$$
\begin{aligned}
& & & (p'\|q')\uparrow_1 a \\
& (\text{Regel 11}) & \Longleftrightarrow\ & q'\uparrow_q a \wedge p' \xrightarrow{a}_p \\
& (\text{Lemma 2.10 bzw. 2.11}) & \Longleftrightarrow\ & q'\uparrow_q a \wedge p' \xrightarrow{a}_{p_I} \\
& (\text{Regel 11}) & \Longleftrightarrow\ & (p'\|q')\uparrow_2 a
\end{aligned}
$$

Regel 12: Hier gilt $p'\uparrow_p a$ und $q'\uparrow_q a$, und deswegen:

<u>„ $\Rightarrow$ “:</u>

$$
\begin{aligned}
& (p'\|q')\uparrow_1 a & \Rightarrow\ & p'\uparrow_p a \wedge q'\uparrow_q a \\
& (\text{Definition 2.3 (5) (a)}) & \Rightarrow\ & p'\uparrow_{p_I} a \wedge q'\uparrow_q a \\
& (\text{Regel 12}) & \Rightarrow\ & (p'\|q')\uparrow_2 a
\end{aligned}
$$

<u>„ $\Leftarrow$ “:</u>

$$
\begin{aligned}
& (p'\|q')\uparrow_2 a & \Rightarrow\ & p'\uparrow_{p_I} a \wedge q'\uparrow_q a \\
& (\text{Definition 2.3 (5)}) & \Rightarrow\ & (5a)\ p'\uparrow_p a \wedge q'\uparrow_q a \Rightarrow (\text{Regel 12})\ (p'\|q')\uparrow_1 a \\
& & \text{oder}\ & (5b)\ p' \xrightarrow{a}_p \wedge q'\uparrow_q a \Rightarrow (\text{Regel 11})\ (p'\|q')\uparrow_1 a
\end{aligned}
$$

Per Induktionsprinzip folgt die Behauptung.

[7] ad “ $\Leftarrow$ ”: Siehe vorige Fußnote.

Nun kann der Beweis geführt werden, daß $\overline{\Pi}$ ein Reduktionsoperator ist:

Beweis
Zum Beweis von Satz 2.5 sind lediglich die Forderungen (i) und (ii) aus Definition 2.1 zu verifizieren.

ad (i): Die Aussagen aus den Lemmata 2.7 und 2.8 liefern sofort die Gültigkeit der Bedingung $\overline{\Pi}(I,p) \quad p$ für beliebige $p \in$ Processes und $I \in \mathcal{I}(p)$.

ad (ii): Die Gültigkeit von $\Pi(I,p)\|q \approx^d p\|q$ für beliebige $p, q \in$ Processes und $I \in \mathcal{I}(p,q)$ ist eine unmittelbare Konsequenz aus Satz 2.9.

Aus (i) und (ii) folgt mit Definition 2.1 die Behauptung. Zusätzlich erfüllt $\overline{\Pi}$ die technische Bedingung (iii).

ad (iii): Seien $p \in$ Processes und $I \in \mathcal{I}(p)$ beliebig.
Nach Definition 2.3 (1) gilt $S = \{q \in S_p \mid \exists\, i \in S_I .\, q\|i \in S_{p\|I}\} \subseteq S_p$, also folgt

$$\mid S_{\overline{\Pi}(I,p)} \mid \leq \mid S_p \mid .$$

Die zweite Bedingung $\mid \longrightarrow_{\overline{\Pi}(I,p)} \mid \leq \mid \longrightarrow_p \mid$ folgt mit Hilfe von Lemma 2.11 ebenso einfach.

Verträglichkeit von $\overline{\Pi}$ mit der semantischen Äquivalenz

Die Verträglichkeit eines Reduktionsoperators Π mit der semantischen Äquivalenz $\approx^d$ ist bereits durch Proposition 2.2 im Rahmen eines beliebigen Parallelkontextes $(\cdot\|q)$ mit $q \in$ Processes gezeigt worden. Dies reicht aus, um die Korrektheit und Optimalität der in Kapitel 3 vorgestellten $\mathcal{RM}$–Methode beweisen zu können. Dennoch soll hier gezeigt werden, daß für den speziellen Reduktionsoperator $\overline{\Pi}$ die „Verträglichkeit“ allgemeiner gilt.

Lemma 2.12
Seien $p, q \in$ Processes und $I \in \mathcal{I}(p) \cap \mathcal{I}(q)$. Dann gilt:

$$p \approx^d q \;\; \textit{impliziert} \;\; \overline{\Pi}_I(p) \approx^d \overline{\Pi}_I(q).$$

Beweis
Seien $p, q \in$ Processes mit $p \approx^d q$, sowie $I \in \mathcal{I}(p) \cap \mathcal{I}(q)$ beliebig. Ferner sei

$$\overline{\Pi}_I(p) = ((S_1, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_1, \uparrow_1), p) \text{ und } \overline{\Pi}_I(q) = ((S_2, \mathcal{A}_q \cup \{\tau\}, \longrightarrow_2, \uparrow_2), q).$$

Definiere

$$R =_{df} \{(\overline{p}, \overline{q}) \mid \overline{p} \in S_1, \overline{q} \in S_2, \overline{p} \approx^d \overline{q}\}$$

und zeige, daß R eine Relation im Sinne von Definition 1.12 ist.[8] Seien dazu $(\overline{p}, \overline{q}) \in R$ und $a \in \mathcal{A} =_{df} \mathcal{A}_p = \mathcal{A}_q$ beliebig. Dann ist zu zeigen:

1. $\overline{p} \Uparrow_1 a$ genau dann, wenn $\overline{q} \Uparrow_2 a$,
2. $\overline{p} \stackrel{a}{\Longrightarrow}_1 \overline{p}'$ impliziert $\exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow}_2 \overline{q}' \wedge \overline{p}'\, R\, \overline{q}'$ und
3. $\overline{q} \stackrel{a}{\Longrightarrow}_1 \overline{q}'$ impliziert $\exists\, \overline{p}'.\ \overline{p} \stackrel{a}{\Longrightarrow}_2 \overline{p}' \wedge \overline{p}'\, R\, \overline{q}'$.

Dann gilt $R \subseteq \approx^d$ und wegen $(p, q) \in R$ folgt $\overline{\Pi}_I(p) \approx^d \overline{\Pi}_I(q)$. In den folgenden Beweisen ist zu beachten, daß die Eigenschaften aus Definition 2.3 die entsprechenden Eigenschaften mit schwacher Transitionsrelation bzw. schwachem Undefiniertheitsprädikat (vgl. Definition 1.11) implizieren. Dies gilt analog für die operationelle Semantik aus Definition 1.8.

<u>ad (1):</u>

Fallunterscheidung entsprechend den Bedingungen (5a) und (5b) aus Definition 2.3:

<u>1. Fall:</u> Es gilt Bedingung (5a).

$$\begin{array}{lcl} & & \overline{p} \Uparrow_1 a \\ (\text{Definition 2.3 (5a)}) & \Longleftrightarrow & \overline{p} \Uparrow_p a \\ (\overline{p} \approx^d \overline{q}) & \Longleftrightarrow & \overline{q} \Uparrow_q a \\ (\text{Definition 2.3 (5a)}) & \Longleftrightarrow & \overline{q} \Uparrow_2 a \end{array}$$

<u>2. Fall:</u> Es gilt Bedingung (5b).

$$\begin{array}{lcl} & & \overline{p} \Uparrow_1 a \\ (\text{Definition 2.3 (5b)}) & \Longleftrightarrow & \exists\, \overline{p}' \in S_p.\ \overline{p} \stackrel{a}{\Longrightarrow}_p \overline{p}' \ \wedge\ \nexists\, \overline{p}' \in S_1.\ \overline{p} \stackrel{a}{\Longrightarrow}_1 \overline{p}' \\ (\overline{p} \approx^d \overline{q}) & \Longleftrightarrow & \exists\, \overline{q}' \in S_q.\ \overline{q} \stackrel{a}{\Longrightarrow}_q \overline{q}' \ \wedge\ \nexists\, \overline{q}' \in S_2.\ \overline{q} \stackrel{a}{\Longrightarrow}_2 \overline{q}' \\ (\text{Definition 2.3 (5b)}) & \Longleftrightarrow & \overline{q} \Uparrow_2 a \end{array}$$

[8] Beachte bei der Definition von R, daß sich $\overline{p} \approx^d \overline{q}$ auf die Prozesse p und q bezieht.

<u>ad (2):</u> Betrachte dazu wiederum eine vollständige Fallunterscheidung:

<u>1. Fall:</u> $a \notin \mathcal{A}_I$

$$\overline{p} \stackrel{a}{\Longrightarrow}_1 \overline{p}'$$

$$\begin{array}{lcl}
(\text{Definition 2.3 (3)}) & \Longleftrightarrow & \exists\, i, i' \in S_I.\ \overline{p}\|i \stackrel{a}{\Longrightarrow}_{p\|I} \overline{p}'\|i' \text{ und } \overline{p}\|i \text{ ist in } p\|I \text{ erreichbar} \\[2ex]
(\text{Definition 1.8 (3)}) & \Longleftrightarrow & \exists\, i, i' \in S_I.\ \overline{p} \stackrel{a}{\Longrightarrow}_p \overline{p}',\ i \equiv i' \text{ und } \overline{p}\|i \text{ ist in } p\|I \text{ erreichbar} \\[2ex]
(p \approx^d q \ \&\ \overline{p} \approx^d \overline{q}) & \Longleftrightarrow & \exists\, \overline{q}'.\ \exists\, i, i' \in S_I.\ \overline{q} \stackrel{a}{\Longrightarrow}_q \overline{q}',\ i \equiv i',\ \overline{q}\|i \text{ ist in } q\|I \text{ erreichbar u.} \\
 & & \quad \overline{p}' \approx^d \overline{q}' \\[2ex]
(\text{Definition 1.8 (3)}) & \Longleftrightarrow & \exists\, \overline{q}'.\ \exists\, i, i' \in S_I.\ \overline{q}\|i \stackrel{a}{\Longrightarrow}_{q\|I} \overline{q}'\|i',\ \overline{q}\|i \text{ ist in } q\|I \text{ erreichbar und} \\
 & & \quad \overline{p}' \approx^d \overline{q}' \\[2ex]
(\text{Def. 2.3 (2)} \ \&\ R) & \Longleftrightarrow & \exists\, \overline{q}'.\ \exists\, i, i' \in S_I.\ \overline{q}\|i \stackrel{a}{\Longrightarrow}_{q\|I} \overline{q}'\|i',\ \overline{q}\|i \text{ ist in } q\|I \text{ erreichbar und} \\
 & & \quad \overline{p}'\, R\, \overline{q}' \\[2ex]
(\text{Definition 2.3 (3)}) & \Longleftrightarrow & \exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow}_2 \overline{q}' \ \wedge\ \overline{p}'\, R\, \overline{q}'
\end{array}$$

<u>2. Fall:</u> $a \in \mathcal{A}_I$

$$\overline{p} \stackrel{a}{\Longrightarrow}_1 \overline{p}'$$

$$\begin{array}{lcl}
(\text{Definition 2.3 (3)}) & \Longleftrightarrow & \exists\, i, i' \in S_I.\ \overline{p}\|i \stackrel{a}{\Longrightarrow}_{p\|I} \overline{p}'\|i' \text{ und } \overline{p}\|i \text{ ist in } p\|I \text{ erreichbar} \\[2ex]
(\text{Definition 1.8 (5)}) & \Longleftrightarrow & \exists\, i, i' \in S_I.\ \overline{p} \stackrel{a}{\Longrightarrow}_p \overline{p}',\ i \stackrel{a}{\Longrightarrow}_I i' \text{ und } \overline{p}\|i \text{ ist in } p\|I \text{ erreichbar} \\
(p \approx^d q \ \&\ \overline{p} \approx^d \overline{q}) & \Longleftrightarrow & \exists\, \overline{q}'.\ \exists\, i, i' \in S_I.\ \overline{q} \stackrel{a}{\Longrightarrow}_q \overline{q}',\ i \stackrel{a}{\Longrightarrow}_I i',\ \overline{q}\|i \text{ ist in } q\|I \text{ erreichbar} \\
 & & \quad \text{und } \overline{p}' \approx^d \overline{q}' \\[2ex]
(\text{Definition 1.8 (5)}) & \Longleftrightarrow & \exists\, \overline{q}'.\ \exists\, i, i' \in S_I.\ \overline{q}\|i \stackrel{a}{\Longrightarrow}_{q\|I} \overline{q}'\|i',\ \overline{q}\|i \text{ ist in } q\|I \text{ erreichbar} \\
 & & \quad \text{und } \overline{p}' \approx^d \overline{q}' \\[2ex]
(\text{Def. 2.3 (2)} \ \&\ R) & \Longleftrightarrow & \exists\, \overline{q}'.\ \exists\, i, i' \in S_I.\ \overline{q}\|i \stackrel{a}{\Longrightarrow}_{q\|I} \overline{q}'\|i',\ \overline{q}\|i \text{ ist in } q\|I \text{ erreichbar} \\
 & & \quad \text{und } \overline{p}'\, R\, \overline{q}' \\[2ex]
(\text{Definition 2.3 (3)}) & \Longleftrightarrow & \exists\, \overline{q}'.\ \overline{q} \stackrel{a}{\Longrightarrow}_2 \overline{q}' \ \wedge\ \overline{p}'\, R\, \overline{q}'
\end{array}$$

ad (3):

Diese Aussage folgt analog zu Teil (2) mit vertauschten Rollen von p und q bzw. $\overline{p}$ und $\overline{q}$.

2.3 Algorithmische Sicht

Bis jetzt ist mit Definition 2.3 nur eine abstrakte Beschreibung des Reduktionsoperators $\overline{\Pi}$ angewandt auf einen Prozeß $p \in$ Processes und eine Interfacespezifikation $I \in \mathcal{I}(p)$ gegeben. Es stellt sich die Frage, wie man zu gegebenem p und I den reduzierten Prozeß $\overline{\Pi}_I(p)$ berechnen kann. Dazu bedarf es einer neuen Charakterisierung des Reduktionsoperators $\overline{\Pi}$ aus einem anderen, eher algorithmischen Blickwinkel heraus. Die zugrundeliegende Idee ist, die Zustände des betrachteten Transitionssystems p jeweils mit der Menge aller Aktionen zu beschriften, die bzgl. der Schnittstelle I (bzw. deren Sprache) von dem betrachteten Zustand aus ausgeführt werden dürfen. Die Bestimmung dieser Aktionenmenge ist eine Anwendung der *Datenflußanalyse,* wobei das *Koinzidenztheorem* von Kam und Ullman [KU77] eine zentrale Rolle spielt. Einen Einblick in die Datenflußanalyse geben die Ausführungen in Anhang C. Als Ergebnis erhält man neben einem korrekten, effektiven Algorithmus, welcher die abstrakte Charakterisierung des Reduktionsoperators $\overline{\Pi}$ in eine Berechnung umsetzt, auch ein wichtiges theoretisches Ergebnis, nämlich die *Repräsentationsunabhängigkeit.* Diese besagt, daß der reduzierte Prozeß $\overline{\Pi}_I(p)$ bzgl. p nur von der Sprache $\mathcal{L}(I)$ der Interfacespezifikation I und nicht von deren Prozeßstruktur abhängt. Diese Unabhängigkeit erlaubt insbesondere bei theoretischen Beweisen (wie im Beweis von Lemma 2.10), statt einer nichtdeterministischen eine dazu sprachäquivalente deterministische Interfacespezifikation zu betrachten. Ausgangspunkt ist zunächst die folgende Beobachtung:

Proposition 2.13 *(IL_q–Mengen)*
Sei $p \in$ Processes und $I \in \mathcal{I}(p)$. Definiere für $q \in S_p$ die Menge

$$\mathcal{L}_I(q) =_{df} \{a \in \mathcal{A}_I \cup \{\epsilon\} \mid \exists\, q \| i \in S_{p\|I}.\ i \xrightarrow{a}_I\}.$$

Dann ist $\overline{\Pi}_I(p)$ bereits durch die Mengen $IL_q =_{df} \mathcal{L}_I(q) \cup (\mathcal{A}_p \setminus \mathcal{A}_I) \cup \{\tau\}$ und folgende Konstruktionsvorschrift für $q \in S_p$ bestimmt:

1. *Ist $\epsilon \notin IL_q$, so eliminiere den Zustand q und alle zu q führenden und von q ausgehenden Transitionen.*

2. *Gilt $q \xrightarrow{a}_p$, aber $a \notin IL_q$, so eliminiere alle von q ausgehenden a–Transitionen und füge $q{\uparrow}a$ der Divergenzrelation hinzu.*

Die Mengen IL_q (für $q \in S_p$) enthalten diejenigen Aktionen bzgl. des Zustands q, die in dem durch die Interfacespezifikation I beschriebenen Kontext erlaubt sind. Eine wichtige Rolle spielt in diesem Zusammenhang ϵ. Ist $\epsilon \in IL_q$, so bedeutet dies, daß der Zustand q in S liegt,[9] also q im reduzierten Prozeß $\overline{\Pi}_I(p)$ erreichbar ist; denn in diesem Falle reduziert sich die Bedingung $\exists\, q\|i \in S_{p\|I}.\ i \xrightarrow{\epsilon}_I$ in der Definition von $\mathcal{L}_I(q)$ zu $\exists\, q\|i \in S_{p\|I}$.[10] Nun zum Beweis von Proposition 2.13:

Beweis

Um den Beweis der Korrespondenz zwischen Definition 2.3 und obiger Konstruktionsvorschrift zu führen, genügt es, folgendes zu zeigen:

1. $\forall\, q \in S_p.\quad q \in S \iff \epsilon \in IL_q$ und
2. $\forall\, q \in S\ \forall a \in \mathcal{A} \cup \{\tau\}.\quad q \xrightarrow{a} \iff q \xrightarrow{a}_p \wedge\ a \in IL_q.$

Dabei ist zu beachten, daß:

- nach Definition 2.3 (2) $\mathcal{A} = \mathcal{A}_p$ gilt,
- nach Definition 2.3 (4 & 5a) genau wie in obiger Konstruktionsvorschrift Undefiniertheiten an Zuständen aus S_p erhalten bleiben und
- nach Definition 2.3 (5b) genau wie in obiger Konstruktionsvorschrift a–Undefiniertheiten zusätzlich nur an solchen Zuständen $q \in S$ entstehen, an denen eine a–Transition in p durch Anwendung von $\overline{\Pi}_I$ gestrichen wurde.

<u>Beweis von (1):</u>

Sei $q \in S_p$ beliebig.

<u>„ $\Rightarrow$ “:</u>

$$
\begin{array}{lll}
 & & q \in S \\
(\text{Definition 2.3 (1)}) & \Rightarrow & \exists\, i \in S_I.\ q\|i \in S_{p\|I} \\
 & \Rightarrow & \exists\, i \in S_I.\ q\|i \in S_{p\|I} \wedge q\|i \xrightarrow{\epsilon}_{p\|I} q\|i \\
 & \Rightarrow & \exists\, q\|i \in S_{p\|I}.\ q\|i \xrightarrow{\epsilon}_{p\|I} \\
 & \Rightarrow & \exists\, q\|i \in S_{p\|I}.\ i \xrightarrow{\epsilon}_I \\
(\text{Definition von } IL_q) & \Rightarrow & \epsilon \in IL_q
\end{array}
$$

[9] Es gelte wieder $\overline{\Pi}_I(p) = ((S, \longrightarrow, \mathcal{A} \cup \{\tau\}, \uparrow), p)$.

[10] Beachte, daß für jeden Zustand s eines Prozesses $s \xrightarrow{\epsilon} s$ und damit $s \xrightarrow{\epsilon}$ gilt.

„ $\Leftarrow$ “:

$$\epsilon \in IL_q$$

(Definition von IL_q) $\Rightarrow$ $\exists\, q\|i \in S_{p\|I}.\ i \xrightarrow{\epsilon}_I$

$\Rightarrow$ $\exists\, i \in S_I.\ q\|i \in S_{p\|I}$

(Definition 2.3 (1)) $\Rightarrow$ $q \in S$

Beweis von (2):

Sei $q \in S$ und $a \in \mathcal{A} \cup \{\tau\}$ beliebig.

„ $\Rightarrow$ “:

Fall 1: $a \in (\mathcal{A} \setminus \mathcal{A}_I) \cup \{\tau\}$

$$q \xrightarrow{a}$$

(Definition 2.3 (3)) $\Rightarrow$ $\exists\, i \in S_I.\ q\|i \xrightarrow{a}_{p\|I}$

(Definition von IL_q und Definition 1.8 (3)) $\Rightarrow$ $a \in IL_q \wedge q \xrightarrow{a}_p$

Fall 2: $a \in \mathcal{A}_I \cup \{\epsilon\}$

$$q \xrightarrow{a}$$

(Definition 2.3 (3)) $\Rightarrow$ $\exists\, i \in S_I.\ q\|i \xrightarrow{a}_{p\|I}$

$\Rightarrow$ $\exists\, q\|i \in S_{p\|I}.\ q\|i \xrightarrow{a}_{p\|I}$

(Definition 1.8 (5)) $\Rightarrow$ $\exists\, q\|i \in S_{p\|I}.\ i \xrightarrow{a}_I \wedge q \xrightarrow{a}_p$

(Definition von IL_q) $\Rightarrow$ $a \in IL_q \wedge q \xrightarrow{a}_p$

„ $\Leftarrow$ “:

Fall 1: $a \in (\mathcal{A} \setminus \mathcal{A}_I) \cup \{\tau\}$

$$a \in IL_q \wedge q \xrightarrow{a}_p$$

(Definition 2.3 (1)) $\Rightarrow$ $\exists\, i \in S_I.\ q\|i \in S_{p\|I} \wedge q \xrightarrow{a}_p$

(Def. 1.8 (3) und $q\|i \in S_{p\|I}$) $\Rightarrow$ $\exists\, i \in S_I.\ q\|i \xrightarrow{a}_{p\|I} \wedge$

$q\|i$ ist in $p\|I$ erreichbar

(Definition 2.3 (3)) $\Rightarrow$ $q \xrightarrow{a}$

Fall 2: $a \in \mathcal{A}_I \cup \{\epsilon\}$[11]

$$a \in IL_q \wedge q \xrightarrow{a}_p$$

(Definition von IL_q) $\Rightarrow$ $\exists\, q\|i \in S_{p\|I}.\ i \xrightarrow{a}_I \wedge q \xrightarrow{a}_p$

(Definition 1.8 (5)) $\Rightarrow$ $\exists\, q\|i \in S_{p\|I}.\ q\|i \xrightarrow{a}_{p\|I}$

($q\|i \in S_{p\|I}$) $\Rightarrow$ $\exists\, i \in S_I.\ q\|i \xrightarrow{a}_{p\|I} \wedge q\|i$ ist in $p\|I$ erreichbar

(Definition 2.3 (3)) $\Rightarrow$ $q \xrightarrow{a}$

Entsprechend der Vorbemerkungen folgt die Behauptung.

Mit Hilfe dieses Satzes läßt sich $\overline{\Pi}_I(p)$ genau dann berechnen, wenn $\mathcal{L}_I(q)$ (für $q \in S_p$) berechenbar ist. Um dies zu erkennen und um ein effektives Verfahren angeben zu können, wird ein weiterer Zwischenschritt benötigt:

Lemma 2.14
Sei $q \in S_p$ und $\mathcal{L}'_I(q) =_{df} \{a \in \mathcal{A}_I \cup \{\epsilon\} \mid \exists\, v \in \mathcal{A}_I^*.\ av \in \bigcup\{\, \mathcal{L}(i) \mid q\|i \in S_{p\|I}\,\}\}$ die Menge aller Präfixe der Länge 1 [12] *der Sprachmenge $\bigcup\{\, \mathcal{L}(i) \mid q\|i \in S_{p\|I}\,\}$. Dann gilt:*

$$\mathcal{L}_I(q) = \mathcal{L}'_I(q).$$

[11] Beachte: $a \in \mathcal{A}_I \cup \{\epsilon\} \subseteq \mathcal{A}_p \cup \{\epsilon\}$

[12] Als solches soll hier auch ϵ verstanden werden.

Beweis

Beim Beweis ist zu beachten, daß eine Interfacespezifikation I nach Definition 1.23 total definiert ist. Außerdem gilt nach Definition 1.7 für einen beliebigen Prozeß $r \in$ Processes: $\epsilon \in \mathcal{L}(r)$. Sei nun $a \in \mathcal{A}_I \cup \{\epsilon\}$ beliebig.

„$\subseteq$"

$$a \in \mathcal{L}_I(q)$$

$$\begin{array}{lcl}
(\text{Definition von } \mathcal{L}_I(q)) & \Rightarrow & \exists\, q\|i \in S_{p\|I} \,.\, i \xrightarrow{a}_I \\
\textbf{Fall (1) : } a = \epsilon & & \\
(\text{Wähle } v = \epsilon) & \Rightarrow & \exists\, q\|i \in S_{p\|I} \;\; \exists\, v \in \mathcal{A}_I^* \,.\, av \in \mathcal{L}(i) \\
\textbf{Fall (2) : } a \neq \epsilon & & \\
(\text{Definition 1.7})^{13} & \Rightarrow & \exists\, q\|i \in S_{p\|I} \,.\, \mathcal{L}_a(i) \neq \emptyset \\
(\text{Definition 1.7})^{14} & \Rightarrow & \exists\, q\|i \in S_{p\|I} \,.\, \exists\, v \in \mathcal{A}_I^* \,.\, av \in \mathcal{L}(i) \\
\textbf{Insgesamt:} & \Rightarrow & \exists\, v \in \mathcal{A}_I^* \,.\, av \in \bigcup \{\, \mathcal{L}(i) \mid q\|i \in S_{p\|I} \,\} \\
(\text{Definition von } \mathcal{L}'_I(q)) & \Rightarrow & a \in \mathcal{L}'_I(q)
\end{array}$$

„$\supseteq$"

$$a \in \mathcal{L}'_I(q)$$

$$\begin{array}{lcl}
(\text{Definition von } \mathcal{L}'_I(q)) & \Rightarrow & \exists\, v \in \mathcal{A}_I^* \,.\, av \in \bigcup \{\, \mathcal{L}(i) \mid q\|i \in S_{p\|I} \} \\
 & \Rightarrow & \exists\, q\|i \in S_{p\|I} \;\; \exists\, v \in \mathcal{A}_I^* \,.\, av \in \mathcal{L}(i) \\
\textbf{Fall (1) : } a = \epsilon & & \\
(i \xrightarrow{\epsilon}_I i) & \Rightarrow & \exists\, q\|i \in S_{p\|I} \,.\, i \xrightarrow{a}_I
\end{array}$$

[13] Beachte: $\exists\, i' \in S_I \,.\, i \xrightarrow{a}_I i'$ und $\epsilon \in \mathcal{L}(i') \neq \emptyset$.

[14] Wähle $v \in \mathcal{L}_a(i) \neq \emptyset$ beliebig.

Fall (2) : $a \neq \epsilon$

$$
\begin{array}{lll}
(\text{Definition 1.7})^{15} & \Rightarrow & \exists\, q\|i \in S_{p\|I} \,.\, i \overset{a}{\Longrightarrow}_I \\
(\text{vgl. Definition 1.11}) & \Rightarrow & \exists\, q\|i \in S_{p\|I}\ \exists\, i' \in S_I \,.\, i \xrightarrow{\tau}_I \ldots \xrightarrow{\tau}_I i' \xrightarrow{a}_I \\
(\text{Definition 1.8 (4)}) & \Rightarrow & \exists\, q\|i \in S_{p\|I}\ \exists\, i' \in S_I \,.\, q\|i \xrightarrow{\tau}_{p\|I} \ldots \xrightarrow{\tau}_{p\|I} q\|i' \ \wedge \\
 & & \qquad\qquad i' \xrightarrow{a}_I \\
 & \Rightarrow & \exists\, q\|i' \in S_{p\|I} \,.\, i' \xrightarrow{a}_I
\end{array}
$$

Insgesamt:

$$
(\text{Definition von } \mathcal{L}_I(q)) \quad \Rightarrow \quad a \in \mathcal{L}_I(q)
$$

Im folgenden wird eine Prozedur vorgestellt, die für $q \in S_p$ die Sprachmengen

$$
\bigcup \{\, \mathcal{L}(i) \mid q\|i \in S_{p\|I} \,\}
$$

berechnet. Dabei handelt es sich um eine Anwendung der Datenflußanalyse, deren Grundlagen in Anhang C erläutert werden. Aufgrund der effizienten Berechenbarkeit dieser Mengen sind auch die Mengen $\mathcal{L}_I(q)$ effizient berechenbar.

Kern der Prozedur sind die folgenden Funktionen, die das für die Datenflußanalyse benötigte lokale abstrakte Semantikfunktional definieren, wobei <u>Languages</u> die Potenzmenge von $\mathcal{A}_I^*$ bezeichnet:

$$
\mathcal{E}_a^I : \ \underline{\text{Languages}} \ \rightarrow \ \underline{\text{Languages}}
$$

definiert durch

$$
\mathcal{E}_a^I(\mathcal{L}) = \begin{cases} \mathcal{L}_a & \text{falls } a \in \mathcal{A}_I \\ \mathcal{L} & \text{sonst} \end{cases}
$$

Dabei bezeichnet $\mathcal{L}_a$ die Menge aller a–Suffixe von $\mathcal{L}$ (vgl. Definition 1.7).

[15] Beachte $a \neq \epsilon$, und I ist total definiert.

Beachte dabei, daß die Funktionen $\mathcal{E}_a^I$ nicht von der speziellen Struktur der Interfacespezifikation I, sondern nur von deren Alphabet $\mathcal{A}_I$ abhängen.

Um diese Funktionen im Rahmen einer Datenflußanalyse sinnvoll anwenden zu können, müssen sie, wie im Anhang C gezeigt wird, *additiv* sein. Diese Eigenschaft ist eine Voraussetzung des Koinzidenztheorems, welches die Korrektheit und die Optimalität der unten angegebenen Prozedur garantiert. Tatsächlich gilt:

Lemma 2.15 (Additivität der $\mathcal{E}_a^I$)
Die Funktionen $\mathcal{E}_a^I$ sind additiv, d.h. es gilt:

$$\mathcal{E}_a^I(\bigcup\{\,\mathcal{L}_k \mid k \geq 0\,\}) \;=\; \bigcup\{\,\mathcal{E}_a^I(\mathcal{L}_k) \mid k \geq 0\,\}.$$

Folglich sind die Funktionen $\mathcal{E}_a^I$ insbesondere monoton.

Beweis
Betrachte zum Beweis die folgende Fallunterscheidung:

Fall 1: $a \notin \mathcal{A}_I$

$$\begin{aligned}
&\mathcal{E}_a^I(\bigcup\{\,\mathcal{L}_k \mid k \geq 0\,\}) \\
&(a \notin \mathcal{A}_I) && = \bigcup\{\,\mathcal{L}_k \mid k \geq 0\,\} \\
&(\mathcal{L}_k = \mathcal{E}_a^I(\mathcal{L}_k),\ \text{da } a \notin \mathcal{A}_I) && = \bigcup\{\,\mathcal{E}_a^I(\mathcal{L}_k) \mid k \geq 0\,\}
\end{aligned}$$

Fall 2: $a \in \mathcal{A}_I$

$$\begin{aligned}
&\bigcup\{\,\mathcal{E}_a^I(\mathcal{L}_k) \mid k \geq 0\,\} \\
&(\mathcal{E}_a^I(\mathcal{L}_k) = (\mathcal{L}_k)_a,\ \text{da } a \in \mathcal{A}_I) && = \bigcup\{\,(\mathcal{L}_k)_a \mid k \geq 0\,\} \\
& && = (\bigcup\{\,\mathcal{L}_k \mid k \geq 0\,\})_a \\
&(a \in \mathcal{A}_I) && = \mathcal{E}_a^I(\bigcup\{\,\mathcal{L}_k \mid k \geq 0\,\})
\end{aligned}$$

Die iterative Prozedur berechnet zu jedem Zustand des betrachteten Transitionssystems approximativ die gewünschten Sprachmengen und besteht aus zwei Schritten:

1. Initial wird der Zustand p mit $\mathcal{L}(I)$ und alle anderen Zustände mit der leeren Menge (bzw. Sprache) beschriftet.

2. Falls ein Zustand q momentan mit $\mathcal{L}$ beschriftet und q' einer seiner a–Nachfolger ist, dann füge $\mathcal{E}_a^I(\mathcal{L})$ zur momentanen Beschriftung von q' hinzu, bis ein Fixpunkt erreicht ist.

Wie in Anhang C beschrieben wird, berechnet diese Prozedur mit Hilfe der (lokalen abstrakten Semantik–) Funktionen $\mathcal{E}_a^I$ einen kleinsten Fixpunkt. Eine effiziente Realisierung dieser Prozedur mittels eines sogenannten *Workset–Algorithmus* ist ebenfalls in Anhang C angegeben. Die zugehörige Zeitkomplexität kann durch das Produkt der Anzahl der Transitionen von p und der Anzahl der Zustände von I abgeschätzt werden. Daß diese Prozedur die gewünschten Sprachmengen berechnet, besagt der folgende Satz:

Satz 2.16 (Korrektheit der Prozedur)
Bzgl. eines Prozesses $p \in \underline{\text{Processes}}$ und einer Interfacespezifikation $I \in \mathcal{I}(p)$ beschriftet die obige Prozedur jeden Zustand $q \in S_p$ mit der Menge $\bigcup \{ \mathcal{L}(i) \mid q \| i \in S_{p\|I} \}$.

Beweis
Die Aussage ergibt sich sofort mit Hilfe des Koinzidenztheorems (vgl. Satz C.3), dessen Voraussetzung wegen Lemma 2.15 erfüllt ist. Das Koinzidenztheorem besagt, daß die durch obige Prozedur berechnete „minimal fixed point"–Lösung mit der gewünschten „join over all paths"–Lösung $\bigcup \{ \mathcal{L}(i) \mid q \| i \in S_{p\|I} \}$ übereinstimmt (vgl. Lemma C.4), falls die betrachteten Semantikfunktionen (hier: $\mathcal{E}_a^I$) additiv sind.

Somit ist durch Proposition 2.13 und obige Prozedur ein effizienter und nach Satz 2.16 korrekter Algorithmus zur Berechnung eines reduzierten Prozesses gegeben.

Die obigen Betrachtungen liefern ein wichtiges theoretisches Resultat, welches in den Beweis von Lemma 2.10 eingegangen ist, nämlich die *Repräsentationsunabhängigkeit*. Grundlegend ist die folgende Beobachtung:
Bei der Konstruktion der IL_q–Mengen, d.h. insbesondere bei der Berechnung von $\mathcal{L}'_I(q)$, wird lediglich die Sprachmenge $\mathcal{L}(I)$ benötigt, welche von der Prozeßstruktur der betrachteten Interfacespezifikation I unabhängig ist. Insbesondere macht es keinen Unterschied, ob I eine deterministische oder eine nichtdeterministische Interfacespezifikation ist. Formal ist somit der folgende Satz gültig:

Satz 2.17 (Repräsentationsunabhängigkeit)
$\forall p \in \underline{\text{Processes}}\ \forall I, I' \in \mathcal{I}(p).\ \mathcal{L}(I) = \mathcal{L}(I')$ *impliziert* $\overline{\Pi}_I(p) = \overline{\Pi}_{I'}(p)$

Weitere Eigenschaften von $\overline{\Pi}$

Um die Übersicht der Eigenschaften des Reduktionsoperators $\overline{\Pi}$, die aus obiger algorithmischer Betrachtung gewonnen werden können, zu erweitern, sei hier die folgende Proposition angegeben:

Proposition 2.18
Für alle Prozesse $p \in$ Processes und Interfacespezifikationen $I, I' \in \mathcal{I}(p)$, sowie für alle Mengen L beobachtbarer Aktionen gilt:

1. $\mathcal{L}(I) \subseteq \mathcal{L}(I')$ *impliziert* $\overline{\Pi}_I(p) \leq \overline{\Pi}_{I'}(p)$, *also insbesondere* $\overline{\Pi}_I(p) \quad \overline{\Pi}_{I'}(p)$.
2. $\mathcal{L}(p\langle \mathcal{A}_I \rangle) \subseteq \mathcal{L}(I)$ *impliziert* $\overline{\Pi}_I(p) = p$.
3. $\overline{\Pi}_I(p\langle \mathcal{A}_I \cup L \rangle) = (\overline{\Pi}_I(p))\langle \mathcal{A}_I \cup L \rangle$.

Beweisskizze
Die erste Aussage ergibt sich als Konsequenz der Monotonie der Funktionen $\mathcal{E}_a^I$ (vgl. Lemma 2.15) und der Tatsache, daß weniger mächtige IL_q–Mengen bzgl. der Halbordnung $\leq$ kleinere Prozesse liefern.
Die zweite Aussage resultiert aus der Beobachtung am Algorithmus, daß eine Reduktion bzgl. einer Interfacespezifikation, deren Sprache $\mathcal{L}(I)$ die Sprache $\mathcal{L}(p\langle \mathcal{A}_I \rangle)$ enthält, keinen Effekt auf p hat. Sie ist also trivial, da in allen IL_q–Mengen zumindest die Transitionsbeschriftungen der q–Nachfolgertransitionen enthalten sind.
Der Grund für die Gültigkeit der dritten Aussage ist, daß sowohl die Aktion τ als auch die Aktionen a mit $a \notin \mathcal{A}_I$ per Definition in jeder IL_q–Menge enthalten sind.

Kapitel 3

Anwendung des Reduktionsoperators

In diesem Kapitel wird gezeigt, wie man einen Reduktionsoperator in eine *Methode* zur kompositionellen Minimierung endlicher verteilter Systeme einbetten kann. Dazu wird eine *kompositionelle* Methode betrachtet, die $\mathcal{RM}$–Methode, deren Korrektheit und Optimalität mit den in Definition 2.1 geforderten Eigenschaften eines Reduktionsoperators bewiesen werden kann. Anschließend wird die $\mathcal{RM}$–Methode bzgl. des speziellen Reduktionsoperators $\overline{\Pi}$ anhand des begleitenden Beispiels demonstriert. Die Mächtigkeit der $\mathcal{RM}$–Methode wird mit Hilfe eines weiteren Beispielsystems verdeutlicht, welches den wechselseitigen Ausschluß beim Zugriff beliebig vieler identischer Prozesse auf eine gemeinsame Ressource modelliert. Mit einer kritischen Betrachtung der Vor– und Nachteile der $\mathcal{RM}$–Methode schließt dieses Kapitel ab.

3.1 Die $\mathcal{RM}$–Methode

In diesem Abschnitt soll die $\mathcal{RM}$–Methode zur Minimierung endlicher verteilter Systeme vorgestellt werden. Zweck der $\mathcal{RM}$–Methode soll es sein, unnötig komplexe Zwischentransitionssysteme während der Konstruktion des minimalen Transitionssystems zu vermeiden. Die $\mathcal{RM}$–Methode soll dazu den folgenden Anforderungen genügen:

- Sie sei auf Systeme der Form $\mathcal{P} = (p_1 \| \cdots \| p_n)\langle L \rangle$ anwendbar, wobei die p_i für $1 \leq i \leq n$ als Transitionssysteme gegeben seien.
  Diese „Standard Concurrent Form“, wie sie in CCS genannt wird, ist für das Problem der Zustandsexplosion verantwortlich. Sie charakterisiert die Prozesse, die für Analyse– und Verifikationsmethoden, welche auf Transitionssystemen arbeiten, kritisch sind.

- Sie nütze lokale und globale Kontextabhängigkeiten aus.
  Das erfordert, daß die Methode zwei Arten der Minimierung berücksichtigt: eine bzgl. des lokalen und eine bzgl. des globalen Kontextes.

1. Die Minimierung bzgl. des jeweiligen lokalen Kontextes geschieht durch Anwendung der Funktion $\mathcal{M}$, welche zu einem Transitionssystem ein $\approx^d$ – äquivalentes Transitionssystem mit (in der Äquivalenzklasse) minimaler Zustandsanzahl konstruiert.
2. Die Minimierung bzgl. des jeweiligen globalen Kontextes geschieht durch Anwendung eines Reduktionsoperators Π, dessen Nutzen im vorangegangenen Kapitel ausführlich beschrieben wurde. Dazu werden Interfacespezifikationen I_i für $1 \leq i < n$ benötigt, die vom Programmentwickler gegeben sein müssen. Die i–te Interfacespezifikation I_i beschreibt dabei die Schnittstelle zwischen den Teilsystemen $(p_1 \| \cdots \| p_i)$ und $(p_{i+1} \| \cdots \| p_n)$.

- Sie sei im folgenden Sinne kompositionell:
  Will man die $\mathcal{RM}$–Methode auf mehrere Systeme mit teilweise gleichen Parallelkomponenten anwenden, etwa

  $$\mathcal{P}_1 = (p_1 \| \cdots \| p_k \| p_{k+1} \| \cdots \| p_m)\langle L \rangle \quad \text{und} \quad \mathcal{P}_2 = (p_1 \| \cdots \| p_k \| p'_{k+1} \| \cdots \| p'_n)\langle L \rangle$$

  und gilt für die betrachteten Interfacespezifikationen

  $$I_i \in \mathcal{I}(p_1 \| \cdots \| p_i \, , \, p_{i+1} \| \cdots \| p_m) \quad \text{und} \quad I_i \in \mathcal{I}(p_1 \| \cdots \| p_i \, , \, p'_{i+1} \| \cdots \| p'_n)$$

  für $1 \leq i \leq k$, so seien die Reduktions– und Minimierungsschritte der $\mathcal{RM}$–Methode bzgl. des Teilsystems $p_1 \| \cdots \| p_k$ sowohl für die Minimierung des Systems $\mathcal{P}_1$ als auch für die des Systems $\mathcal{P}_2$ verwendbar.

- Sie nütze die Korrespondenz zwischen Parallel– und Fensteroperator (vgl. Lemma 1.10) aus.
  Das Ziel dabei ist, ein $\approx^d$ – äquivalentes System zu betrachten, in dem in jeder Parallelkomponente möglichst viele Transitionen mit τ beschriftet sind, damit die Funktion $\mathcal{M}$ möglichst viele Zustände zusammenfassen kann.

- Ergebnis der $\mathcal{RM}$–Methode sei das Transitionssystem $\mathcal{P}_n$, welches stets $\mathcal{P}_n \quad \mathcal{P}$ erfülle (Korrektheit der Methode). Falls die gegebenen Interfacespezifikationen korrekt sind, soll es zusätzlich in derselben $\approx^d$–Äquivalenzklasse wie das Transitionssystem $\mathcal{P}$ liegen und in dieser eine minimale Zustandsanzahl haben (Optimalität der Methode).

Die $\mathcal{RM}$–Methode konstruiert ausgehend von einem System

$$\mathcal{P} \quad = \quad (p_1 \|_{I_1} p_2 \|_{I_2} \cdots \|_{I_{n-1}} p_n \,)\langle L \rangle$$

sukzessive die Zwischentransitionssysteme $\mathcal{P}_i$:

$$\underbrace{\underbrace{(\underbrace{p_1\|_{I_1}}_{\mathcal{P}_1} p_2\|_{I_2}}_{\mathcal{P}_2} \cdots \|_{I_{n-1}} p_n\,)\langle L\rangle}_{\mathcal{P}_n}.$$

Dabei seien die $\mathcal{P}_i$ für $1 \leq i \leq n$ wie folgt definiert:

- $\mathcal{P}_1 =_{df} \mathcal{M}(\Pi_{I_1}(\mathcal{M}(p_1\langle \mathcal{A}_{I_1} \cup L\rangle)))$,
- $\mathcal{P}_i =_{df} \mathcal{M}(\Pi_{I_i}(\mathcal{M}((\mathcal{P}_{i-1}\|p_i)\langle \mathcal{A}_{I_i} \cup L\rangle)))$ für $2 \leq i \leq n-1$ und
- $\mathcal{P}_n =_{df} \mathcal{M}((\mathcal{P}_{n-1}\|p_n)\langle L\rangle)$.

Die $\mathcal{RM}$–Methode basiert prinzipiell auf dem in der Einleitung erläuterten naiven Ansatz, bei dem zusätzlich noch die Parallel–/Fensteroperator–Korrespondenz und ein Reduktionsoperator berücksichtigt werden.

Betrachte die Definition von einem $\mathcal{P}_i$ $(2 \leq i \leq n-1)$ von innen nach außen:
Zunächst wird eine weitere Parallelkomponente hinzugenommen und mit Hilfe des Fensters $\langle \mathcal{A}_{I_i} \cup L\rangle$ (nebst anschließender Anwendung von $\mathcal{M}$) die Parallel–/Fensteroperatorkorrespondenz ausgenutzt. Dann wird der Reduktionsoperator Π zur Reduktion bzgl. des globalen Kontextes und abschließend die Funktion $\mathcal{M}$ zur Minimierung bzgl. des lokalen Kontextes angewandt.

Im folgenden werden theoretische Aspekte der $\mathcal{RM}$–Methode, insbesondere deren Korrektheit und Optimalität, behandelt. Dabei seien $\mathcal{P}$ und $\mathcal{P}_i$ wie oben definiert, sowie für $1 \leq i \leq n$

$$Q_i =_{df} (p_{i+1}\| \cdots \|p_n).$$

Q_n ist dabei der bzgl. des Paralleloperators neutrale Prozeß („neutrales Element"), welcher ein leeres Alphabet und lediglich einen Zustand besitzt.

Für die Korrektheit der $\mathcal{RM}$–Methode spielt folgender Satz, welcher unabhängig von der Korrektheit der Interfacespezifikationen I_i gültig ist, die entscheidende Rolle:

Satz 3.1 (Korrektheit der Methode)
$\forall\, 1 \leq i \leq n.\ (\mathcal{P}_i \| Q_i)\langle L\rangle \quad \mathcal{P}.$

Beweis
Beweis mittels Induktion über i:

Induktionsanfang ($i = 1$):

$$
\begin{array}{lcl}
 & & (\mathcal{P}_1 \| Q_1)\langle L\rangle \\
(\text{Definition von } \mathcal{P}_1) & = & (\mathcal{M}(\Pi_{I_1}(\mathcal{M}(p_1\langle \mathcal{A}_{I_1} \cup L\rangle)))\| Q_1)\langle L\rangle \\
(\text{Prop. 1.19 \& Satz 1.20 \& Definition 2.1 (i)}) & & (p_1\langle \mathcal{A}_{I_1} \cup L\rangle \| Q_1)\langle L\rangle \\
(\text{Lemma 1.10}) & = & (p_1 \| Q_1)\langle L\rangle \\
(\text{Definition von } \mathcal{P}) & = & \mathcal{P}
\end{array}
$$

Gelte nun $(\mathcal{P}_{i-1} \| Q_{i-1})\langle L\rangle \quad \mathcal{P}$ für ein $i > 1$.

Induktionsschritt ($i-1 \longrightarrow i$):

1. Fall: $2 \leq i \leq n-1$

$$
\begin{array}{lcl}
 & & (\mathcal{P}_i \| Q_i)\langle L\rangle \\
(\text{Definition } \mathcal{P}_i) & = & (\mathcal{M}(\Pi_{I_i}(\mathcal{M}((\mathcal{P}_{i-1} \| p_i)\langle \mathcal{A}_{I_i} \cup L\rangle)))\| Q_i)\langle L\rangle \\
(\text{Prop. 1.19 \& Satz 1.20 \& Definition 2.1 (i)}) & & ((\mathcal{P}_{i-1} \| p_i)\langle \mathcal{A}_{I_i} \cup L\rangle \| Q_i)\langle L\rangle \\
(\text{Lemma 1.10}) & = & ((\mathcal{P}_{i-1} \| p_i) \| Q_i)\langle L\rangle \\
(\text{Prop. 1.9}) & = & (\mathcal{P}_{i-1} \| (p_i \| Q_i))\langle L\rangle \\
(\text{Definition von } Q_{i-1}) & = & (\mathcal{P}_{i-1} \| Q_{i-1})\langle L\rangle \\
(\text{Induktionsannahme}) & & \mathcal{P}
\end{array}
$$

2. Fall: $i = n$

$$
\begin{array}{lcl}
 & & (\mathcal{P}_n \| Q_n)\langle L\rangle \\
(\text{Definition von } Q_n) & = & \mathcal{P}_n\langle L\rangle \\
(\text{Definition von } \mathcal{P}_n) & = & (\mathcal{M}((\mathcal{P}_{n-1}\|p_n)\langle L\rangle))\langle L\rangle \\
(\text{Prop. 1.19 \& Satz 1.20 \& Definitionen 2.1 (i) und von } \langle\cdot\rangle) & & (\mathcal{P}_{n-1}\|p_n)\langle L\rangle \\
(\text{Definition von } Q_{n-1}) & = & (\mathcal{P}_{n-1}\|Q_{n-1})\langle L\rangle \\
(\text{Induktionsannahme}) & & \mathcal{P}
\end{array}
$$

Per Induktionsprinzip folgt die Behauptung.

Dieses Lemma garantiert wegen $\mathcal{P}_n \quad \mathcal{P}$ die *Korrektheit der $\mathcal{RM}$–Methode.* Unter Korrektheit der Methode ist zu verstehen, daß eine erfolgreiche Verifikation einer $\approx^d$ – konsistenten Eigenschaft für $\mathcal{P}_n$ die Gültigkeit dieser Eigenschaft für $\mathcal{P}$ impliziert. Deswegen ist der Begriff einer Quasiordnung in diesem Zusammenhang unentbehrlich und geht in die Definition eines Reduktionsoperators mit ein. Inkorrekte Interfacespezifikationen führen also nicht zu falschen Beweisen, sondern können nur zur Folge haben, daß eine für $\mathcal{P}$ gültige Eigenschaft nicht mit Hilfe von $\mathcal{P}_n$ bewiesen werden kann. Um diesen Erfolg zu garantieren, ist die Korrektheit der Interfacespezifikationen eine hinreichende Voraussetzung. In diesem Fall gilt sogar $\mathcal{P}_n \approx^d \mathcal{P}$, wie der Beweis des folgenden Satzes zeigt:

Satz 3.2 (Optimalität der Methode)
$\forall\, 1 \leq i \leq n.\ (\forall j \leq i.\ I_j \in \mathcal{I}(p_1\| \cdots \|p_j, Q_j))$ *impliziert* $(\mathcal{P}_i\|Q_i)\langle L\rangle \approx^d \mathcal{P}$.

Beweis
Beweis des Satzes mittels Induktion über i:

Induktionsanfang ($i = 1$):

$$
\begin{array}{lcl}
 & & (\mathcal{P}_1\|Q_1)\langle L\rangle \\
(\text{Definition von } \mathcal{P}_1) & = & (\mathcal{M}(\Pi_{I_1}(\mathcal{M}(p_1\langle \mathcal{A}_{I_1}\cup L\rangle)))\|Q_1)\langle L\rangle \\
(\text{Definition von } \mathcal{M} \text{ \& Satz 1.20}) & \approx^d & (\Pi_{I_1}(\mathcal{M}(p_1\langle \mathcal{A}_{I_1}\cup L\rangle))\|Q_1)\langle L\rangle \\
(\text{Proposition 2.2 \& Satz 1.20}) & \approx^d & (\Pi_{I_1}(p_1\langle \mathcal{A}_{I_1}\cup L\rangle)\|Q_1)\langle L\rangle \\
(\text{Def. 2.1 (ii) \& Lemma 1.26 \& Satz 1.20}) & \approx^d & (p_1\langle \mathcal{A}_{I_1}\cup L\rangle\|Q_1)\langle L\rangle
\end{array}
$$

$$\begin{aligned}
(\text{Lemma 1.10}) \quad &= \quad (p_1 \| Q_1)\langle L\rangle \\
(\text{Definition von } \mathcal{P}) \quad &= \quad \mathcal{P}
\end{aligned}$$

Gelte nun für ein $i > 1$:

$$\forall j < i.\ I_j \in \mathcal{I}(p_1 \| \cdots \| p_j, Q_j) \textit{ impliziert } (\mathcal{P}_{i-1} \| Q_{i-1})\langle L\rangle \approx^d \mathcal{P}.$$

Induktionsschritt ($i-1 \longrightarrow i$):

Hier wird die folgende Hilfsaussage benötigt:
$(*)$ $\quad I_i \in \mathcal{I}(p_1 \| \cdots \| p_i, Q_i)$ impliziert $I_i \in \mathcal{I}(\mathcal{M}((\mathcal{P}_{i-1} \| p_i)\langle \mathcal{A}_{I_i} \cup L\rangle), Q_i)$

Die Hilfsaussage folgt mit Definition 1.23 aus

$$\begin{aligned}
\mathcal{L}(I_i) \quad &\supseteq \quad \mathcal{L}(((p_1 \| \cdots \| p_i) \| Q_i)\langle \mathcal{A}_{I_i}\rangle) \\
(\text{Def. } Q_i\text{, Prop. 1.9, Satz 1.20 \& Lemma 1.22}) \quad &= \quad \mathcal{L}((p_1 \| \cdots \| p_n)\langle \mathcal{A}_{I_i}\rangle) \\
(\text{Ind.–Vor. für } L = \mathcal{A}_{I_i} \text{ \& Lemma 1.22}) \quad &= \quad \mathcal{L}((\mathcal{P}_{i-1} \| Q_{i-1})\langle \mathcal{A}_{I_i}\rangle) \\
(\text{Def. } Q_{i-1}\text{, Prop. 1.9, Satz 1.20 \& Lemma 1.22}) \quad &= \quad \mathcal{L}(((\mathcal{P}_{i-1} \| p_i) \| Q_i)\langle \mathcal{A}_{I_i}\rangle) \\
(\text{Lemmata 1.10 \& 1.22}) \quad &= \quad \mathcal{L}(((\mathcal{P}_{i-1} \| p_i)\langle \mathcal{A}_{I_i} \cup L\rangle \| Q_i)\langle \mathcal{A}_{I_i}\rangle) \\
(\text{Definition von } \mathcal{M}\text{, Satz 1.20 \& Lemma 1.22}) \quad &= \quad \mathcal{L}((\mathcal{M}((\mathcal{P}_{i-1} \| p_i)\langle \mathcal{A}_{I_i} \cup L\rangle) \| Q_i)\langle \mathcal{A}_{I_i}\rangle)
\end{aligned}$$

1. Fall: $2 \leq i \leq n-1$

$$\begin{aligned}
& (\mathcal{P}_i \| Q_i)\langle L\rangle \\
(\text{Definition } \mathcal{P}_i) \quad &= \quad (\mathcal{M}(\Pi_{I_i}(\mathcal{M}((\mathcal{P}_{i-1} \| p_i)\langle \mathcal{A}_{I_i} \cup L\rangle))) \| Q_i)\langle L\rangle \\
(\text{Definition von } \mathcal{M} \text{ \& Satz 1.20}) \quad &\approx^d \quad (\Pi_{I_i}(\mathcal{M}((\mathcal{P}_{i-1} \| p_i)\langle \mathcal{A}_{I_i} \cup L\rangle)) \| Q_i)\langle L\rangle \\
((*) \text{ \& Definition 2.1 (ii) \& Satz 1.20}) \quad &\approx^d \quad (\mathcal{M}((\mathcal{P}_{i-1} \| p_i)\langle \mathcal{A}_{I_i} \cup L\rangle) \| Q_i)\langle L\rangle \\
(\text{Definition von } \mathcal{M}) \quad &\approx^d \quad ((\mathcal{P}_{i-1} \| p_i)\langle \mathcal{A}_{I_i} \cup L\rangle \| Q_i)\langle L\rangle \\
(\text{Lemma 1.10}) \quad &= \quad ((\mathcal{P}_{i-1} \| p_i) \| Q_i)\langle L\rangle \\
(\text{Prop. 1.9 \& Def. von } Q_{i-1} \text{ \& Satz 1.20}) \quad &\approx^d \quad (\mathcal{P}_{i-1} \| Q_{i-1})\langle L\rangle \\
(\text{Induktionsannahme}) \quad &\approx^d \quad \mathcal{P}
\end{aligned}$$

2. Fall: $i = n$

$$
\begin{array}{lcl}
 & & (\mathcal{P}_n \| Q_n)\langle L\rangle \\
(\text{Definition von } Q_n) & = & \mathcal{P}_n\langle L\rangle \\
(\text{Definition von } \mathcal{P}_n) & = & (\mathcal{M}((\mathcal{P}_{n-1}\|p_n)\langle L\rangle))\langle L\rangle \\
(\text{Definition von } \mathcal{M} \text{ bzw. } \langle\cdot\rangle \text{ und Satz 1.20}) & \approx^d & (\mathcal{P}_{n-1}\|p_n)\langle L\rangle \\
(\text{Definition von } Q_{n-1}) & = & (\mathcal{P}_{n-1}\|Q_{n-1})\langle L\rangle \\
(\text{Induktionsannahme}) & \approx^d & \mathcal{P}
\end{array}
$$

Per Induktionsprinzip folgt die Behauptung.

Wie in Kapitel 2 bereits erläutert, ist $\mathcal{P}$ in der Praxis meist total definiert, und in diesem Fall liefert ein Reduktionsoperator nach Definition 2.1 (ii) ein im Kontext äquivalentes System. Dies ist der Grund für die Gültigkeit des folgenden Korollars:

Korollar 3.3
Sei $\mathcal{P}$ total definiert. Dann gilt: $\mathcal{P}_n \approx^d \mathcal{P}$ genau dann, wenn $\mathcal{P}_n$ total definiert ist.

Beweis
Zum Beweis sind zwei Richtungen zu zeigen:

„ $\Rightarrow$ “: Gelte $\mathcal{P}_n \approx^d \mathcal{P}$.
Da $\mathcal{P}$ nach Voraussetzung total definiert ist und nach Definition 1.12 nur solche Prozesse $\approx^d$ – äquivalent sein können, deren Undefiniertheitsprädikate gleich sind, ist auch $\mathcal{P}_n$ total definiert.

„ $\Leftarrow$ “: Es gilt nach Voraussetzung:

- $\mathcal{P}$, $\mathcal{P}_n$ sind total definiert.
- $\mathcal{P}_n \quad \mathcal{P}$ (nach Satz 3.1 für $i = n$, beachte $\mathcal{P}_n\langle L\rangle = \mathcal{P}_n$).

$\Rightarrow$ (Proposition 1.19) $\mathcal{P}_n \approx^d \mathcal{P}$.

Folglich reduziert sich der Beweis von $\mathcal{P}_n \approx^d \mathcal{P}$ in der Praxis auf das Überprüfen der totalen Definiertheit von $\mathcal{P}_n$, so daß mit der $\mathcal{RM}$–Methode nicht–kontexttreue Reduktionen, die aus inkorrekten Interfacespezifikationen resultieren, anhand von in $\mathcal{P}_n$ auftretenden Undefiniertheiten erkannt werden können.

Die folgenden Anwendungsbeispiele basieren auf der $\mathcal{RM}$–Methode bzgl. des speziellen Reduktionsoperators $\overline{\Pi}$.

3.2 Anwendung der $\mathcal{RM}$–Methode auf das Beispielsystem

In diesem Abschnitt wird die $\mathcal{RM}$–Methode zunächst auf das begleitende Beispiel 1.1 angewandt.

Nach der in Abschnitt 3.1 vorgestellten $\mathcal{RM}$–Methode, wobei hier speziell $\Pi = \overline{\Pi}$ gewählt wird, sind die wie folgt definierten Prozesse $\mathcal{P}_1$, $\mathcal{P}_2$ und $\mathcal{P}_3$ zu berechnen:

$$\begin{aligned}
\mathcal{P}_1 &= \mathcal{M}(\overline{\Pi}_{I_1}(\mathcal{M}(P_1\langle\{tk1, tk2, rb1, sb1\}\rangle))),\\
\mathcal{P}_2 &= \mathcal{M}(\overline{\Pi}_{I_2}(\mathcal{M}((\mathcal{P}_1\|B)\langle\{tk1, tk2, rb2, sb2\}\rangle))) \text{ und}\\
\mathcal{P}_3 &= \mathcal{M}((\mathcal{P}_2\|P_2)\langle\{tk1, tk2\}\rangle).
\end{aligned}$$

Durchführung der Reduktion

- Berechnung von $\mathcal{P}_1$:

  Man geht von dem Prozeß $P_1\langle\{tk1, tk2, rb1, sb1\}\rangle$ aus, welcher in Abbildung 3.1 dargestellt ist. Die Anwendung des inneren $\mathcal{M}$–Operators läßt diesen Prozeß unverän-

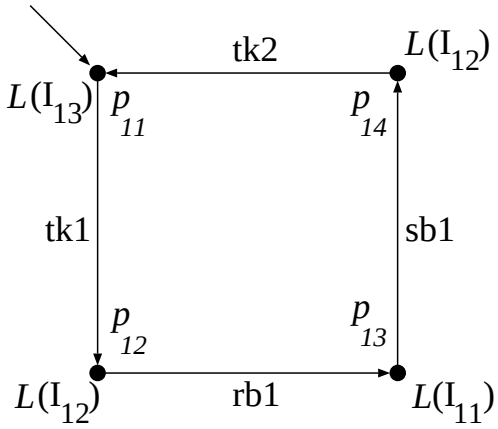


Abbildung 3.1: $\mathcal{P}_1$

dert, ist also trivial. Nun wendet man darauf den Reduktionsoperator $\overline{\Pi}$ bzgl. I_1 an, der die in einem durch I_1 spezifizierten Kontext unerreichbaren Zustände und Transitionen „abschneiden“ soll. Man führt dazu den in Abschnitt 2.3 entwickelten Algorithmus durch. Zunächst werden die Zustände p_{11}, p_{12}, p_{13} und p_{14} gemäß Satz 2.16 und der auf Seite 52 vorgestellten Prozedur unter Benutzung der Funktionen $\mathcal{E}_a^{I_1}$ mit jeweils einer Sprachmenge beschriftet. Das Ergebnis dieser Beschriftung ist in Abbildung 3.1 bereits eingetragen. Die gesuchten Mengen IL_q erhält man jeweils als Menge der Präfixe der Länge 1 der entsprechenden Sprachmengenbeschriftungen: $IL_{p_{11}} = \{tk1, \epsilon\}$, $IL_{p_{12}} = \{rb1, tk2, \epsilon\}$, $IL_{p_{13}} = \{sb1, \epsilon\}$ und $IL_{p_{14}} = \{rb1, tk2, \epsilon\}$. $IL_{p_{12}}$ bzw. $IL_{p_{14}}$ enthalten die für die Reduktion unnötigen Elemente $tk2$ bzw. $rb1$, die aufgrund der bzgl. der Sprache „größer“ als erforderlich definierten Interfacespezifikation I_1 als „Redundanz“ auftreten. Damit ergibt sich nach dem Algorithmus

aus Proposition 2.13 eine triviale Reduktion,[1] d.h. hier sind alle Zustände und Transitionen auch in einem durch I_1 spezifizierten Kontext erreichbar. Die abschließende Anwendung des äußeren $\mathcal{M}$–Operators ist wiederum trivial, d.h. es gilt $\mathcal{P}_1 = P_1$.

- Berechnung von $\mathcal{P}_2$:

Dazu ist zunächst der Prozeß $(\mathcal{P}_1 \| B)\langle\{tk1, tk2, rb2, sb2\}\rangle$ zu konstruieren, der in Abbildung 3.2 dargestellt ist. Die Anwendung des inneren $\mathcal{M}$–Operators liefert die

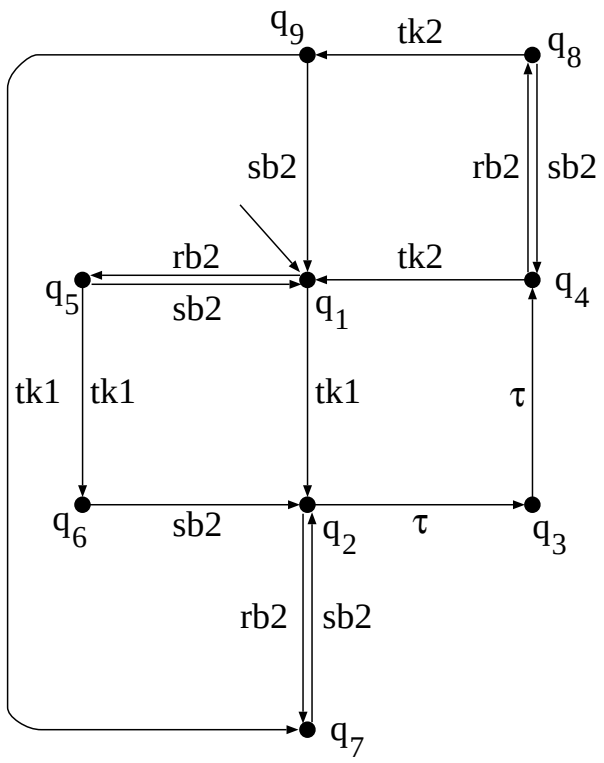


Abbildung 3.2: $(\mathcal{P}_1 \| B)\langle\{tk1, tk2, rb2, sb2\}\rangle$

Äquivalenz der Zustände q_6 und q_7, q_5 und q_9, sowie q_3 und q_4, wodurch der beobachtungsäquivalente Prozeß aus Abbildung 3.3 entsteht.[2] Für dessen Reduktion bzgl. I_2, die bereits in Beispiel 2.4 vorgeführt wurde, sind wiederum die IL_q–Mengen nach dem oben erwähnten Schema zu berechnen. Die Zustandsbeschriftungen sind dabei bereits in Abbildung 3.3 eingetragen, und man erhält $IL_{q_1} = \{tk1, rb2, \epsilon\}$, $IL_{q_2} = \{tk2, \epsilon\} = IL_{q_4}$, $IL_{q_{59}} = \{sb2, \epsilon\}$ und $IL_{q_{67}} = \emptyset = IL_{q_8}$.
Die Zustände q_8 und q_{67} sind aufgrund ihrer Beschriftung mit der leeren Menge in einem durch I_2 spezifizierten Kontext nicht erreichbar und fallen deshalb weg, ebenso die von diesen Zuständen ausgehenden und die zu diesen hinführenden Transitionen, wobei die letzteren durch entsprechende Undefiniertheiten ersetzt werden.

[1] Beachte, daß man i.allg. einen bzgl. kleineren Prozeß erhält.

[2] Im folgenden werden die so zusammengefaßten Zustände mit q_{67}, q_{59} bzw. q_{34} bezeichnet.

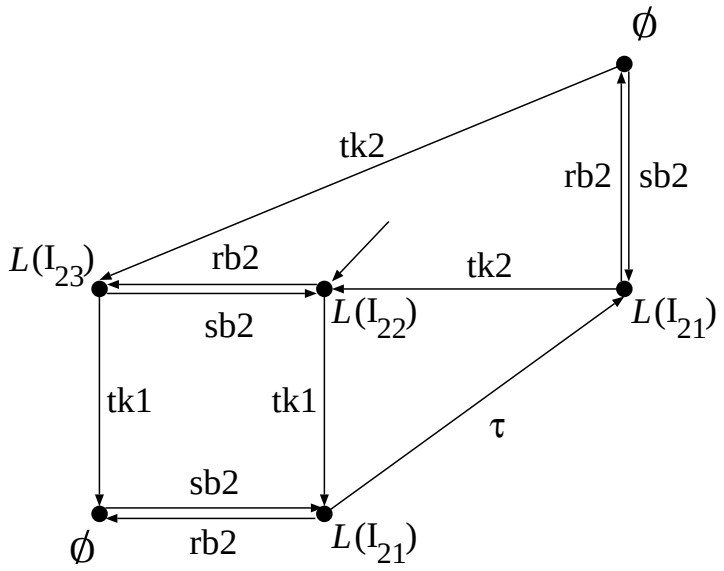


Abbildung 3.3: $\mathcal{M}((\mathcal{P}_1 \| B)\langle\{tk1, tk2, rb2, sb2\}\rangle)$

Das Ergebnis der Anwendung des Reduktionsoperators $\overline{\Pi}_{I_2}$ gibt Abbildung 3.4 wieder. Durch diese Reduktion ist ein bzgl. echt kleineres System entstanden. Ab-

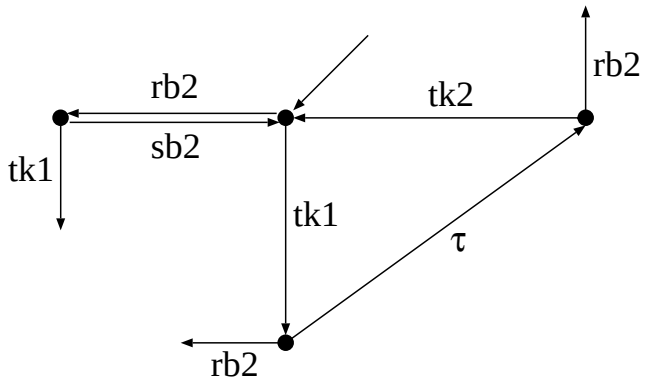


Abbildung 3.4: Nach Anwendung von $\overline{\Pi}_{I_2}$

bildung 3.5 stellt den Prozeß $\mathcal{P}_2$ dar, wobei durch die Anwendung des äußeren $\mathcal{M}$–Operators auf den Prozeß aus Abbildung 3.4 die Zustände q_3 und q_4 zu einem Zustand verschmelzen.

- <u>Berechnung von $\mathcal{P}_3$:</u>

Die Konstruktion von $(\mathcal{P}_2 \| P_2)\langle\{tk1, tk2\}\rangle$ ergibt den Prozeß aus Abbildung 3.6. Man beachte hierbei, daß die bei der Konstruktion von $\mathcal{P}_2$ entstandenen Undefiniertheiten verschwunden sind, da ihre Ausführung gemäß Definition 1.8 vom Prozeß P_2 verhindert wird. Eine letzte Anwendung des $\mathcal{M}$–Operators läßt auch noch die τ–Transitionen verschwinden, und man erhält für $\mathcal{P}_3$ den Prozeß aus Abbildung 3.7.

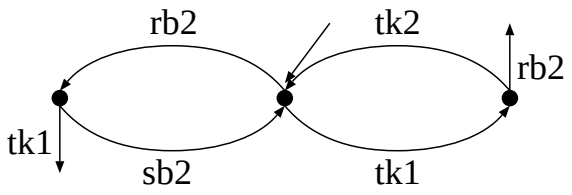


Abbildung 3.5: $\mathcal{P}_2$

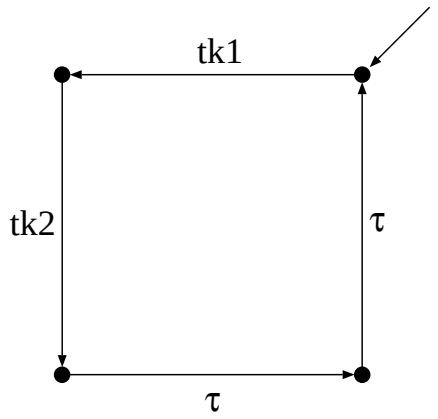


Abbildung 3.6: $(\mathcal{P}_2 \| P_2)\langle\{tk1, tk2\}\rangle$

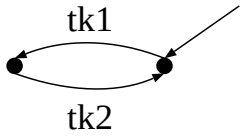


Abbildung 3.7: $\mathcal{P}_3$

Da Sys und $\mathcal{P}_3$ total definiert sind, gilt nach Korollar 3.3 $\mathcal{P}_3 \approx^d Sys$. Das Ergebnis der Reduktion deckt sich mit der intuitiven Vorstellung, daß nach außen nur das „Kreisen" des Tokens über die Kanäle $tk1$ und $tk2$ sichtbar ist.

Betrachtung des Beispiels für eine inkorrekte Interfacespezifikation I_2'

Betrachte die inkorrekte Interfacespezifikation I_2' für $P_1 \| B$ und P_2 aus Abbildung 3.8, die aus I_2 durch Vertauschen der Aktionen $rb2$ und $tk2$ entsteht. I_2' ist nicht korrekt, da sie den geforderten wechselseitigen Ausschluß nicht garantiert, denn: I_2' beschreibt eine Situation, bei der das Token an den Prozeß P_1 weitergereicht wird, bevor P_2 die Nachricht aus dem Buffer B gelesen hat. Formal:
Es gilt $\mathcal{L}(((P_1 \| B) \| P_2)\langle \mathcal{A}_{P_1 \| B} \cap \mathcal{A}_{P_2} \rangle) \not\subseteq \mathcal{L}(I_2')$, denn für $w =_{df} tk1 \cdot tk2 \cdot rb2 \cdot sb2$ ist $w \in \mathcal{L}(((P_1 \| B) \| P_2)\langle \mathcal{A}_{P_1 \| B} \cap \mathcal{A}_{P_2} \rangle)$ aber $w \notin \mathcal{L}(I_2')$. Also ist I_2' gemäß Definition 1.23 inkorrekt.

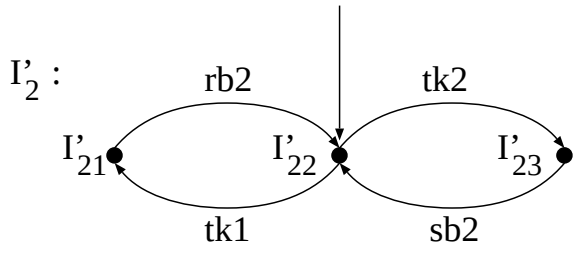


Abbildung 3.8: Inkorrekte Interfacespezifikation I_2'

Für die Sprache von I_2' gilt:

$$\mathcal{L}(I_2') = \mathcal{L}(I_{22}') = tk1 \cdot rb2 \cdot \mathcal{L}(I_{22}') \cup tk2 \cdot sb2 \cdot \mathcal{L}(I_{22}') \cup \{tk1 \cdot \epsilon\} \cup \{tk2 \cdot \epsilon\} \cup \{\epsilon\}.$$

Selbstverständlich ist die Durchführung der Methode bis zum Schritt der Reduktion bzgl. I_2' mit obigem Vorgehen identisch:

- Berechnung von $\mathcal{P}_2$ bzgl. I_2':

  $\mathcal{M}((\mathcal{P}_1 \| B)\langle\{tk1, tk2, rb2, sb2\}\rangle)$ besitzt, wie bereits oben gesehen, die Gestalt aus Abbildung 3.9. Als nächstes sind die IL_q–Mengen zur Reduktion des Prozesses aus

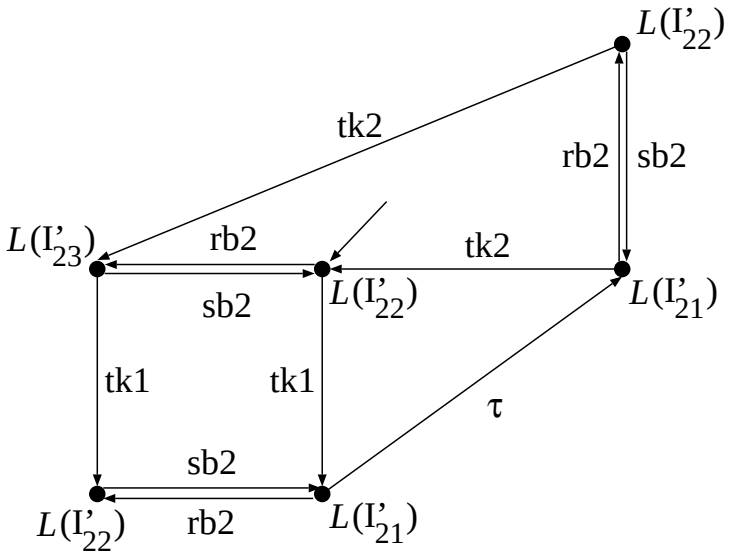


Abbildung 3.9: $\mathcal{M}((\mathcal{P}_1 \| B)\langle\{tk1, tk2, rb2, sb2\}\rangle)$

Abbildung 3.9 bzgl. der inkorrekten Interfacespezifikation I_2' zu bestimmen:[3]
Man erhält $IL_{q_1} = \{tk1, tk2, \epsilon\} = IL_{q_{67}} = IL_{q_8}$, $IL_{q_2} = \{rb2, \epsilon\} = IL_{q_4}$ und $IL_{q_{59}} = \{sb2, \epsilon\}$. Die Anwendung des Reduktionsoperators $\overline{\Pi}$ bzgl. I_2' führt auf den Prozeß aus Abbildung 3.10. Durch diese Reduktion ist wie oben ein bzgl. echt kleineres System entstanden. Es sind Zustände und Transitionen, die in einem durch

[3] Dazu sind in Abbildung 3.9 bereits die Zustände q mit den Mengen $\mathcal{L}_q$ bzgl. I_2' beschriftet.

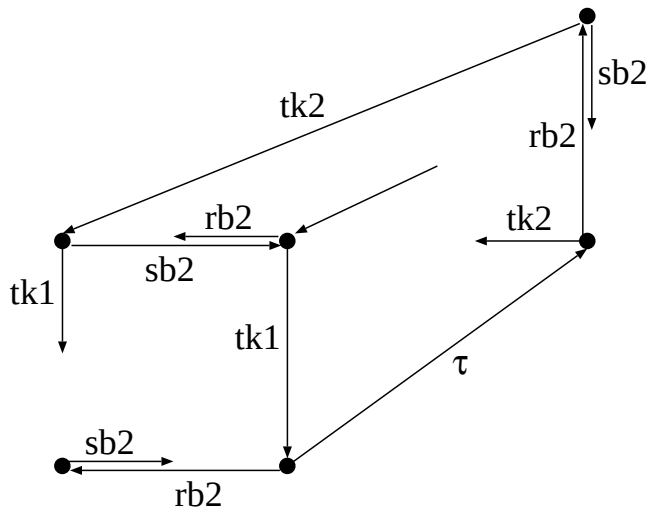


Abbildung 3.10: Nach Anwendung von $\overline{\Pi}_{I_2'}$

I_2' spezifizierten Kontext als nicht möglich erkannt wurden, weggefallen. Letztere haben entsprechende Undefiniertheiten erzeugt (vgl. Definition 2.3).
Die abschließende Anwendung des $\mathcal{M}$–Operators ist trivial, d.h. es werden keine Zustände zusammengefaßt. Also ist $\mathcal{P}_2$ bereits durch Abbildung 3.10 gegeben.

- Berechnung von $\mathcal{P}_3$:

  Die Hinzunahme der letzten Komponente P_2 ergibt den zu bestimmenden Prozeß $\mathcal{P}_3$ (vgl. Abbildung 3.11).[4] Beachte hierbei, daß die in $\mathcal{P}_2$ entstandenen Undefiniertheiten

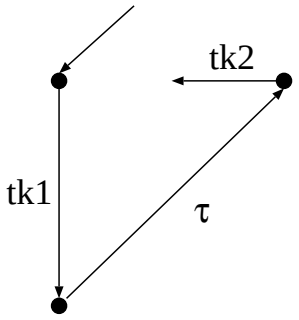


Abbildung 3.11: $\mathcal{P}_3$

  nicht vollständig verschwunden sind, da die Ausführung der Undefiniertheit $tk2$ im Kontext mit P_2 nicht verhindert wird (vgl. Definition 1.8).

Da das betrachtete System Sys total definiert ist, $\mathcal{P}_3$ jedoch nicht, gilt zunächst nach Korollar 3.3 $\mathcal{P}_3 \not\approx^d Sys$. Mit Hilfe von Satz 3.2 (für $i = n$) läßt sich auf die Inkorrektheit einer Interfacespezifikation schließen.[5]

[4] Die abschließende Anwendung des $\mathcal{M}$–Operators ist trivial.

[5] Beachte, daß eine inkorrekte Interfacespezifikation nicht zwingend dazu führt, daß $\mathcal{P}_n \not\approx^d Sys$ gilt. Dies

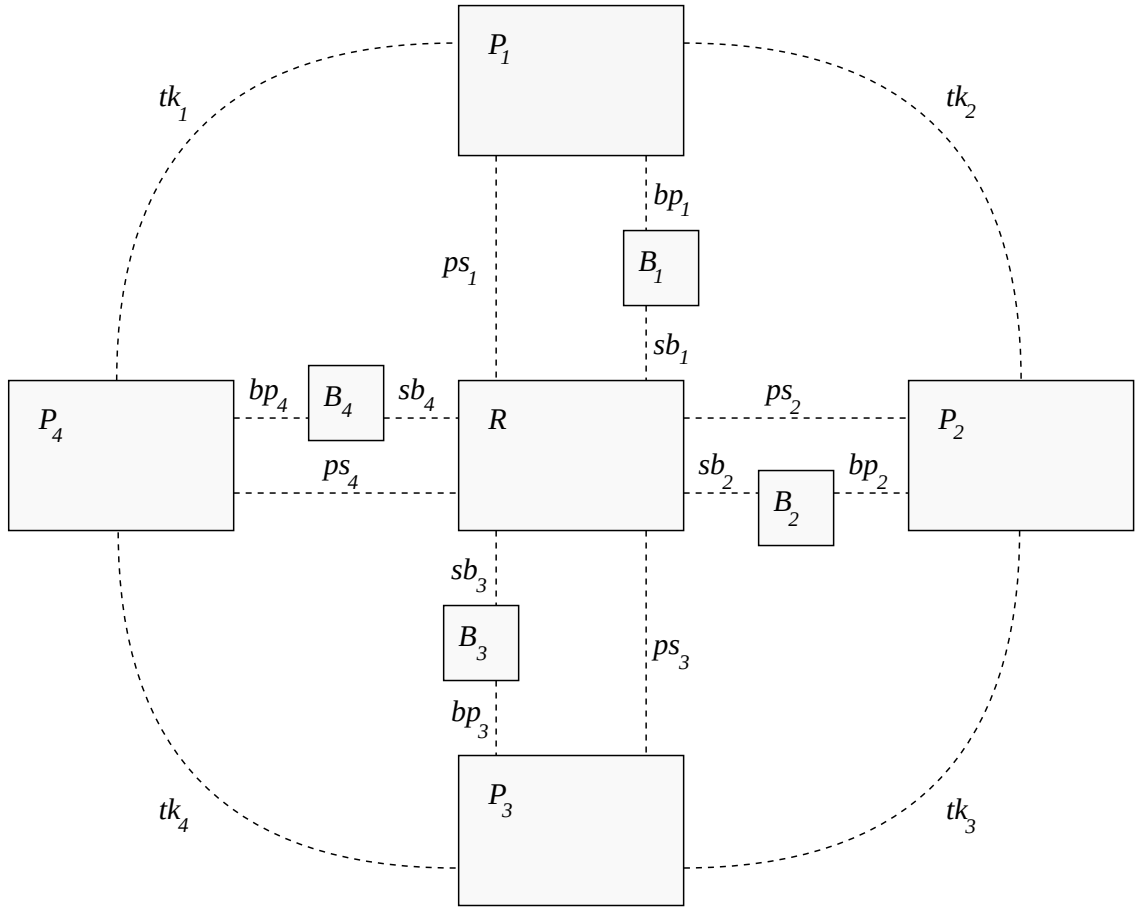
Der eigentliche Sinn der $\mathcal{RM}$–Methode, nämlich die algorithmische Komplexität der realen anzunähern, ist im Beispiel leider nur schwer nachvollziehbar. Daher wird im folgenden ein System vorgestellt, dessen scheinbare und algorithmische Komplexität exponentiell sind, obwohl seine reale Komplexität linear ist.

3.3 Mächtigkeit der $\mathcal{RM}$–Methode

Das hier vorzustellende Beispielsystem beschreibt den Zugriff von n Prozessen P_i (für $1 \leq i \leq n$) auf eine gemeinsame Ressource R nach dem *Round–Robin–Verfahren.* Abbildung 3.12 illustriert ein solches System für $n = 4$. Der *wechselseitige Ausschluß* beim Zugriff auf die gemeinsame Ressource wird durch ein *Token* garantiert, welches mit Hilfe der Kommunikationskanäle tk_i zwischen den Prozessen herumgereicht wird. Dabei darf nur derjenige Prozeß P_k auf die Ressource R zugreifen, der momentan das Token besitzt. Dieser Prozeß sendet via ps_k eine Anfrage an die Ressource, welche durch die Übertragung des gewünschten (Daten–)Objektes antwortet. Der zugehörige Übertragungsweg wird durch einen *Buffer* B_k modelliert, da dies für die Übertragung umfangreicher Objekte (im vgl. zu einer Modellierung mittels einer atomaren *„Handshake"*–Kommunikation) realistischer ist.
Angenommen, man möchte nun beweisen, daß dieses Modell für den „Round–Robin" – Zugriff unserer Intuition entspricht (also das Token zwischen den Prozessen „ringförmig" herumgeschoben wird). Zu diesem Zweck genügt es, nur das Herumreichen des Tokens nach außen sichtbar zu machen, und zu zeigen, daß der daraus resultierende Prozeß $System(n) =_{df} (R \| P_1 \| B_1 \| \cdots \| P_n \| B_n)\langle\{tk_1, .., tk_n\}\rangle$ zu dem Prozeß $Spec(n)$, der ausschließlich die Aktionensequenz $tk_1, \cdots, tk_n$ wiederholt ausführt, beobachtungs– oder sogar $\approx^d$ – äquivalent ist, also $System(n) \approx^d Spec(n)$ gilt.

Es ist leicht einzusehen, daß die scheinbare Komplexität des Systems $System(n)$ in der Anzahl der Komponenten n exponentiell ist, denn: Jeder Prozeß P_i kann, nachdem er eine Anfrage an die Ressource R gerichtet und diese ihm das gewünschte Objekt in seinen Buffer B_i geschrieben hat, das Token an den nächsten Prozeß P_{i+1} weiterreichen (d.h. die Aktion tk_{i+1} ausführen) und die Nachricht irgendwann (mittels der Aktion bp_i) aus seinem Buffer B_i lesen. Dies führt dazu, daß die Aktionen bp_i (für $1 \leq i \leq n$) in beliebiger Reihenfolge ausgeführt werden dürfen und damit zu einer fakultativen Zustandsexplosion. Diese kann jedoch alleine unter Berücksichtigung der Parallel–/Fensteroperatorkorrespondenz vermieden werden, falls man vor einer Minimierung des Systems bereits die Komponenten P_i und B_i zusammenfaßt, d.h. $P_i \| B_i$ für $1 \leq i \leq n$ berechnet, da auf der Aktion bp_i nur zwischen den Parallelkomponenten P_i und B_i kommuniziert wird. Das System eignet sich dennoch für die Demonstration der Mächtigkeit der in dieser Arbeit vorgestellten $\mathcal{RM}$–Methode, da sich eine weitere Ursache für eine Zustandsexplosion bei der

ist genau dann nicht der Fall, wenn die betreffenden Transitionen einer Interfacespezifikation, die deren Inkorrektheit verursachen, bzgl. des bei der Reduktion betrachteten Kontextes nicht „ausgeführt" werden können und deshalb für die Reduktion unerheblich sind.



mit

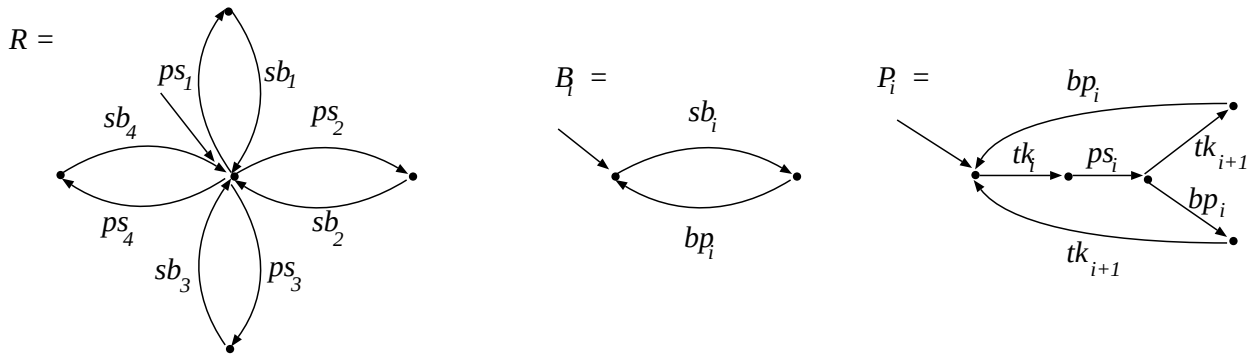


Abbildung 3.12: „Round Robin“ - Zugriff

Konstruktion des globalen Transitionssystems finden läßt: Egal wann wir bei der Konstruktion die Ressource R berücksichtigen, können die Aktionen sb_i der Buffer B_i in sehr willkürlicher Reihenfolge ausgeführt werden. Erst wenn man mehrere in ihrer Indizierung aufeinanderfolgende Komponenten P_j (und B_j) betrachtet hat, schränken sich die

n	scheinbare Komplexität		alg. Komplexität		reale Komplexität	
	Zust.	Trans.	Zust.	Trans.	Zust.	Trans.
4	144	368	20	29	4	4
5	361	1101	24	35	5	5
6	865	3073	28	41	6	6
7	2017	8177	32	47	7	7

Tabelle 3.1: Numerische Belege für die Effizienz der $\mathcal{RM}$–Methode

Interaktionsmöglichkeiten der Aktionen sb_i ein. Auch hier kann folglich eine fakultative Zustandsexplosion beobachtet werden, die globaler Natur ist und nicht durch eine Berücksichtigung der Parallel–/Fensteroperatorkorrespondenz verhindert werden kann. Folglich ist die scheinbare Komplexität von $System(n)$ exponentiell.
Im Gegensatz dazu ist die reale Komplexität von $System(n)$ in n linear, was durch das Erreichen des Beweisziels bestätigt werden wird. Ebenso ist es möglich, eine in n lineare algorithmische Komplexität zu erreichen, falls man für die $\mathcal{RM}$–Methode die exakten Interfacespezifikationen I_i (für $1 \leq i < n$) berücksichtigt und die Parallelkomponenten des Systems $System(n)$ wie folgt zusammenfaßt:

$$(\overbrace{R \| P_1 \| B_1} \|_{I_1} \overbrace{P_2 \| B_2} \|_{I_2} \cdots \|_{I_{n-1}} \overbrace{P_n \| B_n})\langle\{tk_1, .., tk_n\}\rangle$$

Tabelle 3.1 gibt für dieses Beispiel numerische Belege für die Effizienz der Methode. Die Tabelle wurde mit Hilfe des *Aldébaran Verification Tools* [Fer88] erstellt. Sie zeigt die Größe des globalen Transitionssystems (scheinbare Komplexität), die maximale Größe eines Zwischentransitionssystems während der Konstruktion des minimalen Transitionssystems (algorithmische Komplexität) und die Größe des minimalen Transitionssystems (reale Komplexität). Dabei wurde zur Reduktion, wie bereits angedeutet, die exakten Interfacespezifikationen herangezogen.

Um zu sehen, daß sich die Anstrengungen zur Entwicklung der $\mathcal{RM}$–Methode gelohnt haben, sind die Vergleichszahlen bzgl. der algorithmischen Komplexität in Tabelle 3.2 angegeben, die man erhält, wenn man den naiven Ansatz, der in der Einleitung beschrieben wurde, verfolgt.

Diese Gegenüberstellung zeigt die Wichtigkeit von Interfacespezifikationen für automatische Beweistechniken, da die zusätzliche Information entscheidend zur Minimierung endlicher verteilter Systeme beiträgt.

n	Zust.	Trans.
4	96	243
5	324	927
6	972	3024
7	2916	9801

Tabelle 3.2: Zahlen für die „naive“ Methode

3.4 Bewertung der $\mathcal{RM}$–Methode

Im folgenden sollen die Vor– und Nachteile der im Abschnitt 3.1 vorgestellten $\mathcal{RM}$–Methode herausgestellt werden. Ein Abwägen der Vor– und Nachteile macht wenig Sinn, weil es sich bei dieser Reduktionsmethode um eine Heuristik handelt, die dazu dient, Eigenschaften eines Systems auch dann noch verifizieren zu können, wenn das volle Transitionssystem auf einem Rechner nicht generiert werden kann. Es gilt vielmehr zu erkennen, an welchen Stellen die Stärken und Schwächen der Methode liegen und an welchen Stellen Verbesserungen wünschenswert wären, so daß zumindest in Spezialfällen Nachteile vermieden und Vorteile beibehalten werden können.

Vorteile:

- Kompositionalität
- Die Korrektheit der $\mathcal{RM}$–Methode ist von der Korrektheit der Interfacespezifikationen unabhängig.
- Die Korrektheit der Interfacespezifikationen muß nicht explizit verifiziert werden. Interfacespezifikationen können sogar geraten werden.
- Eine nichtoptimale Reduktion (aufgrund inkorrekter Interfacespezifikationen) wird automatisch angezeigt, falls man von einem total definierten System ausgeht.
- Es gibt eine einfache und effiziente algorithmische Realisierung der $\mathcal{RM}$–Methode für spezielle Reduktionsoperatoren (beispielsweise für $\overline{\Pi}$).

Nachteile:

- Die Interfacespezifikationen müssen vom Programmentwickler bereitgestellt werden.
- Die Güte der Reduktion, d.h. wie niedrig die algorithmische Komplexität liegt, hängt von der Güte der Interfacespezifikationen ab, d.h. wie nahe die Sprachen der angegebenen Interfacespezifikationen an diejenigen der exakten Interfacespezifikationen herankommen.

- Das für die $\mathcal{RM}$–Methode benötigte zusätzliche Hilfsprädikat $\uparrow$ bewirkt, daß Äquivalenzklassen bzgl. der Beobachtungsäquivalenz $\approx$ nach Milner, der primär das Interesse gilt, evtl. in jeweils mehrere Äquivalenzklassen bzgl. $\approx^d$ aufgesplittet werden.

Insgesamt gesehen besitzt die $\mathcal{RM}$–Methode viele wünschenswerte Eigenschaften, jedoch bekommt man die Methode nicht „geschenkt". Der Preis besteht darin, die benötigten Interfacespezifikationen angeben zu müssen, wobei die Güte der Reduktion von der Güte der Interfacespezifikationen abhängt. Das Beispiel aus Abschnitt 3.3 zeigt jedoch die Wichtigkeit von Interfacespezifikationen für automatische Beweistechniken. Softwareentwickler sollten daher die Angabe von Interfacespezifikationen als Teil der Implementation betrachten, die neben einer Ermöglichung der automatischen Verifikation auch der *strukturierten Programmierung* dient. Eine analoge Situation ist beispielsweise die für *While–Programme.* Dort hängt eine automatische Verifikation von dem Vorhandensein der *Schleifeninvarianten* ab, die ebenfalls vom Programmierer angegeben werden müssen.

Kapitel 4

Die Methode für Spezialfälle

In diesem Kapitel wird untersucht, inwieweit die in Kapitel 3 angemerkten Nachteile der $\mathcal{RM}$–Methode zur kompositionellen Minimierung endlicher verteilter Systeme vermieden werden können. Dabei steht in Abschnitt 4.1 die Betrachtung von Spezialfällen im Vordergrund, genauer gesagt die Betrachtung der $\mathcal{RM}$–Methode für den speziellen Reduktionsoperator $\overline{\Pi}$ unter schärferen Voraussetzungen, z.B. unter der Voraussetzung *korrekter Interfacespezifikationen* oder *total definierter Prozesse*. Setzt man total definierte Prozesse voraus, so lassen sich für den speziellen Reduktionsoperator algebraische Eigenschaften zeigen (vgl. Abschnitt 4.2), welche zu einer Beziehung zwischen dem speziellen Reduktionsoperator $\overline{\Pi}$ und einer Halbordnung auf total definierten Prozessen führen. Unter der Voraussetzung allgemeiner, also insbesondere auch partiell definierter, Prozesse gilt eine analoge Beziehung bzgl. der Quasiordnung jedoch nicht. In Abschnitt 4.3 wird gezeigt, wie korrekte Interfacespezifikationen automatisch konstruiert werden können und wie man dies in einer Methode zur kompositionellen Minimierung endlicher verteilter Systeme effektiv umsetzen kann. Die dazu notwendigen Abstraktionen sind i.allg. jedoch so groß, daß unter Verwendung dieser konstruierten korrekten Interfacespezifikationen die $\mathcal{RM}$–Methode gegenüber der trivialen Methode (inkl. Ausnutzung der Parallel–/Fensteroperatorkorrespondenz) in der Praxis keine Vorteile bietet. Ausnahmen bilden solche Systeme, bei denen nur die Reihenfolge, in der die einzelnen Parallelkomponenten bei der Methode betrachtet werden, für die Zustandsexplosion verantwortlich ist. In Abschnitt 4.4 werden die in diesem Kapitel vorgeschlagenen Verbesserungsansätze zusammengefaßt.

In dieser Arbeit gilt das Interesse der Minimierung endlicher verteilter Systeme (dargestellt durch Transitionssysteme) bzgl. der Beobachtungsäquivalenz $\approx$ von Milner. Da die Korrektheit der $\mathcal{RM}$–Methode unabhängig von der Korrektheit der Interfacespezifikationen sein sollte, mußte als zusätzliches Hilfsmittel das Undefiniertheitsprädikat für Transitionssysteme bzw. Prozesse und eine Quasiordnung auf Prozessen eingeführt werden. Ersteres bedeutete jedoch auch, daß die semantische Äquivalenz $\approx$ verfeinert werden mußte und daher nur solche Zustände eines Transitionssystems semantisch äquivalent sein können, die dieselben Undefiniertheiten besitzen (vgl. Definition 1.12). Eine Äquivalenz-

klasse bzgl. der Beobachtungsäquivalenz $\approx$ von Milner zerfällt bzgl. der Äquivalenz $\approx^d$ i.allg. in mehrere Äquivalenzklassen. Dies führt bei der Anwendung der $\mathcal{RM}$–Methode zu einer (in bezug auf $\approx$) höheren algorithmischen Komplexität. Um zu einem gegebenen Prozeß p und einer Interfacespezifikation $I \in \mathcal{I}(p)$ den reduzierten Prozeß (bzgl. des speziellen Reduktionsoperators) $\overline{\Pi}_I(p)$ zu konstruieren, berechnet man gemäß Kapitel 2, Abschnitt 2.3, zu jedem Zustand q von p die Menge IL_q. Dort wurde ebenfalls festgestellt (vgl. auch Satz 2.9), daß die mit Aktionen aus $\mathcal{A}_p \setminus IL_q$ beschrifteten Undefiniertheiten im Kontext wieder verschwinden, da diese Undefiniertheiten nach Definition und Konstruktion der IL_q–Mengen und nach Definition der operationellen Semantik im Kontext verhindert werden. Folglich lassen sich die mit diesen Aktionen beschrifteten Undefiniertheiten als „*Don't–Cares*" bezeichnen, welche benutzt werden können, um den Zerfall von Äquivalenzklassen bzgl. $\approx^d$ zu mindern, denn: Sind zwei Zustände p', p'' $\approx$–äquivalent, aber besitzen unterschiedliche Undefiniertheiten (d.h. $p' \not\approx^d p''$), so lassen sich die Undefiniertheiten mit Hilfe der Don't–Cares u.U. angleichen, so daß dann $p' \approx^d p''$ gelten würde. Der mögliche Gewinn dieses Vorgehens ist eine geringere algorithmische Komplexität bei der Anwendung der $\mathcal{RM}$–Methode. Die Schwierigkeit dieses Ansatzes liegt in der Heuristik im Umgang mit Don't–Care–Undefiniertheiten: Von welchen Zuständen soll man die Undefiniertheiten wie angleichen? Am günstigsten wird es sein, Zuständen nur soviele Don't–Care–Undefiniertheiten hinzuzufügen, wie unbedingt erforderlich sind, um eine $\approx^d$–Äquivalenz zwischen den betrachteten $\approx$–äquivalenten Zuständen zu erzielen. Da die $\mathcal{RM}$–Methode jedoch aus mehreren Schritten mit sukzessiver Hinzunahme von Parallelkomponenten besteht, kann folgende Situation eintreten: Obwohl in einem Schritt zuvor im Zwischentransitionssystem Don't–Care–Undefiniertheiten so hinzugefügt wurden, daß nach seiner Minimierung bzgl. $\approx^d$ seine Zustandsanzahl minimal ist, können in einem nachfolgenden Schritt die hinzugefügten Don't–Care–Undefiniertheiten eine Zusammenfassung von ursprünglich $\approx^d$–äquivalenten Zuständen verhindern. Heuristiken für den Umgang mit Don't–Care–Undefiniertheiten lassen sich nur schwer aufstellen, weil der *Wirkungsgrad* dieser Technik zur Senkung der algorithmischen Komplexität vom betrachteten System abhängt. Hat man spezielle Kenntnisse über das betrachtete System, so läßt sich die Technik der Don't–Care–Undefiniertheiten u.U. wirkungsvoll einsetzen.

4.1 Betrachtung von $\overline{\Pi}$ für Spezialfälle

Im folgenden wird der in Kapitel 2 vorgestellte spezielle Reduktionsoperator $\overline{\Pi}$ für Spezialfälle näher betrachtet.

Der spezielle Reduktionsoperator $\overline{\Pi}$ ist nach Definition 2.3 (5b) so definiert, daß er zur Kennzeichnung der Zustände, an denen bei der Reduktion im Kontext unerreichbare Transitionen entfernt werden, die entsprechenden Undefiniertheiten hinzufügt. Geht man bei der Minimierung endlicher verteilter Systeme von total definierten Prozessen aus, so erlaubt diese Technik zusammen mit der in Kapitel 3 entwickelten $\mathcal{RM}$–Methode zu erkennen, ob der konstruierte minimale Prozeß im Sinne von Milner semantisch äquivalent

zum ursprünglichen System ist (vgl. Korollar 3.3 & Proposition 1.19). Betrachtet man die $\mathcal{RM}$–Methode unter der Voraussetzung, daß alle benötigten Interfacespezifikationen korrekt sind, so kann man sich offensichtlich (vgl. Satz 2.9) die obige Technik sparen, da die gemäß Definition 2.3 (5b) hinzugefügten Undefiniertheiten im globalen Kontext wieder verschwinden. Warum sollten sie also zuvor hinzugefügt werden? In der Situation korrekter Interfacespezifikationen genügt es, die in einem beliebigen Prozeß vorhandenen Undefiniertheiten bei der Anwendung von $\overline{\Pi}$ bzgl. einer beliebigen korrekten Interfacespezifikation zu vererben. Falls das betrachtete System zusätzlich total definiert ist, kann man Undefiniertheiten gänzlich außer acht lassen. Für eine Methode zur kompositionellen Minimierung endlicher verteilter Systeme, welche total definierte Prozesse und korrekte Interfacespezifikationen voraussetzt, genügt somit der folgende Reduktionsoperator , , wobei $\underline{\text{TotDefProc}}$ die Menge aller total definierten Prozesse bezeichnet.

Definition 4.1 (Der Reduktionsoperator ,)
Sei $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \emptyset), p) \in \underline{TotDefProc}$ *und* $I \in \mathcal{I}(p)$.
Dann ist , : $\underline{Interfaces} \times \underline{TotDefProc} \longrightarrow \underline{TotDefProc}$ *definiert durch*

$$(I, p) \longmapsto \ , (I, p) =_{df} \ , {}_I(p) =_{df} ((S, \longrightarrow, \mathcal{A} \cup \{\tau\}, \emptyset), p),$$

wobei

1. $S = \{q \in S_p \mid \exists\, i \in S_I.\ q \| i \in S_{p\|I}\}$,
2. $\mathcal{A} = \mathcal{A}_p$ *und*
3. $\forall\, q, q' \in S\ \forall\, a \in \mathcal{A} \cup \{\tau\}.\ \ q \xrightarrow{a} q'$ *genau dann, wenn* $\exists\, i, i' \in S_I.\ q \| i \xrightarrow{a}_{p\|I} q' \| i'$.[1]

Die Definition entspricht der des speziellen Reduktionsoperators $\overline{\Pi}$ (vgl. Definition 2.3), wobei die dort vorkommenden Regeln (4) und (5) weggelassen wurden (vgl. obige Diskussion). Es folgt unmittelbar (man vergleiche die Sätze und Beweise aus Kapitel 2, Abschnitt 2.2, bzgl. der Definition von , und unter der Voraussetzung total definierter Prozesse), daß es sich bei , tatsächlich um einen Reduktionsoperator handelt. Dabei ist die Forderung (i) aus Definition 2.1 nun unzweckmäßig (und auch nicht der Intuition entsprechend), da im Fall total definierter Prozesse , $\approx^d$ und $\approx$ nach Proposition 1.19 zusammenfallen und deswegen diese Forderung für einen Reduktionsoperator keinen Sinn macht. Formal wird sie durch eine analoge Bedingung ersetzt, bei der die Quasiordnung durch die Halbordnung $\leq_t$ (siehe Definition 4.3) ausgetauscht wird. Der Reduktionsoperator , erfüllt wie auch $\overline{\Pi}$ Forderung (iii) aus Definition 2.1. Diese Forderung ist hier trivial, da sie bereits durch die modifizierte Bedingung (i) derselben Definition impliziert wird.

[1] Beachte: Diese Forderung impliziert bereits, daß $q \| i$ in $p \| I$ erreichbar ist.

Lemma 4.2
Die Abbildung , : Interfaces × TotDefProc − → TotDefProc aus Definition 4.1 definiert einen Reduktionsoperator.

Wie bereits erwähnt, soll in der hier betrachteten Situation eine Halbordnung statt einer Quasiordnung auf Prozessen im Mittelpunkt stehen. Da der Begriff *Halbordnung* im Vergleich zum Begriff *Quasiordnung* stärker ist (siehe auch Anhang A), läßt sich nun die Gültigkeit von Aussagen erhoffen, die analog für lediglich partiell definierte Prozesse bzgl. der Quasiordnung nicht richtig sind. Die Halbordnung $\leq$ aus Definition 2.6 läßt sich wie folgt an die neue Situation anpassen:

Definition 4.3 (Halbordnung auf total definierten Prozessen)
Seien $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \emptyset), p)$ *und* $q = ((S_q, \mathcal{A}_q \cup \{\tau\}, \longrightarrow_q, \emptyset), q) \in$ *TotDefProc. Es gelte* $p \leq_t q$ *genau dann, wenn folgende Bedingungen erfüllt sind:*

1. $\mathcal{A}_p = \mathcal{A}_q$,
2. $S_p \subseteq S_q$ *(die Startzustände* p *und* q *seien identisch) und*
3. $\forall\, p', p'' \in S_p \;\; \forall\, a \in \mathcal{A}_p \cup \{\tau\}.\;\; p' \xrightarrow{a}_p p''$ *impliziert* $p' \xrightarrow{a}_q p''$.

Der Beweis, daß $\leq_t$ eine Halbordnung auf Prozessen ist, wird im folgenden Abschnitt 4.2 geführt. Vergleicht man Definition 4.3 mit Definition 4.1, so stellt man wie in Kapitel 2 fest, daß erstere von letzterer „abstammt“. Die Eigenschaften (1) und (2) aus Definition 4.3 werden sofort durch die Punkte (1) und (2) aus Definition 4.1 impliziert. Für den Punkt (3) muß man noch die Aussage von Lemma 2.11 zu Hilfe nehmen, welche analog für den Reduktionsoperator , gilt (vgl. dazu den Beweis von Lemma 2.11). Diese triviale Beobachtung ist die Aussage und der Beweis des nächsten Lemmas, welches der neuen Bedingung (i) aus Definition 2.1 entspricht. Zusammen mit dem Beweis von Satz 2.9 (analog für den Reduktionsoperator ,) liefert Lemma 4.4 den Beweis von Lemma 4.2.

Lemma 4.4
Seien $p \in$ *TotDefProc und* $I \in \mathcal{I}(p)$. *Dann gilt* $,_I(p) \leq_t p$.

Es sei noch erwähnt, daß man einen reduzierten Prozeß bzgl. , für total definierte Prozesse ebenfalls wie in Kapitel 2, Abschnitt 2.3, vorgeschlagen berechnen kann. Insbesondere sind die Mengen IL_q hilfreich. Der einzige Unterschied bei der Berechnung von , im vgl. zu $\overline{\Pi}$ besteht in der Konstruktionsvorschrift aus Proposition 2.13. Dort läßt sich der zweite Punkt nun wie folgt vereinfachen, da wegen der Voraussetzung korrekter Interfacespezifikationen und total definierter Prozesse keine Undefiniertheiten mehr betrachtet werden müssen (vgl. Definition 4.1): „Gilt $q \xrightarrow{a}_p$, aber $a \notin IL_q$, so eliminiere alle von q ausgehenden a–Transitionen“. Die Komplexität für die Berechnung eines reduzierten Prozesses bzgl. , ändert sich dadurch offensichtlich nicht. Der Hauptaufwand wird immer noch für die Berechnung der IL_q–Mengen benötigt.

4.2 Algebraische Eigenschaften von $\leq_t$ und ,

In diesem Abschnitt geht es um algebraische Eigenschaften der Halbordnung $\leq_t$ und des Reduktionsoperators , unter der Voraussetzung total definierter Prozesse. Diese Eigenschaften dienen zum besseren Verständnis des Reduktionsoperators , und sind zum Beweis einer algebraischen Beziehung zwischen $\leq_t$ und , von Nutzen. Es wird gezeigt, daß einige analoge Eigenschaften und Beziehungen für partiell definierte Prozesse bzgl. der Halbordnung $\leq$ oder der Quasiordnung und des Reduktionsoperators $\overline{\Pi}$ nicht gelten.

Halbordnung

Daß durch $\leq_t$ tatsächlich eine Halbordnung auf total definierten Prozessen definiert wird, besagt das folgende Lemma:

Lemma 4.5
Die Relation $\leq_t$ aus Definition 4.3 definiert eine Halbordnung auf total definierten Prozessen.

Beweis
Seien $p, q, r \in$ TotDefProc beliebig.

1. Transitivität: Gelte $p \leq_t q$ und $q \leq_t r$. Dann gilt auch $p \leq_t r$, denn:
   (a) $\mathcal{A}_p = \mathcal{A}_q = \mathcal{A}_r$ impliziert $\mathcal{A}_p = \mathcal{A}_r$.
   (b) $S_p \subseteq S_q \subseteq S_r$ impliziert $S_p \subseteq S_r$ und,
   da die Startzustände p und q sowie q und r identisch sind, sind auch die Startzustände p und r identisch.
   (c) Seien $p', p'' \in S_p$ und $a \in \mathcal{A}_p \cup \{\tau\}$ beliebig. Dann gilt:

$$\begin{aligned} & p' \xrightarrow{a}_p p'' \\ (p \leq_t q) \quad \Rightarrow \quad & p' \xrightarrow{a}_q p'' \\ (q \leq_t r) \quad \Rightarrow \quad & p' \xrightarrow{a}_r p''. \end{aligned}$$

2. Reflexivität: Es gilt $p \leq_t p$, denn:
   (a) $\mathcal{A}_p = \mathcal{A}_p$.
   (b) $S_p \subseteq S_p$, und die Startzustände p und p sind identisch.
   (c) Seien $p', p'' \in S_p$ und $a \in \mathcal{A}_p \cup \{\tau\}$ beliebig. Dann gilt offensichtlich $p' \xrightarrow{a}_p p''$ impliziert $p' \xrightarrow{a}_p p''$.

3. Antisymmetrie: Gelte $p \leq_t q$ und $q \leq_t p$. Dann gilt $p = q$, denn:
   (a) Wegen $p \leq_t q$ und $q \leq_t p$ gilt $\mathcal{A}_p = \mathcal{A}_q$.

(b) Mit $S_p \subseteq S_q$ $(p \leq_t q)$ und $S_q \subseteq S_p$ $(q \leq_t p)$ folgt $S_p = S_q$, und die Startzustände p und q sind identisch.

(c) Seien $p', p'' \in S_p$ und $a \in \mathcal{A}_p \cup \{\tau\}$ beliebig. Dann gilt (wegen $p' \xrightarrow{a}_p p''$ impliziert $p' \xrightarrow{a}_q p''$ $(p \leq_t q)$ und $p' \xrightarrow{a}_q p''$ impliziert $p' \xrightarrow{a}_p p''$ $(q \leq_t p)$) $p' \xrightarrow{a}_p p''$ genau dann, wenn $p' \xrightarrow{a}_q p''$. Also folgt $\longrightarrow_p = \longrightarrow_q$.

Kompositionalität

Weiterhin gelten für total definierte Prozesse die zu Satz 1.20 analogen Aussagen über die Kompositionalität der Operatoren $\|$ und $\langle L \rangle$ bzgl. $\leq_t$:

Lemma 4.6 (Kompositionalität)
Für alle Prozesse $p, q, r \in$ TotDefProc und alle Mengen L beobachtbarer Aktionen gilt:

(1) $p \leq_t q$ impliziert $p \| r \leq_t q \| r$ und

(2) $p \leq_t q$ impliziert $p\langle L \rangle \leq_t q\langle L \rangle$.

Beweis
Beweis von (1):

Seien $p, q, r \in$ TotDefProc beliebig mit $p \leq_t q$. Dann gilt $p \| r \leq_t q \| r$, denn:

1. Wegen $p \leq_t q$ gilt nach Definition 4.3 (1) $\mathcal{A}_p = \mathcal{A}_q$ und damit (vgl. Definition 1.8) $\mathcal{A}_{p\|r} = \mathcal{A}_p \cup \mathcal{A}_r = \mathcal{A}_q \cup \mathcal{A}_r = \mathcal{A}_{q\|r}$.

2. $S_{p\|r} \subseteq S_{q\|r}$ ist eine unmittelbare Konsequenz aus Punkt (3) dieses Beweises. Da p und q identische Startzustände sind $(p \leq_t q)$, sind auch die Startzustände $p\|r$ und $q\|r$ identisch.

3. Seien $p'\|r', p''\|r'' \in S_{p\|r}$ und $a \in \mathcal{A}_{p\|r} \cup \{\tau\}$ mit $p'\|r' \xrightarrow{a}_{p\|r} p''\|r''$ beliebig. Unterscheide nun gemäß Definition 1.8 die folgenden Fälle:

   (a) Regel (3):

$$
\begin{array}{lcl}
 & & p' \xrightarrow{a}_p p'' \ \wedge \ r' = r'' \ \wedge \ a \in \mathcal{A}_p \setminus \mathcal{A}_r \\
(p \leq_t q) & \Rightarrow & p' \xrightarrow{a}_q p'' \ \wedge \ r' = r'' \ \wedge \ a \in \mathcal{A}_q \setminus \mathcal{A}_r \\
(\text{Regel (3)}) & \Rightarrow & p'\|r' \xrightarrow{a}_{q\|r} p''\|r''
\end{array}
$$

(b) <u>Regel (4):</u>

$$
\begin{array}{lcl}
 & & r' \xrightarrow{a}_r r'' \;\wedge\; p' = p'' \;\wedge\; a \in \mathcal{A}_r \setminus \mathcal{A}_p \\
(p \leq_t q) & \Rightarrow & r' \xrightarrow{a}_r r'' \;\wedge\; p' = p'' \;\wedge\; a \in \mathcal{A}_r \setminus \mathcal{A}_q \\
(\text{Regel (4)}) & \Rightarrow & p' \| r' \xrightarrow{a}_{q \| r} p'' \| r''
\end{array}
$$

(c) <u>Regel (5):</u>

$$
\begin{array}{lcl}
 & & p' \xrightarrow{a}_p p'' \;\wedge\; r' \xrightarrow{a}_r r'' \;\wedge\; a \in \mathcal{A}_p \cap \mathcal{A}_r \\
(p \leq_t q) & \Rightarrow & p' \xrightarrow{a}_q p'' \;\wedge\; r' \xrightarrow{a}_r r'' \;\wedge\; a \in \mathcal{A}_q \cap \mathcal{A}_r \\
(\text{Regel (5)}) & \Rightarrow & p' \| r' \xrightarrow{a}_{q \| r} p'' \| r''
\end{array}
$$

Beweis von (2):

Seien $p, q \in$ <u>TotDefProc</u> beliebig mit $p \leq_t q$ und L eine beliebige Menge beobachtbarer Aktionen. Dann gilt $p\langle L \rangle \leq_t q\langle L \rangle$, denn:

1. Wegen $p \leq_t q$ gilt nach Definition 4.3 (1) $\mathcal{A}_p = \mathcal{A}_q$ und damit (vgl. Definition 1.8) $\mathcal{A}_{p\langle L \rangle} = \mathcal{A}_p \cap L = \mathcal{A}_q \cap L = \mathcal{A}_{q\langle L \rangle}$.

2. $S_{p \| r} \subseteq S_{q \| r}$ ist eine unmittelbare Konsequenz aus Punkt (3) dieses Beweises. Da p und q identische Startzustände sind ($p \leq_t q$), sind auch die Startzustände $p\langle L \rangle$ und $q\langle L \rangle$ identisch.

3. Seien $p'\langle L \rangle, p''\langle L \rangle \in S_{p\langle L \rangle}$ und $a \in \mathcal{A}_{p\langle L \rangle} \cup \{\tau\}$ mit $p'\langle L \rangle \xrightarrow{a}_{p\langle L \rangle} p''\langle L \rangle$ beliebig. Unterscheide nun gemäß Definition 1.8 die folgenden Fälle:

   (a) <u>$a = \tau$:</u>

$$
\begin{array}{lcl}
 & & p'\langle L \rangle \xrightarrow{\tau}_{p\langle L \rangle} p''\langle L \rangle \\
(\text{Definition 1.8 (2)}) & \Rightarrow & p' \xrightarrow{b}_p p'' \;\wedge\; b \notin L \\
(p \leq_t q) & \Rightarrow & p' \xrightarrow{b}_q p'' \;\wedge\; b \notin L \\
(\text{Definition 1.8 (2)}) & \Rightarrow & p'\langle L \rangle \xrightarrow{\tau}_{q\langle L \rangle} p''\langle L \rangle
\end{array}
$$

(b) $\underline{a \neq \tau}$:

$$
\begin{array}{lcl}
 & & p'\langle L\rangle \xrightarrow{a}_{p\langle L\rangle} p''\langle L\rangle \\
(\text{Definition 1.8 (1)}, a \neq \tau) & \Rightarrow & p' \xrightarrow{a}_{p} p'' \ \wedge \ a \in L \\
(p \leq_t q) & \Rightarrow & p' \xrightarrow{a}_{q} p'' \wedge \ a \in L \\
(\text{Definition 1.8 (1)}, a \in L) & \Rightarrow & p'\langle L\rangle \xrightarrow{a}_{q\langle L\rangle} p''\langle L\rangle
\end{array}
$$

Die erste Aussage von Lemma 4.6 läßt sich auf total definierte Prozesse bzgl. des Reduktionsoperators $\overline{\Pi}$ und der Halbordnung $\leq$ verallgemeinern. Dies gilt jedoch nicht für die zweite Aussage, wie das folgende Beispiel zeigt:

Beispiel 4.7
Betrachte die Prozesse $p, q \in$ Processes aus Abbildung 4.1 mit $\mathcal{A}_p = \mathcal{A}_q = \{a, b\}$.

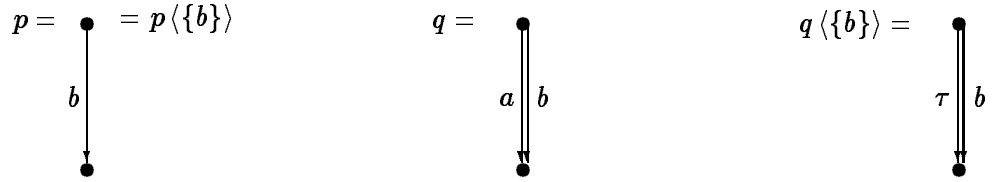


Abbildung 4.1: Gegenbeispiel zur Kompositionalität

Offensichtlich gilt $p \leq q$, aber wegen Punkt (3) aus Definition 2.6 nicht $p\langle\{b\}\rangle \leq q\langle\{b\}\rangle$.

Idempotenz

Der Reduktionsoperator , ist im folgenden Sinne *idempotent*:

Lemma 4.8 (Idempotenz)
$\forall\, p \in$ TotDefProc $\forall\, I \in \mathcal{I}(p).\ \ ,_I(,_I(p)) = ,_I(p)$.

Beweis
Seien $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \emptyset), p) \in$ TotDefProc und $I \in \mathcal{I}(p)$ beliebig. Ferner gelte $,_I(p) = ((S_1, \mathcal{A}_1 \cup \{\tau\}, \longrightarrow_1, \emptyset), p)$ und $,_I(,_I(p)) = ((S_2, \mathcal{A}_2 \cup \{\tau\}, \longrightarrow_2, \emptyset), p)$.
Dann gilt $,_I(,_I(p)) = ,_I(p)$, denn:

1. Nach Definition 4.1 (2) gilt $\mathcal{A}_p = \mathcal{A}_1 = \mathcal{A}_2$.

2. Nach Definition 4.1 ist p Startzustand von $,_I(p)$ und $,_I(,_I(p))$. Weiter gilt:

$$
\begin{array}{lcl}
 & & S_2 \\
(\text{Definition 4.1 (1)}) & = & \{q \in S_1 \mid \exists\, i \in S_I.\ q \| i \in S_{\Gamma_I(p)\|I}\} \\
(\text{Definition 4.1 (1)}) & = & \{q \in S_1 \mid \exists\, i \in S_I.\ q \| i \in S_{\Gamma_I(p)\|I} \ \wedge\ \exists\, i \in S_I.\ q \| i \in S_{p\|I}\} \\
(\text{Definition 4.1 (1)}) & = & \{q \in S_1 \mid \exists\, i \in S_I.\ q \| i \in S_{p\|I}\} \\
(\text{Definition 4.1 (1)}) & = & S_1
\end{array}
$$

3. Seien $p', p'' \in S_1 = S_2$ und $a \in \mathcal{A}_1 \cup \{\tau\}$ beliebig. Dann gilt:

$$
\begin{array}{lcl}
 & & p' \xrightarrow{a}_1 p'' \\
(\text{Definition 4.1 (3)}) & \Longleftrightarrow & \exists\, i', i'' \in S_I.\ p' \| i' \xrightarrow{a}_{p\|I} p'' \| i'' \\
(\text{Definition 1.8}) & \Longleftrightarrow & \exists\, i', i'' \in S_I.\ (p' \| i') \| i' \xrightarrow{a}_{(p\|I)\|I} (p'' \| i'') \| i'' \\
(\text{Definition 4.1 (3)}) & \Longleftrightarrow & \exists\, i', i'' \in S_I.\ p' \| i' \xrightarrow{a}_{\Gamma_I(p)\|I} p'' \| i'' \\
(\text{Definition 4.1 (3)}) & \Longleftrightarrow & p' \xrightarrow{a}_2 p''
\end{array}
$$

Damit folgt $\longrightarrow_1 = \longrightarrow_2$.

Eine analoge Aussage gilt auch für beliebige (insbesondere partiell definierte) Prozesse bzgl. des Reduktionsoperators $\overline{\Pi}$ und der Halbordnung $\leq$. Obwohl die Idempotenz also auch allgemeiner gilt, wird auf den Beweis dazu verzichtet, weil diese Eigenschaft für eine allgemeinere Gültigkeit von Proposition 4.11 alleine nicht ausreicht.

Monotonie

Der Reduktionsoperator $,$ ist eine in der zweiten Komponente monotone Abbildung bzgl. der Halbordnung $\leq_t$.

Lemma 4.9 (Monotonie)
Für beliebige total definierte Prozesse $p, q \in \underline{TotDefProc}$ und eine beliebige Interfacespezifikation $I \in \mathcal{I}(p)$ gilt:

$$p \leq_t q \ \ \textit{impliziert} \ \ ,_I(p) \leq_t ,_I(q).$$

Beweis

Seien $p, q \in \underline{\text{TotDefProc}}$ mit $p \leq_t q$ und $I \in \mathcal{I}(p)$ beliebig. Ferner gelte $,_I(p) = ((S_1, \mathcal{A}_1 \cup \{\tau\}, \longrightarrow_1, \emptyset), p)$ und $,_I(q) = ((S_2, \mathcal{A}_2 \cup \{\tau\}, \longrightarrow_2, \emptyset), q)$.
Dann gilt nach Definition 4.3 $,_I(p) \leq_t ,_I(q)$, denn:

1. Wegen $p \leq_t q$ und Definition 4.1 (2) gilt: $\mathcal{A}_1 = \mathcal{A}_p = \mathcal{A}_q = \mathcal{A}_2$.
   Insbesondere folgt mit Definition 1.23: $I \in \mathcal{I}(q)$.

2. Es gilt:

$$
\begin{array}{rcl}
 & & S_1 \\
(\text{Definition 4.1 (1)}) & = & \{p' \in S_p \mid \exists\, i \in S_I.\ p' \| i \in S_{p\|I}\} \\
(S_p \subseteq S_q, \text{ da } p \leq_t q) & \subseteq & \{p' \in S_q \mid \exists\, i \in S_I.\ p' \| i \in S_{q\|I}\} \\
(\text{Definition 4.1 (1)}) & = & S_2
\end{array}
$$

3. Seien $p', p'' \in S_1$ und $a \in \mathcal{A}_1 \cup \{\tau\}$ beliebig. Dann gilt:

$$
\begin{array}{rcl}
 & & p' \xrightarrow{a}_1 p'' \\
(\text{Definition 4.1 (3)}) & \Rightarrow & \exists\, i', i'' \in S_I.\ p' \| i' \xrightarrow{a}_{p\|I} p'' \| i''
\end{array}
$$

<u>1. Fall:</u> $a \notin \mathcal{A}_I$

$$
\begin{array}{rcl}
(\text{Definition 1.8 (3)}) & \Rightarrow & \exists\, i' \equiv i'' \in S_I.\ p' \xrightarrow{a}_p p'' \ \wedge\ p' \| i' \text{ ist in } p\|I \text{ erreichbar} \\
(p \leq_t q) & \Rightarrow & \exists\, i' \equiv i'' \in S_I.\ p' \xrightarrow{a}_q p'' \ \wedge\ p' \| i' \text{ ist in } q\|I \text{ erreichbar} \\
(\text{Definition 1.8 (3)}) & \Rightarrow & \exists\, i', i'' \in S_I.\ p' \| i' \xrightarrow{a}_{q\|I} p'' \| i'' \\
(\text{Definition 4.1 (3)}) & \Rightarrow & p' \xrightarrow{a}_2 p''
\end{array}
$$

<u>2. Fall:</u> $a \in \mathcal{A}_I$

$$
\begin{array}{lll}
(\text{Definition 1.8 (5)}) & \Rightarrow & \exists\, i', i'' \in S_I.\ p' \xrightarrow{a}_p p'' \ \wedge\ i' \xrightarrow{a}_I i'' \ \wedge \\
 & & \qquad p' \| i' \text{ ist in } p \| I \text{ erreichbar} \\
(p \leq_t q) & \Rightarrow & \exists\, i', i'' \in S_I.\ p' \xrightarrow{a}_q p'' \ \wedge\ i' \xrightarrow{a}_I i'' \ \wedge \\
 & & \qquad p' \| i' \text{ ist in } q \| I \text{ erreichbar} \\
(\text{Definition 1.8 (5)}) & \Rightarrow & \exists\, i', i'' \in S_I.\ p' \| i' \xrightarrow{a}_{q \| I} p'' \| i'' \\
(\text{Definition 4.1 (3)}) & \Rightarrow & p' \xrightarrow{a}_2 p''
\end{array}
$$

In bezug auf die Quasiordnung ist der spezielle Reduktionsoperator $\overline{\Pi}$ in seiner zweiten Komponente ebenfalls monoton. Diese allgemeinere Aussage liefert jedoch keine allgemeinere Gültigkeit von Proposition 4.11 im folgenden Unterabschnitt, da nach Kapitel 1 keine Halbordnung ist und die Monotonie bzgl. der Halbordnung $\leq$ *nicht* gilt.

Beispiel 4.10
Die Monotonieeigenschaft gilt i.allg. nicht für den Reduktionsoperator $\overline{\Pi}$ und partiell definierte Prozesse bzgl. der Halbordnung $\leq$.

Betrachte als Gegenbeispiel die Prozesse $p, q \in$ <u>Processes</u> und die Interfacespezifikation $I \in \mathcal{I}(p) \cap \mathcal{I}(q)$ mit $\mathcal{A}_p = \mathcal{A}_q = \mathcal{A}_I = \{a\}$ aus Abbildung 4.2.

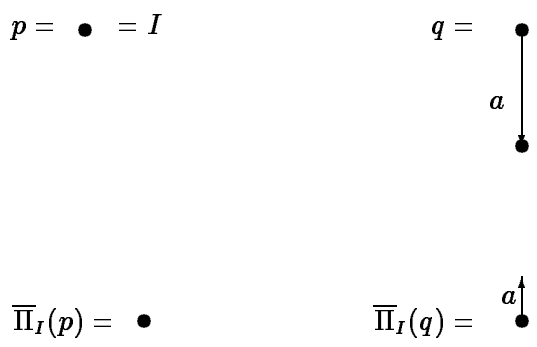


Abbildung 4.2: Gegenbeispiel zur Monotonie

Es gilt gemäß Definition 2.6 (6) nicht $\overline{\Pi}_I(p) \leq \overline{\Pi}_I(q)$, jedoch $p \leq q$.

Eine Beziehung zwischen Γ und $\leq_t$

Zur Vervollständigung des Überblicks über die Zusammenhänge des speziellen Reduktionsoperators , und der Halbordnung $\leq_t$ trägt die folgende Proposition bei:

Proposition 4.11
Seien $p, q \in \underline{TotDefProc}$, $I \in \mathcal{I}(p) \cap \mathcal{I}(q)$ beliebig und gelte $, _I(p) \leq_t q \leq_t p$. Dann gilt

$$, _I(p) = , _I(q).$$

Beweis
Zum Beweis seien $p, q \in \underline{\text{TotDefProc}}$, $I \in \mathcal{I}(p) \cap \mathcal{I}(q)$ beliebig, und es gelte
$, _I(p) \leq_t q \leq_t p$.

<u>„$\geq_t$“:</u>

$q \leq_t p$ impliziert (Lemma 4.9) $, _I(q) \leq_t , _I(p)$.

<u>„$\leq_t$“:</u>

$$
\begin{array}{lcl}
 & & , _I(p) \leq_t q \\
(\text{Lemma 4.9}) & \Rightarrow & , _I(, _I(p)) \leq_t , _I(q) \\
(\text{Lemma 4.8}) & \Rightarrow & , _I(p) \leq_t , _I(q)
\end{array}
$$

Aus „$\leq_t$“ und „$\geq_t$“ folgt mit Lemma 4.5 (Antisymmetrie) die Behauptung.

Die Aussage des obigen Lemmas gilt *nicht* für partiell definierte Prozesse und den Reduktionsoperator $\overline{\Pi}$, falls man $\leq_t$ durch und $=$ durch $\approx^d$ ersetzt.

Beispiel 4.12
Als Gegenbeispiel dienen die Prozesse $p, q \in Proc$ und die Interfacespezifikation $I \in \mathcal{I}(p) \cap \mathcal{I}(q)$ mit $\mathcal{A}_p = \mathcal{A}_q = \mathcal{A}_I = \{a, b\}$ aus Abbildung 4.3.
Nach Definition 1.15 gilt $\overline{\Pi}_I(p)$ q p, aber nicht $\overline{\Pi}_I(p) \approx^d \overline{\Pi}_I(q)$. Die Ursache liegt darin, daß keine Halbordnung ist, denn es gilt sowohl $\overline{\Pi}_I(p)$ $\overline{\Pi}_I(q)$ als auch $\overline{\Pi}_I(q)$ $\overline{\Pi}_I(p)$.

4.3 Automatische Konstruktion korrekter Interfacespezifikationen in einer Methode

Ein wesentlicher Nachteil der in Kapitel 3 vorgestellten $\mathcal{RM}$–Methode betrifft die benötigten Interfacespezifikationen, da diese vom Programmierer oder Programmdesigner vorgegeben werden müssen. Die Methode läßt zwar zu, daß die Interfacespezifikationen nicht notwendigerweise korrekt sind, d.h. sogar erraten werden können, jedoch bestimmt die Exaktheit der Interfacespezifikationen die Kontexttreue bzw. die Güte der Reduktion. Falls durch inkorrekte Interfacespezifikationen ein zum Ausgangssystem $\mathcal{P}$ semantisch nicht äquivalentes System $\mathcal{P}_n$ Resultat der Methode ist, d.h. $\mathcal{P}_n \not\approx^d \mathcal{P}$ gilt, und das Beweisziel

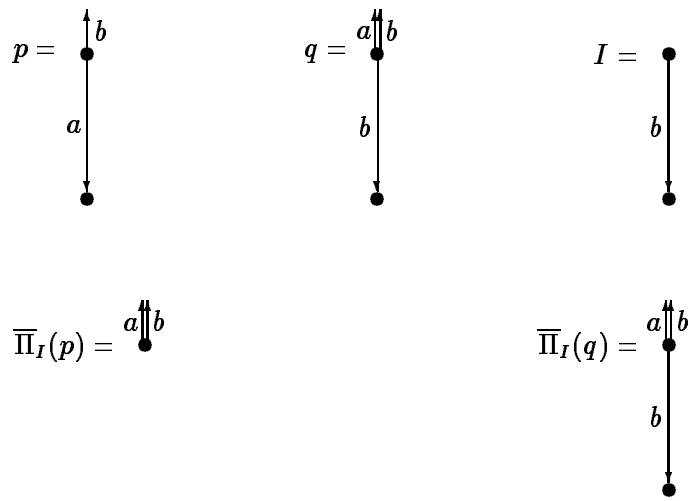


Abbildung 4.3: Gegenbeispiel

Φ nicht erreicht wird, so ist offen, ob $\mathcal{P} \models \Phi$ oder $\mathcal{P} \not\models \Phi$ gilt. Ein „Verbessern" der Interfacespezifikationen ist in diesem Fall erforderlich. Dies motiviert den Wunsch, korrekte Interfacespezifikationen automatisch konstruieren zu können, so daß $\mathcal{P}_n \approx^d \mathcal{P}$ garantiert ist. Grundlegend ist die Beobachtung, daß exakte Interfacespezifikationen zwischen zwei Prozessen effizient berechenbar sind, nämlich durch *einmalige* Anwendung des Paralleloperators und anschließende Anwendung des Fensteroperators. Wie aber sind für die $\mathcal{RM}$–Methode aus Kapitel 3 oder eine ähnliche Methode, welche (korrekte) Interfacespezifikationen zwischen zwei Teilen des gesamten Systems benötigt, solche aus denjenigen zwischen je zwei Parallelkomponenten zu konstruieren? Dazu wird im folgenden ein Vorschlag gemacht, der total definierte Prozesse und die Halbordnung aus Definition 4.3 voraussetzt.

Bestimmung korrekter Interfacespezifikationen

Für eine Methode, die korrekte Interfacespezifikationen zwischen den einzelnen Prozessen verlangt und den speziellen Reduktionsoperator , für total definierte Prozesse benutzt, muß man eine Beziehung zwischen diesen „lokal" korrekten Interfacespezifikationen und den „global" korrekten Interfacespezifikationen finden, wie sie für die $\mathcal{RM}$–Methode benötigt werden. Erst dann kann eine globale Reduktion durchgeführt und die Korrektheit einer Methode auf bereits in Kapitel 2 bewiesene Aussagen zurückgeführt werden.

Satz 4.13
Für alle total definierten Prozesse $p, p_1, p_2 \in$ <u>TotDefProc</u> und alle korrekten Interfacespezifikationen $I_1 \in \mathcal{I}(p, p_1)$ und $I_2 \in \mathcal{I}(p, p_2)$ gilt: $I_1 \| I_2 \in \mathcal{I}(p, p_1 \| p_2)$.

Beweis
Seien $p, p_1, p_2 \in$ <u>TotDefProc</u> beliebige, total definierte, o.B.d.A. (vgl. Satz 2.17) deterministische Prozesse, sowie $I_1 \in \mathcal{I}(p, p_1)$ und $I_2 \in \mathcal{I}(p, p_2)$ beliebige korrekte Interfacespezi-

fikationen. Nach Definition 1.23 gilt:

$$\mathcal{L}((p\|p_1)\langle\mathcal{A}_p\cap\mathcal{A}_{p_1}\rangle)\subseteq\mathcal{L}(I_1)\ \text{ und }\ \mathcal{L}((p\|p_2)\langle\mathcal{A}_p\cap\mathcal{A}_{p_2}\rangle)\subseteq\mathcal{L}(I_2).$$

Zum Beweis von $I_1\|I_2\in\mathcal{I}(p,p_1\|p_2)$ ist nach Definition 1.23

$$\mathcal{L}((p\|p_1\|p_2)\langle\mathcal{A}_p\cap(\mathcal{A}_{p_1}\cup\mathcal{A}_{p_2})\rangle)\subseteq\mathcal{L}(I_1\|I_2)$$

zu zeigen, was gleichbedeutend ist mit

$$w\in\mathcal{L}((p\|p_1\|p_2)\langle\mathcal{A}_p\cap(\mathcal{A}_{p_1}\cup\mathcal{A}_{p_2})\rangle)\ \text{ impliziert }\ w\in\mathcal{L}(I_1\|I_2).$$

Da p, p_1, p_2, I_1 und I_2 total definiert sind, ist diese Aussage nach Definition 1.7 äquivalent zu:

$$\exists\, p',p_1',p_2'.\ \ (p\|p_1\|p_2)\langle\mathcal{A}_p\cap(\mathcal{A}_{p_1}\cup\mathcal{A}_{p_2})\rangle\stackrel{w}{\Longrightarrow}(p'\|p_1'\|p_2')\langle\mathcal{A}_p\cap(\mathcal{A}_{p_1}\cup\mathcal{A}_{p_2})\rangle$$

impliziert

$$\exists\, I_1',I_2'.\ \ I_1\|I_2\stackrel{w}{\Longrightarrow}I_1'\|I_2'.$$

Zeige dazu die Existenz eines $p''\in S_p$ mit:

1. $\exists\, p_1''.\ (p\|p_1)\langle\mathcal{A}_p\cap\mathcal{A}_{p_1}\rangle\stackrel{w_1}{\Longrightarrow}(p''\|p_1'')\langle\mathcal{A}_p\cap\mathcal{A}_{p_1}\rangle$
   (Dabei entstehe w_1 aus w durch Streichen aller Aktionen, die nicht in $\mathcal{A}_p\cap\mathcal{A}_{p_1}$ liegen.)
   Diese Aussage impliziert, da nach Definition 1.23 I_1 total definiert ist und $I_1\in\mathcal{I}(p,p_1)$ gilt: $\exists\, I_1''.\ I_1\stackrel{w_1}{\Longrightarrow}I_1''$.

2. $\exists\, p_2''.\ (p\|p_2)\langle\mathcal{A}_p\cap\mathcal{A}_{p_2}\rangle\stackrel{w_2}{\Longrightarrow}(p''\|p_2'')\langle\mathcal{A}_p\cap\mathcal{A}_{p_2}\rangle$
   (Dabei entstehe w_2 aus w durch Streichen aller Aktionen, die nicht in $\mathcal{A}_p\cap\mathcal{A}_{p_2}$ liegen.)
   Diese Aussage impliziert, da nach Definition 1.23 I_2 total definiert ist und $I_2\in\mathcal{I}(p,p_2)$ gilt: $\exists\, I_2''.\ I_2\stackrel{w_2}{\Longrightarrow}I_2''$.

3. $\exists\, I_1''\|I_2''.\ I_1\|I_2\stackrel{w}{\Longrightarrow}I_1''\|I_2''$
   Dies impliziert nach Definition 1.7 $w\in\mathcal{L}(I_1\|I_2)$, was zu zeigen ist.

Beweis von (1)–(3) mittels simultaner vollständiger Induktion über die Länge i von w:

Induktionsanfang ($i=0$):

Hier gilt $w=w_1=w_2=\epsilon$. Die Punkte (1)–(3) folgen sofort mit $p''=_{df}p$, $p_1''=_{df}p_1$, $p_2''=_{df}p_2$, $I_1''=_{df}I_1$ und $I_2''=_{df}I_2$.

Induktionsschluß ($i \longrightarrow i+1$):

w habe die Form $w = w'a$ mit Länge $w' = i$. Da w' Präfix der Länge i ist, gilt $w' \in \mathcal{L}((p\|p_1\|p_2)\langle \mathcal{A}_p \cap (\mathcal{A}_{p_1} \cup \mathcal{A}_{p_2})\rangle)$, und damit existiert nach Induktionsvoraussetzung ein $p' \in S_p$ mit:

(i) $\exists\, p_1'.\ (p\|p_1)\langle \mathcal{A}_p \cap \mathcal{A}_{p_1}\rangle \stackrel{w_1'}{\Longrightarrow} (p'\|p_1')\langle \mathcal{A}_p \cap \mathcal{A}_{p_1}\rangle$, also $\exists\, I_1'.\ I_1 \stackrel{w_1'}{\Longrightarrow} I_1'$.
(Dabei entsteht w_1' aus w' durch Streichen aller Aktionen, die nicht in $\mathcal{A}_p \cap \mathcal{A}_{p_1}$ sind.)

(ii) $\exists\, p_2'.\ (p\|p_2)\langle \mathcal{A}_p \cap \mathcal{A}_{p_2}\rangle \stackrel{w_2'}{\Longrightarrow} (p'\|p_2')\langle \mathcal{A}_p \cap \mathcal{A}_{p_2}\rangle$, also $\exists\, I_2'.\ I_2 \stackrel{w_2'}{\Longrightarrow} I_2'$.
(Dabei entsteht w_2' aus w' durch Streichen aller Aktionen, die nicht in $\mathcal{A}_p \cap \mathcal{A}_{p_2}$ sind.)

(iii) $\exists\, I_1'\|I_2'.\ I_1\|I_2 \stackrel{w'}{\Longrightarrow} I_1'\|I_2'$, also $w' \in \mathcal{L}(I_1\|I_2)$

Die Situation ist nun:

$$\begin{aligned}(p\|p_1\|p_2)\langle \mathcal{A}_p \cap (\mathcal{A}_{p_1} \cup \mathcal{A}_{p_2})\rangle &\stackrel{w'}{\Longrightarrow} (p'\|p_1'\|p_2')\langle \mathcal{A}_p \cap (\mathcal{A}_{p_1} \cup \mathcal{A}_{p_2})\rangle \\ &\stackrel{a}{\Longrightarrow} (p''\|p_1''\|p_2'')\langle \mathcal{A}_p \cap (\mathcal{A}_{p_1} \cup \mathcal{A}_{p_2})\rangle\,.\end{aligned}$$

Nach Definition 1.8 (1) gilt $a \in \mathcal{A}_p \cap (\mathcal{A}_{p_1} \cup \mathcal{A}_{p_2})$. Betrachte nun die folgende Fallunterscheidung:

<u>1. Fall:</u> $a \in \mathcal{A}_p \cap \mathcal{A}_{p_1}$, also $w_1 = w_1'a$.

In diesem Fall gilt wegen (i) und Definition 1.8:

$$\begin{aligned}(p\|p_1)\langle \mathcal{A}_p \cap \mathcal{A}_{p_1}\rangle &\stackrel{w_1'}{\Longrightarrow} (p'\|p_1')\langle \mathcal{A}_p \cap \mathcal{A}_{p_1}\rangle \\ &\stackrel{a}{\Longrightarrow} (p''\|p_1'')\langle \mathcal{A}_p \cap \mathcal{A}_{p_1}\rangle\,.\end{aligned}$$

D.h. es gilt (a): $\exists\, I_1''.\ I_1 \stackrel{w_1'}{\Longrightarrow} I_1' \stackrel{a}{\Longrightarrow} I_1''$.

<u>2. Fall:</u> $a \in \mathcal{A}_p \cap \mathcal{A}_{p_2}$, also $w_2 = w_2'a$.

In diesem Fall gilt wegen (ii) und Definition 1.8:

$$\begin{aligned}(p\|p_2)\langle \mathcal{A}_p \cap \mathcal{A}_{p_2}\rangle &\stackrel{w_2'}{\Longrightarrow} (p'\|p_2')\langle \mathcal{A}_p \cap \mathcal{A}_{p_2}\rangle \\ &\stackrel{a}{\Longrightarrow} (p''\|p_2'')\langle \mathcal{A}_p \cap \mathcal{A}_{p_2}\rangle\,.\end{aligned}$$

D.h. es gilt (b): $\exists\, I_2''.\ I_2 \stackrel{w_2'}{\Longrightarrow} I_2' \stackrel{a}{\Longrightarrow} I_2''$.

Zusammenfassend:

Nach (iii) gilt $I_1 \| I_2 \stackrel{w'}{\Longrightarrow} I_1' \| I_2'$ und mit obiger Fallunterscheidung folgt wegen (a) und (b):
$I_1' \| I_2' \stackrel{a}{\Longrightarrow} I_1''' \| I_2'''$ mit

$$I_1''' \| I_2''' =_{df} \begin{cases} I_1'' \| I_2' & \text{falls } a \in (\mathcal{A}_p \cap \mathcal{A}_{p_1}) \setminus \mathcal{A}_{p_2} \\ I_1' \| I_2'' & \text{falls } a \in (\mathcal{A}_p \cap \mathcal{A}_{p_2}) \setminus \mathcal{A}_{p_1} \\ I_1'' \| I_2'' & \text{falls } a \in \mathcal{A}_p \cap \mathcal{A}_{p_1} \cap \mathcal{A}_{p_2}\,. \end{cases}$$

Wegen $\exists\, I_1''' \| I_2'''.\ I_1 \| I_2 \stackrel{w}{\Longrightarrow} I_1''' \| I_2'''$ gilt (c), d. h. $w \in \mathcal{L}(I_1 \| I_2)$. Es ist zu beachten, daß obige Schlußfolgerungsweise nur möglich ist, da die total definierten Prozesse p, p_1 und p_2 o.B.d.A. deterministisch gewählt werden konnten.

Per Induktionsprinzip folgt die Behauptung.

Auf die Voraussetzung des obigen Satzes, daß die Prozesse p, p_1 und p_2 total definiert sind, kann *nicht* verzichtet werden, denn:

Beispiel 4.14
Betrachte als Gegenbeispiel die Prozesse $p, p_1, p_2 \in$ Processes und die Interfacespezifikationen $I_1, I_2 \in \mathcal{I}(p) \cap \mathcal{I}(p_1) \cap \mathcal{I}(p_2)$ aus Abbildung 4.4, wobei das Alphabet dieser Prozesse jeweils $\{a\}$ sein soll.
Es gilt $I_1 \in \mathcal{I}(p, p_1)$, da $\mathcal{L}((p \| p_1)\langle \mathcal{A}_p \cap \mathcal{A}_{p_1} \rangle) = \{\epsilon, a\} = \mathcal{L}(I_1)$ (vgl. Definition 1.23). Wegen $\mathcal{L}((p \| p_2)\langle \mathcal{A}_p \cap \mathcal{A}_{p_2} \rangle) = a^* = \mathcal{L}(I_2)$, folgt ebenso $I_2 \in \mathcal{I}(p, p_2)$. Jedoch ist die Aussage $I_1 \| I_2 \in \mathcal{I}(p, p_1 \| p_2)$ falsch, denn (vgl. Abbildung 4.4):

$$\mathcal{L}((p \| p_1 \| p_2)\langle \mathcal{A}_p \cap (\mathcal{A}_{p_1} \cup \mathcal{A}_{p_2}) \rangle) = a^* \not\subseteq \{\epsilon, a\} = \mathcal{L}(I_1 \| I_2).$$

Satz 4.13 zeigt eine Möglichkeit auf, wie aus lokal korrekten Interfacespezifikationen eine global korrekte Interfacespezifikation konstruiert werden kann. Dabei wird durch den Gebrauch des Paralleloperators im folgenden Sinne abstrahiert: Sind I_1 und I_2 die exakten Interfacespezifikationen zwischen p und p_1 bzw. p_2, so ist $I_1 \| I_2$ zwar eine korrekte Interfacespezifikation zwischen p und $p_1 \| p_2$, jedoch i.allg. nicht mehr exakt. Das bedeutet nach Definition 1.23, daß die exakte Interfacespezifikation I_e zwischen p und $p_1 \| p_2$ bzgl. ihrer Sprache i.allg. kleiner ist als die korrekte Interfacespezifikation $I_1 \| I_2$.

Beispiel 4.15
Betrachte als Beispiel die Prozesse $p, p_1, p_2 \in$ Processes mit $\mathcal{A}_p = \{a, b\}$, $\mathcal{A}_{p_1} = \{a\}$ und $\mathcal{A}_{p_2} = \{b\}$ aus Abbildung 4.5, wobei I_e die exakte Interfacespezifikation zwischen p und $p_1 \| p_2$ bezeichnet.

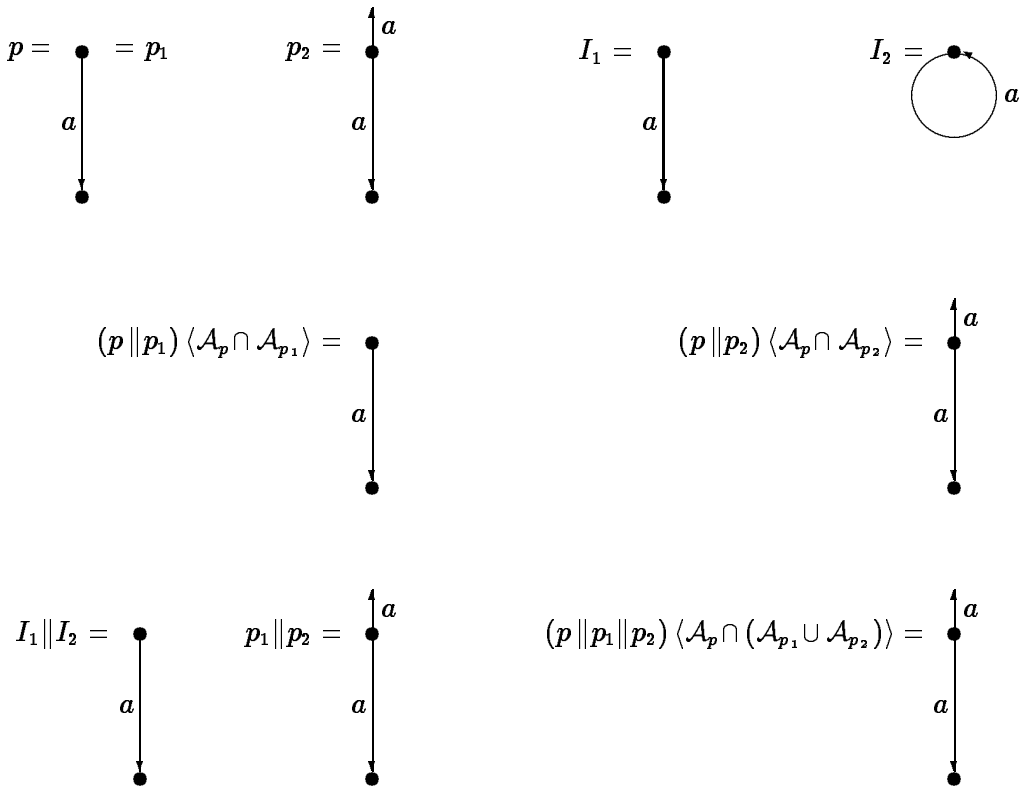


Abbildung 4.4: Gegenbeispiel zur korrekten Konstruktion von Interfacespezifikationen

Eine Methode zur kompositionellen Minimierung endlicher verteilter Systeme, welche Satz 4.13 benutzt, wird deshalb bzgl. einer Minimierung der algorithmischen Komplexität weniger effizient (oder sogar ineffektiv) sein (vgl. Proposition 2.18 (1)).

Effektive Konstruktion

Wie bereits erwähnt, geht man von korrekten Interfacespezifikationen zwischen den einzelnen total definierten Parallelkomponenten aus, da diese effektiv und sogar effizient berechnet werden können. Satz 4.13 beschreibt, wie man daraus global korrekte Interfacespezifikationen erhält. Im Beweis der Optimalität einer Methode kann deshalb direkt auf Satz 2.9 zurückgegriffen werden. Diese explizite Konstruktion würde jedoch, abgesehen vom oben diskutierten Nachteil bzgl. der Abstraktion, dem Prinzip der Vermeidung einer Zustandsexplosion widersprechen, da die Konstruktion von global korrekten Interfacespezifikationen gemäß Satz 4.13 auf der parallelen Komposition beruht, die für das Zustandsexplosionsproblem verantwortlich ist. Der folgende Satz zeigt einen Weg auf, wie man für eine Reduktion bzgl. einer globalen Interfacespezifikation auf die oben genannte explizite Konstruktion verzichten kann. Statt den Reduktionsoperator , bzgl. der globalen Interfacespezifikation auf einen Prozeß p anzuwenden, wird , sukzessive bzgl. der paarweisen Interfacespezifikationen auf p angewandt:

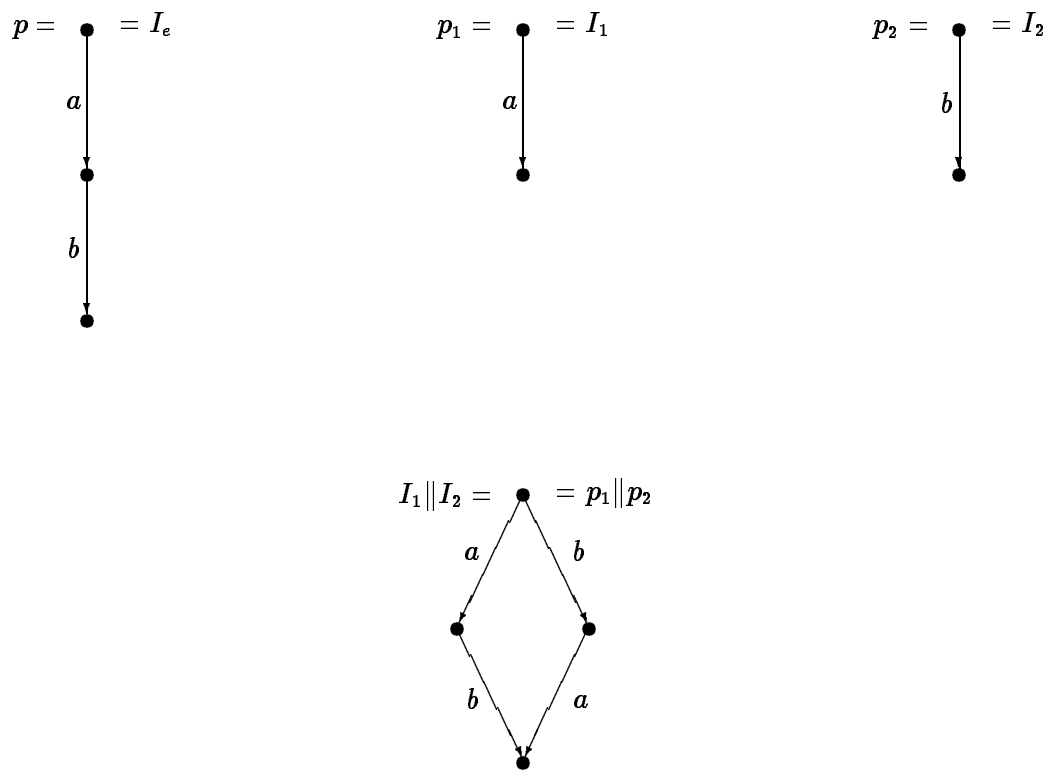


Abbildung 4.5: Gegenbeispiel zur Exaktheit von Interfacespezifikationen

Satz 4.16
Für alle total definierten Prozesse $p \in$ <u>TotDefProc</u> und alle Interfacespezifikationen $I_1, I_2 \in \mathcal{I}(p)$ gilt:

$$, _{I_1 \| I_2}(p) \;=\; , _{I_2}(, _{I_1}(p)).$$

Beweis
Seien $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \emptyset), p) \in$ <u>TotDefProc</u> und $I_1, I_2 \in \mathcal{I}(p)$ beliebig. Ferner gelte $, _{I_1 \| I_2}(p) = ((S_1, \mathcal{A}_1 \cup \{\tau\}, \longrightarrow_1, \emptyset), p)$ und $, _{I_2}(, _{I_1}(p)) = ((S_2, \mathcal{A}_2 \cup \{\tau\}, \longrightarrow_2, \emptyset), p)$.
Betrachte zunächst folgende Hilfsaussage für beliebige $q \in S_p$. Es gilt:

$$(*) \qquad \exists\, i_1 \in S_{I_1}\ \exists\, i_2 \in S_{I_2}.\ q \| i_1 \| i_2 \in S_{p \| I_1 \| I_2} \iff \exists\, i_1' \in S_{I_1}\ \exists\, i_2' \in S_{I_2}.\ q \| i_1' \in S_{p \| I_1} \ \wedge\ q \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2}.$$

Denn:

„ $\Rightarrow$ “:

$$\exists\, i_1 \in S_{I_1}\ \exists\, i_2 \in S_{I_2}.\ q\|i_1\|i_2 \in S_{p\|I_1\|I_2}$$

$$\begin{array}{lll}
(\text{Def. 1.8 \& Prop. 1.9}) & \Rightarrow & \exists\, i_1' \equiv i_1 \in S_{I_1}\ \exists\, i_2' \equiv i_2 \in S_{I_2}.\ q\|i_1' \in S_{p\|I_1}\ \wedge \\
 & & (q\|i_1)\|i_2 \in S_{(p\|I_1)\|I_2} \\
(\text{Def. 4.1 (1)}) & \Rightarrow & \exists\, i_1' \in S_{I_1}\ \exists\, i_2' \in S_{I_2}.\ q\|i_1' \in S_{p\|I_1}\ \wedge \\
 & & q\|i_2' \in S_{\Gamma_{I_1}(p)\|I_2}
\end{array}$$

„ $\Leftarrow$ “:

$$\exists\, i_1' \in S_{I_1}\ \exists\, i_2' \in S_{I_2}.\ q\|i_1' \in S_{p\|I_1}\ \wedge\ q\|i_2' \in S_{\Gamma_{I_1}(p)\|I_2}$$

$$\Rightarrow\ \exists\, i_1' \in S_{I_1}\ \exists\, i_2' \in S_{I_2}.\ q\|i_2' \in S_{\Gamma_{I_1}(p)\|I_2}$$

$$(\text{Def. 4.1 (1)})\ \Rightarrow\ \exists\, i_1 \in S_{I_1}\ \exists\, i_2 \equiv i_2' \in S_{I_2}.\ (q\|i_1)\|i_2 \in S_{(p\|I_1)\|I_2}$$

$$(\text{Prop. 1.9})\ \Rightarrow\ \exists\, i_1 \in S_{I_1}\ \exists\, i_2 \in S_{I_2}.\ q\|i_1\|i_2 \in S_{p\|I_1\|I_2}$$

Es gilt $,_{I_1\|I_2}(p) = ,_{I_2}(,_{I_1}(p))$, denn:

1. Wegen Definition 4.1 gilt $\mathcal{A}_p = \mathcal{A}_1 = \mathcal{A}_2$.
2. Für die Zustandsmengen gilt:

$$\begin{array}{lll}
 & & S_1 \\
(\text{Definition 4.1 (1)}) & = & \{q \in S_p \mid \exists\, i_1\|i_2 \in S_{I_1\|I_2}.\ q\|(i_1\|i_2) \in S_{p\|(I_1\|I_2)}\} \\
 & = & \{q \in S_p \mid \exists\, i_1 \in S_{I_1}\ \exists\, i_2 \in S_{I_2}.\ q\|(i_1\|i_2) \in S_{p\|(I_1\|I_2)}\} \\
(*) & = & \{q \in S_p \mid \exists\, i_1 \in S_{I_1}\ \exists\, i_2 \in S_{I_2}.\ q\|i_1 \in S_{p\|I_1}\ \wedge \\
 & & \qquad q\|i_2 \in S_{\Gamma_{I_1}(p)\|I_2}\} \\
(\text{Definition 4.1 (1)}) & = & \{q \in S_{\Gamma_{I_1}(p)} \mid \exists\, i_2 \in S_{I_2}.\ q\|i_2 \in S_{\Gamma_{I_1}(p)\|I_2}\} \\
(\text{Definition 4.1 (1)}) & = & S_2
\end{array}$$

3. Seien $p', p'' \in S_1$ und $a \in \mathcal{A}_1 \cup \{\tau\}$ beliebig. Dann gilt

$$p' \xrightarrow{a}_1 p'' \iff p' \xrightarrow{a}_2 p''.$$

Zum Beweis betrachte folgende vollständige Fallunterscheidung:

<u>**1. Fall:**</u> $a \notin \mathcal{A}_{I_1}$ und $a \notin \mathcal{A}_{I_2}$, also $a \notin \mathcal{A}_{I_1 \| I_2}$:

$$p' \xrightarrow{a}_1 p''$$

$$\begin{aligned}
&(\text{Definition 4.1 (3)}) && \iff && \exists\, i_1' \| i_2', i_1'' \| i_2'' \in S_{I_1 \| I_2}.\ p' \| (i_1' \| i_2') \xrightarrow{a}_{p \| (I_1 \| I_2)} p'' \| (i_1'' \| i_2'') \\
& && && \quad \wedge\ p' \| (i_1' \| i_2') \in S_{p \| (I_1 \| I_2)} \\
&(\text{Definition 1.8 (3)}) && \iff && \exists\, i_1' \| i_2', i_1'' \| i_2'' \in S_{I_1 \| I_2}.\ p' \xrightarrow{a}_p p'' \ \wedge\ i_1' \| i_2' = i_1'' \| i_2'' \ \wedge \\
& && && p' \| (i_1' \| i_2') \in S_{p \| (I_1 \| I_2)} \\
& && \iff && \exists\, i_1', i_1'' \in S_{I_1}\ \exists\, i_2', i_2'' \in S_{I_2}.\ p' \xrightarrow{a}_p p'' \ \wedge\ i_1' = i_1'' \ \wedge \\
& && && i_2' = i_2'' \ \wedge\ p' \| (i_1' \| i_2') \in S_{p \| (I_1 \| I_2)} \\
&(*) && \iff && \exists\, i_1', i_1'' \in S_{I_1}\ \exists\, i_2', i_2'' \in S_{I_2}.\ p' \xrightarrow{a}_p p'' \ \wedge\ i_1' = i_1'' \ \wedge \\
& && && i_2' = i_2'' \ \wedge\ p' \| i_1' \in S_{p \| I_1} \ \wedge\ p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\
&(\text{Definition 1.8 (3)}) && \iff && \exists\, i_1', i_1'' \in S_{I_1}\ \exists\, i_2', i_2'' \in S_{I_2}.\ p' \| i_1' \xrightarrow{a}_{p \| I_1} p'' \| i_1'' \ \wedge \\
& && && i_2' = i_2'' \ \wedge\ p' \| i_1' \in S_{p \| I_1} \ \wedge\ p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\
&(\text{Definition 4.1 (3)}) && \iff && \exists\, i_2', i_2'' \in S_{I_2}.\ p' \xrightarrow{a}_{\Gamma_{I_1}(p)} p'' \ \wedge\ i_2' = i_2'' \ \wedge \\
& && && p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\
&(\text{Definition 1.8 (3)}) && \iff && \exists\, i_2', i_2'' \in S_{I_2}.\ p' \| i_2' \xrightarrow{a}_{\Gamma_{I_1}(p) \| I_2} p'' \| i_2'' \ \wedge \\
& && && p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\
&(\text{Definition 4.1 (3)}) && \iff && p' \xrightarrow{a}_2 p''
\end{aligned}$$

<u>2. Fall:</u> $a \notin \mathcal{A}_{I_1}$ und $a \in \mathcal{A}_{I_2}$, also $a \in \mathcal{A}_{I_1 \| I_2} \subseteq \mathcal{A}_p$:

$$p' \xrightarrow{a}_1 p''$$

$$\begin{array}{lll}
(\text{Definition 4.1 (3)}) & \Longleftrightarrow & \exists\, i_1' \| i_2', i_1'' \| i_2'' \in S_{I_1 \| I_2} .\ p' \| (i_1' \| i_2') \xrightarrow{a}_{p \| (I_1 \| I_2)} p'' \| (i_1'' \| i_2'') \\
 & & \quad \wedge\ p' \| (i_1' \| i_2') \in S_{p \| (I_1 \| I_2)} \\
(\text{Definition 1.8 (5)}) & \Longleftrightarrow & \exists\, i_1' \| i_2', i_1'' \| i_2'' \in S_{I_1 \| I_2} .\ p' \xrightarrow{a}_p p'' \ \wedge \\
 & & i_1' \| i_2' \xrightarrow{a}_{I_1 \| I_2} i_1'' \| i_2'' \ \wedge\ p' \| (i_1' \| i_2') \in S_{p \| (I_1 \| I_2)} \\
(\text{Definition 1.8 (4)}) & \Longleftrightarrow & \exists\, i_1', i_1'' \in S_{I_1}\ \exists\, i_2', i_2'' \in S_{I_2} .\ p' \xrightarrow{a}_p p'' \ \wedge\ i_1' = i_1'' \ \wedge \\
 & & i_2' \xrightarrow{a}_{I_2} i_2'' \ \wedge\ p' \| (i_1' \| i_2') \in S_{p \| (I_1 \| I_2)} \\
(*) & \Longleftrightarrow & \exists\, i_1', i_1'' \in S_{I_1}\ \exists\, i_2', i_2'' \in S_{I_2} .\ p' \xrightarrow{a}_p p'' \ \wedge\ i_1' = i_1'' \ \wedge \\
 & & i_2' \xrightarrow{a}_{I_2} i_2'' \ \wedge\ p' \| i_1' \in S_{p \| I_1} \ \wedge\ p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\
(\text{Definition 1.8 (3)}) & \Longleftrightarrow & \exists\, i_1', i_1'' \in S_{I_1}\ \exists\, i_2', i_2'' \in S_{I_2} .\ p' \| i_1' \xrightarrow{a}_{p \| I_1} p'' \| i_1'' \ \wedge \\
 & & i_2' \xrightarrow{a}_{I_2} i_2'' \ \wedge\ p' \| i_1' \in S_{p \| I_1} \ \wedge\ p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\
(\text{Definition 4.1 (3)}) & \Longleftrightarrow & \exists\, i_2', i_2'' \in S_{I_2} .\ p' \xrightarrow{a}_{\Gamma_{I_1}(p)} p'' \ \wedge\ i_2' \xrightarrow{a}_{I_2} i_2'' \ \wedge \\
 & & p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\
(\text{Definition 1.8 (5)}) & \Longleftrightarrow & \exists\, i_2', i_2'' \in S_{I_2} .\ p' \| i_2' \xrightarrow{a}_{\Gamma_{I_1}(p) \| I_2} p'' \| i_2'' \ \wedge \\
 & & p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\
(\text{Definition 4.1 (3)}) & \Longleftrightarrow & p' \xrightarrow{a}_2 p''
\end{array}$$

<u>3. Fall:</u> $a \in \mathcal{A}_{I_1}$ und $a \notin \mathcal{A}_{I_2}$, also $a \in \mathcal{A}_{I_1 \| I_2} \subseteq \mathcal{A}_p$:

$$p' \xrightarrow{a}_1 p''$$

$$\begin{array}{lll}
(\text{Definition 4.1 (3)}) & \Longleftrightarrow & \exists\, i_1' \| i_2', i_1'' \| i_2'' \in S_{I_1 \| I_2} .\ p' \| (i_1' \| i_2') \xrightarrow{a}_{p \| (I_1 \| I_2)} p'' \| (i_1'' \| i_2'') \\
 & & \quad \wedge\ p' \| (i_1' \| i_2') \in S_{p \| (I_1 \| I_2)} \\
(\text{Definition 1.8 (5)}) & \Longleftrightarrow & \exists\, i_1' \| i_2', i_1'' \| i_2'' \in S_{I_1 \| I_2} .\ p' \xrightarrow{a}_p p'' \ \wedge \\
 & & i_1' \| i_2' \xrightarrow{a}_{I_1 \| I_2} i_1'' \| i_2'' \ \wedge\ p' \| (i_1' \| i_2') \in S_{p \| (I_1 \| I_2)} \\
(\text{Definition 1.8 (3)}) & \Longleftrightarrow & \exists\, i_1', i_1'' \in S_{I_1}\ \exists\, i_2', i_2'' \in S_{I_2} .\ p' \xrightarrow{a}_p p'' \ \wedge\ i_2' = i_2'' \ \wedge \\
 & & i_1' \xrightarrow{a}_{I_1} i_1'' \ \wedge\ p' \| (i_1' \| i_2') \in S_{p \| (I_1 \| I_2)} \\
(*) & \Longleftrightarrow & \exists\, i_1', i_1'' \in S_{I_1}\ \exists\, i_2', i_2'' \in S_{I_2} .\ p' \xrightarrow{a}_p p'' \ \wedge\ i_2' = i_2'' \ \wedge \\
 & & i_1' \xrightarrow{a}_{I_1} i_1'' \ \wedge\ p' \| i_1' \in S_{p \| I_1} \ \wedge\ p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\
(\text{Definition 1.8 (5)}) & \Longleftrightarrow & \exists\, i_1', i_1'' \in S_{I_1}\ \exists\, i_2', i_2'' \in S_{I_2} .\ p' \| i_1' \xrightarrow{a}_{p \| I_1} p'' \| i_1'' \ \wedge \\
 & & i_2' = i_2'' \ \wedge\ p' \| i_1' \in S_{p \| I_1} \ \wedge\ p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\
(\text{Definition 4.1 (3)}) & \Longleftrightarrow & \exists\, i_2', i_2'' \in S_{I_2} .\ p' \xrightarrow{a}_{\Gamma_{I_1}(p)} p'' \ \wedge\ i_2' = i_2'' \ \wedge \\
 & & p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\
(\text{Definition 1.8 (3)}) & \Longleftrightarrow & \exists\, i_2', i_2'' \in S_{I_2} .\ p' \| i_2' \xrightarrow{a}_{\Gamma_{I_1}(p) \| I_2} p'' \| i_2'' \ \wedge \\
 & & p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\
(\text{Definition 4.1 (3)}) & \Longleftrightarrow & p' \xrightarrow{a}_2 p''
\end{array}$$

<u>4. Fall:</u> $a \in \mathcal{A}_{I_1}$ und $a \in \mathcal{A}_{I_2}$, also $a \in \mathcal{A}_{I_1 \| I_2} \subseteq \mathcal{A}_p$:

$$p' \xrightarrow{a}_1 p''$$

$$\begin{array}{lcl}
(\text{Definition 4.1 (3)}) & \Longleftrightarrow & \exists\, i_1' \| i_2', i_1'' \| i_2'' \in S_{I_1 \| I_2}.\ p' \| (i_1' \| i_2') \xrightarrow{a}_{p \| (I_1 \| I_2)} p'' \| (i_1'' \| i_2'') \\
 & & \quad \wedge\ p' \| (i_1' \| i_2') \in S_{p \| (I_1 \| I_2)} \\[2ex]
(\text{Definition 1.8 (5)}) & \Longleftrightarrow & \exists\, i_1' \| i_2', i_1'' \| i_2'' \in S_{I_1 \| I_2}.\ p' \xrightarrow{a}_p p'' \ \wedge \\
 & & i_1' \| i_2' \xrightarrow{a}_{I_1 \| I_2} i_1'' \| i_2'' \ \wedge\ p' \| (i_1' \| i_2') \in S_{p \| (I_1 \| I_2)} \\[2ex]
(\text{Definition 1.8 (3)}) & \Longleftrightarrow & \exists\, i_1', i_1'' \in S_{I_1}\ \exists\, i_2', i_2'' \in S_{I_2}.\ p' \xrightarrow{a}_p p'' \ \wedge\ i_1' \xrightarrow{a}_{I_1} i_1'' \ \wedge \\
 & & i_2' \xrightarrow{a}_{I_2} i_2'' \ \wedge\ p' \| (i_1' \| i_2') \in S_{p \| (I_1 \| I_2)} \\[2ex]
(*) & \Longleftrightarrow & \exists\, i_1', i_1'' \in S_{I_1}\ \exists\, i_2', i_2'' \in S_{I_2}.\ p' \xrightarrow{a}_p p'' \ \wedge\ i_1' \xrightarrow{a}_{I_1} i_1'' \ \wedge \\
 & & i_2' \xrightarrow{a}_{I_2} i_2'' \ \wedge\ p' \| i_1' \in S_{p \| I_1} \ \wedge\ p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\[2ex]
(\text{Definition 1.8 (5)}) & \Longleftrightarrow & \exists\, i_1', i_1'' \in S_{I_1}\ \exists\, i_2', i_2'' \in S_{I_2}.\ p' \| i_1' \xrightarrow{a}_{p \| I_1} p'' \| i_1'' \ \wedge \\
 & & i_2' \xrightarrow{a}_{I_2} i_2'' \ \wedge\ p' \| i_1' \in S_{p \| I_1} \ \wedge\ p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\[2ex]
(\text{Definition 4.1 (3)}) & \Longleftrightarrow & \exists\, i_2', i_2'' \in S_{I_2}.\ p' \xrightarrow{a}_{\Gamma_{I_1}(p)} p'' \ \wedge\ i_2' \xrightarrow{a}_{I_2} i_2'' \ \wedge \\
 & & p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\[2ex]
(\text{Definition 1.8 (5)}) & \Longleftrightarrow & \exists\, i_2', i_2'' \in S_{I_2}.\ p' \| i_2' \xrightarrow{a}_{\Gamma_{I_1}(p) \| I_2} p'' \| i_2'' \ \wedge \\
 & & p' \| i_2' \in S_{\Gamma_{I_1}(p) \| I_2} \\[2ex]
(\text{Definition 4.1 (3)}) & \Longleftrightarrow & p' \xrightarrow{a}_2 p''
\end{array}$$

Der soeben bewiesene Satz gilt auch für partiell definierte Prozesse bzgl. des Reduktionsoperators $\overline{\Pi}$. Da der Satz 4.13 jedoch nur für total definierte Prozesse gültig ist, reicht die Formulierung des Satzes für die in diesem Kapitel beabsichtigten Zwecke aus. Weiterhin erlaubt der Satz, aus dem bereits bekannten Satz 2.9 ein für die Optimalität einer Methode wichtige Proposition folgern zu können:

Proposition 4.17 (Kontexttreue der Reduktion)
Für alle total definierten Prozesse $p, p_1, p_2 \in$ <u>TotDefProc</u> sowie alle korrekten Interfacespezifikationen $I_1 \in \mathcal{I}(p, p_1)$ und $I_2 \in \mathcal{I}(p, p_2)$ gilt:

$$,_{I_2}(,_{I_1}(p)) \| p_1 \| p_2 \ = \ p \| p_1 \| p_2 .$$

Beweis
Die Behauptung folgt sofort mit den Sätzen 4.13, 4.16, 2.9 und Proposition 1.9.

4.4 Schlußfolgerungen und zukünftige Forschungsarbeiten

In diesem Kapitel wurden Ansätze vorgestellt, um die Effizienz und die Praktikabilität der $\mathcal{RM}$–Methode aus Kapitel 3 bzgl. des Reduktionsoperators $\overline{\Pi}$ zu verbessern.

Der erste Verbesserungsansatz (Abschnitt 4.1) betrifft die sog. „Don't–Care"–Undefiniertheiten, die bei der Bestimmung eines reduzierten Prozesses zwar mitberechnet, aber bis jetzt nicht ausgenutzt worden sind. Damit die Effizienz durch Ausnutzung dieser Don't–Cares verbessert werden kann, bedarf es einer Strategie zu ihrer Behandlung. Dieses Problem tritt in vielen anderen Zusammenhängen ebenfalls auf, stellvertretend seien hier *Schaltkreise* erwähnt. Es ist zu untersuchen, ob die dort entwickelten Lösungsansätze auf die in dieser Arbeit beschriebene Anwendung adaptierbar sind.

In Abschnitt 4.2 ist gezeigt worden, daß die $\mathcal{RM}$–Methode aus Kapitel 3 für total definierte Prozesse und korrekte Interfacespezifikationen bzgl. der algorithmischen Komplexität effizienter ist. Tatsächlich erfüllt unter diesen Voraussetzungen der spezielle Reduktionsoperator $\overline{\Pi}$ bzw. , eine Vielzahl von algebraischen Eigenschaften, welche i.allg. nicht gelten. Diese Beobachtungen führen zum zweiten Verbesserungsansatz.

Der zweite Verbesserungsansatz (Abschnitt 4.3) betrifft die Konstruktion von korrekten Interfacespezifikationen aus paarweise korrekten Interfacespezifikationen. Die vorgestellte Konstruktion abstrahiert jedoch so stark, daß eine Anwendung innerhalb einer Methode nur für sehr spezielle Systeme, bei denen die Betrachtung der Reihenfolge der einzelnen Parallelkomponenten in jener Methode sehr wichtig ist, Sinn macht. In zukünftigen Arbeiten, die diesen Ansatz verfolgen, muß deshalb untersucht werden, ob der Ansatz für die beabsichtigten Zwecke überhaupt ausdruckskräftig genug ist, und ggf., ob die angesprochene Abstraktion durch andere Konstruktionen minimiert werden kann.

Kapitel 5

Zusammenfassung

Übersicht über die wesentlichen Eigenschaften der vorgestellten Methode

Es wurde eine halbautomatische Methode zur kompositionellen Minimierung endlicher verteilter Systeme vorgestellt, die zum Ziel hat, die während der Konstruktion von Transitionssystemen i.allg. auftretende Zustandsexplosion zu verhindern. Die bei dieser Konstruktion entstehenden Zwischentransitionssysteme werden zum einen bzgl. lokaler Kontextabhängigkeiten und zum anderen bzgl. globaler Kontextabhängigkeiten minimiert. Ersteres geschieht, wie auch bei „naiven" Methoden üblich, mit Hilfe einer Äquivalenzrelation (hier der Beobachtungsäquivalenz). Letzteres bewerkstelligt ein Reduktionsoperator, der anhand der vom Programmentwickler bereitgestellten Interfacespezifikationen, welche den globalen Kontext beschreiben, minimiert. In dieser Arbeit wurden sowohl die theoretischen Eigenschaften als auch eine algorithmische Realisierung eines speziellen Reduktionsoperators dargestellt.

Der Vorteil der hier vorgestellten Methode gegenüber anderen ist, daß Interfacespezifikationen nicht notwendigerweise korrekt sein müssen, um das resultierende minimierte Transitionssystem für Beweise (etwa von Programmeigenschaften) verwenden zu können. Läßt sich anhand des resultierenden Transitionssystems der Beweis führen, so gilt das Bewiesene auch für das ursprüngliche Transitionssystem. Die Methode leistet jedoch noch mehr. Wurde das Beweisziel verfehlt, obwohl die Methode eine korrekte Minimierung anzeigt, so wird das Beweisziel auch mit Hilfe des ursprünglichen Transitionssystems verfehlt. Weiterhin wurde gezeigt, daß aufgrund verschiedener Eigenschaften der Methode ihre Effizienz in Spezialfällen (wie etwa der totalen Definiertheit des betrachteten Systems und der ausschließlichen Betrachtung korrekter Interfacespezifikationen) gesteigert werden kann.

Ausblicke

Aus der kritischen Betrachtung der Methode ist der Wunsch entsprungen, korrekte Interfacespezifikationen aus paarweise korrekten Interfacespezifikationen automatisch konstru-

ieren zu können. Die dazu vorgestellte (naive) Technik abstrahiert jedoch von der durch die Interfacespezifikationen beschriebenen Information so sehr, daß sie für die Methode nur in Spezialfällen geeignet ist. Adäquatere Techniken zur Lösung dieses Problems sind derzeit nicht bekannt. Es ist daher zunächst zu zeigen, ob paarweise korrekte Interfacespezifikationen überhaupt ausdruckskräftig genug sind, um ein System geeignet beschreiben zu können. Wäre dies der Fall, so ist weiterhin zu klären, ob daraus eine Technik (ggf. approximativ) für eine automatische Methode abgeleitet werden kann.

Ferner ist eine Realisierung der Methode im Rahmen des *PASSAU–Projektes* [St92/2] vorgesehen.

Anhang A

Mathematische Grundlagen

In diesem Abschnitt werden einige mengentheoretische und algebraische Grundbegriffe angegeben. Insbesondere werden die Begriffe *Äquivalenzrelation*, *Quasiordnung* und *Halbordnung* sowie die *Monotonie* und *Additivität* von Abbildungen definiert. Darüberhinaus werden einfache Aussagen und Zusammenhänge hergeleitet, welche u.a. beim Beweis der Charakterisierung eines *Fixpunktes* durch *Fixpunktiteration* von Nutzen sind.

Äquivalenzrelationen

Definition A.1 (Äquivalenzrelation)
Sei D eine Menge und $R \subseteq D \times D$ eine binäre Relation auf D. Gilt für alle $d_1, d_2, d_3 \in D$

1. *$(d_1, d_2) \in R$ und $(d_2, d_3) \in R$ impliziert $(d_1, d_3) \in R$ (Transitivität),*
2. *$(d_1, d_2) \in R$ impliziert $(d_2, d_1) \in R$ (Symmetrie) und*
3. *$(d_1, d_1) \in R$ (Reflexivität),*

so heißt R eine Äquivalenzrelation auf D. Statt $(d_1, d_2) \in R$ schreibt man gewöhnlich auch $d_1 \; R \; d_2$.

Ist R eine Äquivalenzrelation auf D und $d \in D$, so heißt die Menge $\{d' \in D \mid d \, R \, d'\}$ *Äquivalenzklasse* von d. Eine Äquivalenzrelation R auf D bewirkt eine *Partitionierung* von D, d.h. je zwei Äquivalenzklassen sind entweder disjunkt oder gleich.

Quasiordnungen

Definition A.2 (Quasiordnung)
Sei D eine Menge und $\sqsubseteq_D \subseteq D \times D$ eine Relation auf D derart, daß für alle $d_1, d_2, d_3 \in D$ gilt:

1. *$d_1 \sqsubseteq_D d_2$ und $d_2 \sqsubseteq_D d_3$ impliziert $d_1 \sqsubseteq_D d_3$ (Transitivität) und*
2. *$d_1 \sqsubseteq_D d_1$ (Reflexivität).*

Dann heißt $\sqsubseteq_D$ eine Quasiordnungsrelation auf D. Ein Paar $\langle D; \sqsubseteq_D \rangle$, bestehend aus einer Menge D und einer Quasiordnungsrelation $\sqsubseteq_D$ auf D, heißt Quasiordnung.

Im folgenden wird der Index D weggelassen, falls er aus dem Kontext hervorgeht.

Halbordnungen

Eine *Halbordnung* ist eine Quasiordnung, deren Relation zusätzlich *antisymmetrisch* ist.

Definition A.3 (Halbordnung)
Eine Quasiordnung $\langle D; \sqsubseteq_D \rangle$ heißt Halbordnung, falls für alle $d_1, d_2 \in D$ gilt:

$$d_1 \sqsubseteq_D d_2 \text{ und } d_2 \sqsubseteq_D d_1 \text{ impliziert } d_1 = d_2 \text{ (Antisymmetrie).}$$

Statt $d_1 \sqsubseteq_D d_2$ schreibt man auch $d_2 \sqsupseteq_D d_1$.

Sei $\mathcal{P}$ der *Potenzmengenoperator*, d.h. $\mathcal{P}(M) =_{df} \{N \mid N \subseteq M\}$ für eine beliebige Menge M.

Lemma A.4
$\langle \mathcal{P}(M); \subseteq \rangle$ ist eine Halbordnung.

Beweis
Seien $A, B, C \in \mathcal{P}(M)$ beliebig.

- Reflexivität: Wegen $A = A$ gilt auch $A \subseteq A$.
- Transitivität: $A \subseteq B \subseteq C$ impliziert $A \subseteq C$.
- Antisymmetrie: $A \subseteq B$ und $B \subseteq A$ impliziert $A = B$.

Mit Definition A.3 folgt die Behauptung.

Definition A.5 (Kette, Supremum, Vollständigkeit)
Sei $\langle D; \sqsubseteq_D \rangle$ eine Halbordnung, $T \subseteq D$ und $d \in D$.

1. *T heißt Kette in D, falls für alle $d', d'' \in T$ $d' \sqsubseteq_D d''$ oder $d'' \sqsubseteq_D d'$ gilt. Die Länge der Kette T beträgt $\mid T \mid$, falls T abzählbar ist.*
2. *d heißt obere Schranke von T, falls für alle $d' \in T$ $d' \sqsubseteq_D d$ gilt.*
3. *d heißt kleinste obere Schranke oder Supremum von T in D, falls d eine obere Schranke von T ist und für alle oberen Schranken d' von T $d \sqsubseteq_D d'$ gilt.*

4. *$\langle D; \sqsubseteq_D \rangle$ heißt mengenvollständig (kettenvollständig), falls für jede Menge (Kette) $T \subseteq D$ das Supremum von T, bezeichnet mit $\bigsqcup_D T$, existiert. Statt kettenvollständig sagt man auch kurz vollständig.*

Für $\bigsqcup_D \{d_1, d_2\}$ wird im folgenden auch $d_1 \sqcup_D d_2$ geschrieben. Im Falle ihrer Existenz sind Suprema eindeutig bestimmt. In einer mengenvollständigen Halbordnung $\langle D; \sqsubseteq_D \rangle$ existiert insbesondere $\perp =_{df} \bigsqcup_D \emptyset$. Nach Definition des Supremums gilt für alle $d \in D$: $\perp \sqsubseteq_D d$.

Lemma A.6
Die Halbordnung $\langle \mathcal{P}(M); \subseteq \rangle$, wobei M eine beliebige Menge bezeichnet, ist mengenvollständig.

Beweis
Sei $T \subseteq \mathcal{P}(M)$ beliebig. Dann ist $S =_{df} \bigcup \{ N \mid N \in T \}$ das Supremum von T, denn:

1. S ist obere Schranke von T, da nach Definition von S für jedes $N \in T$ $N \subseteq S$ gilt.
2. Angenommen S ist nicht *kleinste* obere Schranke (Supremum) von T, d.h. es gibt eine obere Schranke S' von T mit $S' \subseteq S$. Nach Definition einer oberen Schranke gilt für alle $N \in T$ $N \subseteq S'$ und somit auch $S \subseteq S'$. Zusammen mit Lemma A.4 (Antisymmetrie) folgt $S = S'$.

Somit folgt die Behauptung mit Definition A.5 (4).

Im folgenden werden *Abbildungen* zwischen den Trägermengen von Halbordnungen betrachtet:

Definition A.7 (Bijektion, Monotonie, Additivität und Fixpunkt)
Seien $\langle D; \sqsubseteq_D \rangle$ und $\langle D'; \sqsubseteq_{D'} \rangle$ Halbordnungen und $\phi : D \longrightarrow D'$ eine Abbildung von D nach D'.

1. *Die Abbildung ϕ heißt bijektiv, falls gilt:*
   - $\forall\, d_1, d_2 \in D.\ \phi(d_1) =_{D'} \phi(d_2)$ *impliziert* $d_1 =_D d_2$ *(Injektivität) und*
   - $\forall\, d' \in D'\ \exists\, d \in D.\ \phi(d) =_{D'} d'$ *(Surjektivität).*
2. *Gilt für alle $d_1, d_2 \in D$*

$$d_1 \sqsubseteq_D d_2 \ \text{ impliziert } \ \phi(d_1) \sqsubseteq_{D'} \phi(d_2)\,,$$

   so heißt ϕ monoton oder ordnungserhaltend.

3. *Gilt für alle nicht–leeren Ketten $T \subseteq D$*

$$\phi(\textstyle\bigsqcup_D T) = \bigsqcup_{D'} \{\phi(d) \mid d \in T\}\,,$$

   so heißt ϕ additiv.

4. *Ist $\langle D; \sqsubseteq_D \rangle = \langle D'; \sqsubseteq_{D'} \rangle$, so heißt $d \in D$ Fixpunkt von ϕ, falls $\phi(d) = d$ gilt. $d \in D$ heißt kleinster Fixpunkt von ϕ, in Zeichen $d = \mathbf{fix}(\phi)$, falls d Fixpunkt von ϕ ist und für alle Fixpunkte d' von ϕ gilt: $d \sqsubseteq_D d'$.*

Beachte, daß die Additivität einer Abbildung bereits die Monotonie der Abbildung impliziert und die Komposition additiver Funktionen wieder additiv ist:

Lemma A.8
Seien $\langle D; \sqsubseteq_D \rangle$, $\langle D'; \sqsubseteq_{D'} \rangle$ und $\langle D''; \sqsubseteq_{D''} \rangle$ mengenvollständige Halbordnungen sowie $\phi : D \longrightarrow D'$ und $\phi' : D' \longrightarrow D''$ additive Funktionen. Dann ist auch $\phi' \circ \phi : D \longrightarrow D''$ eine additive Funktion.

Beweis
Sei $\emptyset \neq T \subseteq D$ beliebig. Die Behauptung folgt nun mit Definition A.7 (3) aus

$$\begin{aligned} & \textstyle\bigsqcup_{D''} \phi'(\phi(T)) \\ (\text{Additivität von } \phi') \quad &= \quad \phi'(\textstyle\bigsqcup_{D'} \phi(T)) \\ (\text{Additivität von } \phi) \quad &= \quad \phi'(\phi(\textstyle\bigsqcup_{D} T)) \end{aligned}$$

Für die *Fixpunkttheorie* ist die folgende Spezialisierung des **Satzes von Knaster und Tarski** zentral:

Satz A.9 (Fixpunktiteration)
Für additive Funktionen $\phi : D \longrightarrow D$, welche auf einer mengenvollständigen Halbordnung $\langle D; \sqsubseteq \rangle$ definiert sind, existiert der kleinste Fixpunkt $\mathbf{fix}(\phi)$ und es gilt:

$$\mathbf{fix}(\phi) = \bigsqcup \{\phi^n(\bot) \mid n \in \quad \}.$$

Beweis
Zunächst ist $\mathbf{fix}(\phi)$ wohldefiniert, da $\langle D; \sqsubseteq \rangle$ nach Voraussetzung mengenvollständig ist.

Die Fixpunkteigenschaft von $\mathbf{fix}(\phi)$ folgt aus:

$$
\begin{aligned}
& \phi(\mathbf{fix}(\phi)) \\
(\text{Definition von } \mathbf{fix}(\phi)) \quad &= \quad \phi(\bigsqcup\{\phi^n(\bot) \mid n \in \quad \}) \\
(\text{Additivität von } \phi) \quad &= \quad \bigsqcup\{\phi^{n+1}(\bot) \mid n \in \quad \} \\
&= \quad \bigsqcup\{\phi^n(\bot) \mid n \in \quad \setminus\{0\}\} \\
(d = d \sqcup \bot) \quad &= \quad \bigsqcup\{\phi^n(\bot) \mid n \in \quad \setminus\{0\}\} \sqcup \bot \\
&= \quad \bigsqcup\{\phi^n(\bot) \mid n \in \quad \} \\
(\text{Definition von } \mathbf{fix}(\phi)) \quad &= \quad \mathbf{fix}(\phi)
\end{aligned}
$$

Um zu zeigen, daß $\mathbf{fix}(\phi)$ der kleinste Fixpunkt ist, betrachtet man einen beliebigen Fixpunkt $d_{\text{fix}} \in D$ von ϕ. Dann gilt $\bot \sqsubseteq d_{\text{fix}}$ und damit wegen der Monotonie von ϕ für $n \in \quad$: $\phi^n(\bot) \sqsubseteq \phi^n(d_{\text{fix}}) = d_{\text{fix}}$. Also folgt

$$\mathbf{fix}(\phi) = \bigsqcup\{\phi^n(\bot) \mid n \in \quad \} \sqsubseteq d_{\text{fix}}.$$

Anhang B

Beweise

Beweis zur Parallel–/Fensteroperatorkorrespondenz

Hier wird der Beweis von Lemma 1.10 vorgeführt, welcher eine umfangreiche Fallunterscheidung bzgl. der operationellen Semantikregeln gemäß Definition 1.8 erfordert.

Die Aussage von Lemma 1.10:

> Für alle Prozesse $p, q \in \underline{\text{Processes}}$ und alle Mengen L, L' sichtbarer Aktionen gilt:
>
> $$(p \| q)\langle L\rangle \cong (p\langle L'\rangle \| q)\langle L\rangle \text{ , falls } L' \supseteq L \cup (\mathcal{A}_p \cap \mathcal{A}_q).$$

Beweis
Seien $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p)$, $q = ((S_q, \mathcal{A}_q \cup \{\tau\}, \longrightarrow_q, \uparrow_q), q) \in \underline{\text{Processes}}$, $L' \supseteq L \cup (\mathcal{A}_p \cap \mathcal{A}_q)$. Ferner gelten die folgenden Bezeichnungen:

$$\begin{aligned}
(p \| q)\langle L\rangle &= ((S_1, ((\mathcal{A}_p \cup \mathcal{A}_q) \cap L) \cup \{\tau\}, \longrightarrow_1, \uparrow_1), p \| q) \text{ sowie} \\
(p\langle L'\rangle \| q)\langle L\rangle &= ((S_2, (((\mathcal{A}_p \cap L') \cup \mathcal{A}_q) \cap L) \cup \{\tau\}, \longrightarrow_2, \uparrow_2), p \| q).
\end{aligned}$$

Es sind die Bedingungen (1)–(3) der Definition 1.6 zu zeigen, da die Aktionenmengen von $(p \| q)\langle L\rangle$ und $(p\langle L'\rangle \| q)\langle L\rangle$ wegen der Forderung $L' \supseteq L \cup (\mathcal{A}_p \cap \mathcal{A}_q)$ übereinstimmen. Definiere dazu die bijektive Abbildung $\nu : S_1 \longrightarrow S_2$ durch $\nu((p' \| q')\langle L\rangle) =_{df} (p'\langle L'\rangle \| q')\langle L\rangle$ und ferner für $i = 1, 2$:

- S_i^n, die Teilmenge aller in n Schritten vom Startzustand aus erreichbaren Zustände,
- $\longrightarrow_i^n$, die Menge aller von einem Zustand in S_i^n ausgehender Transitionen, und
- $\uparrow_i^n$, das Undefiniertheitsprädikat der Zustände in S_i^n.

Führe nun eine Induktion über n durch[1]:

[1] Beachte, daß hier ausschließlich endliche Transitionssysteme betrachtet werden.

Wegen $\nu(S_1^0) = \nu(\{(p\|q)\langle L\rangle\}) = \{(p\langle L'\rangle\|q)\langle L\rangle\} = S_2^0$ (Dies ist bereits Punkt (1) der Definition 1.6.) ist der Induktionsanfang trivial. Es genügt, den Induktionsschritt für $n \geq 1$ unter der Induktionsannahme $\nu(S_1^{n-1}) = S_2^{n-1}$ durchzuführen, um den Beweis zu vervollständigen. Es ist also folgendes zu zeigen:

1. $(p'\|q')\langle L\rangle \xrightarrow{a}{}_1^{n-1} (p''\|q'')\langle L\rangle \iff (p'\langle L'\rangle\|q')\langle L\rangle \xrightarrow{a}{}_2^{n-1} (p''\langle L'\rangle\|q'')\langle L\rangle$,
2. $\nu(S_1^n) = S_2^n$ und
3. $(p'\|q')\langle L\rangle\uparrow_1^{n-1} a \iff (p'\langle L'\rangle\|q')\langle L\rangle\uparrow_2^{n-1} a$.

Zunächst ist Punkt (1) des Induktionsschrittes gemäß der Regeln aus Definition 1.8 zu verifizieren:

1. Fall: $a \neq \tau$

Regel 3:

$$(p'\|q')\langle L\rangle \xrightarrow{a}{}_1 (p''\|q')\langle L\rangle \quad , a \in \mathcal{A}_p \setminus \mathcal{A}_q$$

$$\begin{array}{lcl}
(\text{Regel 1}, a \neq \tau) & \iff & p'\|q' \xrightarrow{a}{}_{p\|q} p''\|q' \quad , a \in L\,, a \in \mathcal{A}_p \setminus \mathcal{A}_q \\
(\text{Regel 3}, a \notin \mathcal{A}_q) & \iff & p' \xrightarrow{a}{}_p p'' \quad , a \in L\,, a \in \mathcal{A}_p \setminus \mathcal{A}_q \\
(\text{Regel 1}, a \in L') & \iff & p'\langle L'\rangle \xrightarrow{a}{}_{p\langle L'\rangle} p''\langle L'\rangle \quad , a \in L\,, a \in \mathcal{A}_{p\langle L'\rangle} \setminus \mathcal{A}_q \\
(\text{Regel 3}, a \notin \mathcal{A}_q) & \iff & p'\langle L'\rangle\|q' \xrightarrow{a}{}_{p\langle L'\rangle\|q} p''\langle L'\rangle\|q' \quad , a \in L\,, a \in \mathcal{A}_{p\langle L'\rangle} \setminus \mathcal{A}_q \\
(\text{Regel 1}, a \neq \tau) & \iff & (p'\langle L'\rangle\|q')\langle L\rangle \xrightarrow{a}{}_2 (p''\langle L'\rangle\|q')\langle L\rangle \quad , a \in \mathcal{A}_{p\langle L'\rangle} \setminus \mathcal{A}_q
\end{array}$$

Regel 4:

$$(p'\|q')\langle L\rangle \xrightarrow{a}{}_1 (p'\|q'')\langle L\rangle \quad , a \in \mathcal{A}_q \setminus \mathcal{A}_p$$

$$\begin{array}{lcl}
(\text{Regel 1}, a \neq \tau) & \iff & p'\|q' \xrightarrow{a}{}_{p\|q} p'\|q'' \quad , a \in L\,, a \in \mathcal{A}_q \setminus \mathcal{A}_p \\
(\text{Regel 4}, a \notin \mathcal{A}_p) & \iff & q' \xrightarrow{a}{}_q q'' \quad , a \in L\,, a \in \mathcal{A}_q \setminus \mathcal{A}_p \\
(\text{Regel 4})^2 & \iff & p'\langle L'\rangle\|q' \xrightarrow{a}{}_{p\langle L'\rangle\|q} p'\langle L'\rangle\|q'' \quad , a \in L\,, a \in \mathcal{A}_q \setminus \mathcal{A}_{p\langle L'\rangle} \\
(\text{Regel 1}, a \neq \tau) & \iff & (p'\langle L'\rangle\|q')\langle L\rangle \xrightarrow{a}{}_2 (p'\langle L'\rangle\|q'')\langle L\rangle \quad , a \in \mathcal{A}_q \setminus \mathcal{A}_{p\langle L'\rangle}
\end{array}$$

[2] Beachte: $a \notin \mathcal{A}_p \iff$ (wegen $a \in L \subseteq L'$) $a \notin \mathcal{A}_p \cap L' = \mathcal{A}_{p\langle L'\rangle}$.

Regel 5:

$$(p'\|q')\langle L\rangle \xrightarrow{a}_1 (p''\|q'')\langle L\rangle \quad , a \in \mathcal{A}_p \, , \, a \in \mathcal{A}_q$$

$$\begin{array}{lcl}
(\text{Regel 1}, a \neq \tau) & \Longleftrightarrow & p'\|q' \xrightarrow{a}_{p\|q} p''\|q'' \quad , a \in L \, , \, a \in \mathcal{A}_p \, , \, a \in \mathcal{A}_q \\
(\text{Regel 5}) & \Longleftrightarrow & p' \xrightarrow{a}_p p'' \, , \, q' \xrightarrow{a}_q q'' \quad , a \in L \, , \, a \in \mathcal{A}_p \, , \, a \in \mathcal{A}_q \\
(\text{Regel 1}, a \in L') & \Longleftrightarrow & p'\langle L'\rangle \xrightarrow{a}_{p\langle L'\rangle} p''\langle L'\rangle \, , \, q' \xrightarrow{a}_q q'' \quad , a \in L \, , \, a \in \mathcal{A}_{p\langle L'\rangle} \, , \, a \in \mathcal{A}_q \\
(\text{Regel 5}) & \Longleftrightarrow & p'\langle L'\rangle\|q' \xrightarrow{a}_{p\langle L'\rangle\|q} p''\langle L'\rangle\|q'' \quad , a \in L \, , \, a \in \mathcal{A}_{p\langle L'\rangle} \, , \, a \in \mathcal{A}_q \\
(\text{Regel 1}, a \neq \tau) & \Longleftrightarrow & (p'\langle L'\rangle\|q')\langle L\rangle \xrightarrow{a}_2 (p''\langle L'\rangle\|q'')\langle L\rangle \quad , a \in \mathcal{A}_{p\langle L'\rangle} \, , \, a \in \mathcal{A}_q
\end{array}$$

<u>2. Fall: $a = \tau$</u>

Regel 3:

<u>„ $\Rightarrow$ “:</u>

$$(p'\|q')\langle L\rangle \xrightarrow{\tau}_1 (p''\|q')\langle L\rangle$$

$$\begin{array}{lcl}
(\text{Regel 2}) & \Rightarrow & p'\|q' \xrightarrow{b}_{p\|q} p''\|q' \quad , (b \notin L \, , \, b \in \mathcal{A}_p \setminus \mathcal{A}_q) \vee (b = \tau) \\
(\text{Regel 3}, b \notin \mathcal{A}_q) & \Rightarrow & p' \xrightarrow{b}_p p'' \quad , (b \notin L \, , \, b \in \mathcal{A}_p \setminus \mathcal{A}_q) \vee (b = \tau)
\end{array}$$

Fall 1: $b \notin L'$

$$\begin{array}{lcl}
(\text{Regel 2}) & \Rightarrow & p'\langle L'\rangle \xrightarrow{\tau}_{p\langle L'\rangle} p''\langle L'\rangle \\
(\text{Regel 3}, \tau \notin \mathcal{A}_q) & \Rightarrow & p'\langle L'\rangle\|q' \xrightarrow{\tau}_{p\langle L'\rangle\|q} p''\langle L'\rangle\|q' \\
(\text{Regel 2}) & \Rightarrow & (p'\langle L'\rangle\|q')\langle L\rangle \xrightarrow{\tau}_2 (p''\langle L'\rangle\|q')\langle L\rangle
\end{array}$$

Fall 2: $b \in L'$

$$\begin{array}{lcl}
(\text{Regel 1}) & \Rightarrow & p'\langle L'\rangle \xrightarrow{b}_{p\langle L'\rangle} p''\langle L'\rangle \quad , (b \notin L \, , \, b \in \mathcal{A}_p \setminus \mathcal{A}_q) \vee (b = \tau) \\
(\text{Regel 3}, b \notin \mathcal{A}_q) & \Rightarrow & p'\langle L'\rangle\|q' \xrightarrow{b}_{p\langle L'\rangle\|q} p''\langle L'\rangle\|q' \quad , (b \notin L) \vee (b = \tau) \\
(\text{Regel 2}) & \Rightarrow & (p'\langle L'\rangle\|q')\langle L\rangle \xrightarrow{\tau}_2 (p''\langle L'\rangle\|q')\langle L\rangle
\end{array}$$

„ $\Leftarrow$ “: Betrachte folgende vollständige Fallunterscheidung gemäß den Regeln zur operationellen Semantik (vgl. Definition 1.8):

$$(p'\langle L'\rangle \| q')\langle L\rangle \xrightarrow{\tau}_2 (p''\langle L'\rangle \| q')\langle L\rangle$$

Fall 1:

(Regel 2) $\Rightarrow$ $p'\langle L'\rangle \| q' \xrightarrow{b}_{p\langle L'\rangle \| q} p''\langle L'\rangle \| q'$, $b \notin L$, $b \neq \tau$, $b \in \mathcal{A}_{p\langle L'\rangle} \setminus \mathcal{A}_q$

(Regel 3, $b \notin \mathcal{A}_q$) $\Rightarrow$ $p'\langle L'\rangle \xrightarrow{b}_{p\langle L'\rangle} p''\langle L'\rangle$, $b \notin L$, $b \neq \tau$, $b \in \mathcal{A}_p \setminus \mathcal{A}_q$

(Regel 1, $b \neq \tau$) $\Rightarrow$ $p' \xrightarrow{b}_p p''$, $b \notin L$, $b \in \mathcal{A}_p \setminus \mathcal{A}_q$

(Regel 3, $b \notin \mathcal{A}_q$) $\Rightarrow$ $p' \| q' \xrightarrow{b}_{p\|q} p'' \| q'$, $b \notin L$

(Regel 2, $b \notin L$) $\Rightarrow$ $(p' \| q')\langle L\rangle \xrightarrow{\tau}_1 (p'' \| q')\langle L\rangle$

Fall 2:

(Regel 2) $\Rightarrow$ $p'\langle L'\rangle \| q' \xrightarrow{\tau}_{p\langle L'\rangle \| q} p''\langle L'\rangle \| q'$

(Regel 3) $\Rightarrow$ $p'\langle L'\rangle \xrightarrow{\tau}_{p\langle L'\rangle} p''\langle L'\rangle$

Fall 2.1:

(Regel 1) $\Rightarrow$ $p' \xrightarrow{\tau}_p p''$

(Regel 3) $\Rightarrow$ $p' \| q' \xrightarrow{\tau}_{p\|q} p'' \| q'$

(Regel 2) $\Rightarrow$ $(p' \| q')\langle L\rangle \xrightarrow{\tau}_1 (p'' \| q')\langle L\rangle$

Fall 2.2:

(Regel 2) $\Rightarrow$ $p' \xrightarrow{b}_p p''$, $b \notin L'$, $b \in \mathcal{A}_p$

(Regel 3, $b \notin \mathcal{A}_q$)[3] $\Rightarrow$ $p' \| q' \xrightarrow{b}_{p\|q} p'' \| q'$, $b \notin L$

(Regel 2, $b \notin L$) $\Rightarrow$ $(p' \| q')\langle L\rangle \xrightarrow{\tau}_1 (p'' \| q')\langle L\rangle$

[3] Beachte: $b \notin L' \supseteq (\mathcal{A}_p \cap \mathcal{A}_q) \Rightarrow b \notin \mathcal{A}_q$ (da $b \in \mathcal{A}_p$).

Regel 4:

„ $\Rightarrow$ “:

$$(p'\|q')\langle L\rangle \xrightarrow{\tau}_1 (p'\|q'')\langle L\rangle$$

$$\begin{array}{lcl}
(\text{Regel } 2) & \Rightarrow & p'\|q' \xrightarrow{b}_{p\|q} p'\|q'' \quad , (b \notin L\,,\, b \in \mathcal{A}_q \setminus \mathcal{A}_p) \vee (b = \tau) \\
(\text{Regel } 4,\, b \notin \mathcal{A}_p) & \Rightarrow & q' \xrightarrow{b}_q q''\,,\, (b \notin L\,,\, b \in \mathcal{A}_q \setminus \mathcal{A}_p) \vee (b = \tau) \\
(\text{Regel } 4,\, b \notin \mathcal{A}_{p\langle L'\rangle}) & \Rightarrow & p'\langle L'\rangle\|q' \xrightarrow{b}_{p\langle L'\rangle\|q} p'\langle L'\rangle\|q'' \quad , (b \notin L) \vee (b = \tau) \\
(\text{Regel } 2,\, b \notin L) & \Rightarrow & (p'\langle L'\rangle\|q')\langle L\rangle \xrightarrow{\tau}_2 (p'\langle L'\rangle\|q'')\langle L\rangle
\end{array}$$

„ $\Leftarrow$ “:

$$(p'\langle L'\rangle\|q')\langle L\rangle \xrightarrow{\tau}_2 (p'\langle L'\rangle\|q'')\langle L\rangle$$

$$\begin{array}{lcl}
(\text{Regel } 2) & \Rightarrow & p'\langle L'\rangle\|q' \xrightarrow{b}_{p\langle L'\rangle\|q} p'\langle L'\rangle\|q'' \quad , (b \notin L\,,\, b \in \mathcal{A}_q \setminus \mathcal{A}_{p\langle L'\rangle}) \vee \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (b = \tau) \\
(\text{Regel } 4,\, b \notin \mathcal{A}_{p\langle L'\rangle}) & \Rightarrow & q' \xrightarrow{b}_q q''\,,\, (b \notin L\,,\, b \in \mathcal{A}_q \setminus \mathcal{A}_{p\langle L'\rangle}) \vee (b = \tau) \\
(\text{Regel } 4,\, b \notin \mathcal{A}_p)^4 & \Rightarrow & p'\|q' \xrightarrow{b}_{p\|q} p'\|q'' \quad , (b \notin L) \vee (b = \tau) \\
(\text{Regel } 2) & \Rightarrow & (p'\|q')\langle L\rangle \xrightarrow{\tau}_1 (p'\|q'')\langle L\rangle
\end{array}$$

Diese Fallunterscheidung beweist Punkt (1) des Induktionsschrittes. Punkt (2) ist eine unmittelbare Konsequenz aus Punkt (1). Deshalb ist nur noch Punkt (3) des Induktionsschrittes mit Hilfe von Definition 1.8 zu verifizieren:

[4] Beachte: $b \in \mathcal{A}_q \setminus \mathcal{A}_{p\langle L'\rangle} \Rightarrow b \notin \mathcal{A}_{p\langle L'\rangle} = \mathcal{A}_p \cap L'$. Wäre $b \in \mathcal{A}_p$, so auch $b \in \mathcal{A}_p \cap \mathcal{A}_q \subseteq L'$ und $b \in \mathcal{A}_{p\langle L'\rangle}$ (Widerspruch).

1. Fall: $a \neq \tau$

Regel 8:

$$(p' \| q')\langle L\rangle\uparrow_1 a \quad , a \in \mathcal{A}_p \setminus \mathcal{A}_q$$

$$(\text{Regel 6}, a \neq \tau) \iff (p' \| q')\uparrow_{p\|q} a \quad , a \in L\,, a \in \mathcal{A}_p \setminus \mathcal{A}_q$$

$$(\text{Regel 8}, a \notin \mathcal{A}_q) \iff p'\uparrow_p a \quad , a \in L\,, a \in \mathcal{A}_p \setminus \mathcal{A}_q$$

$$(\text{Regel 6}, a \in L \subseteq L') \iff p'\langle L'\rangle\uparrow_{p\langle L'\rangle} a \quad , a \in L\,, a \in \mathcal{A}_{p\langle L'\rangle} \setminus \mathcal{A}_q$$

$$(\text{Regel 8}, a \notin \mathcal{A}_q) \iff (p'\langle L'\rangle \| q')\uparrow_{p\langle L'\rangle\|q} a \quad , a \in L\,, a \in \mathcal{A}_{p\langle L'\rangle} \setminus \mathcal{A}_q$$

$$(\text{Regel 6}, a \neq \tau) \iff (p'\langle L'\rangle \| q')\langle L\rangle\uparrow_2 a \quad , a \in \mathcal{A}_{p\langle L'\rangle} \setminus \mathcal{A}_q$$

Regel 9:

$$(p' \| q')\langle L\rangle\uparrow_1 a \quad , a \in \mathcal{A}_p\,, q' \xrightarrow{a}_q$$

$$(\text{Regel 6}, a \neq \tau) \iff (p' \| q')\uparrow_{p\|q} a \quad , a \in L\,, a \in \mathcal{A}_p\,, q' \xrightarrow{a}_q$$

$$(\text{Regel 9}, q' \xrightarrow{a}_q) \iff p'\uparrow_p a \quad , a \in L\,, a \in \mathcal{A}_p\,, q' \xrightarrow{a}_q$$

$$(\text{Regel 6}, a \in L \subseteq L') \iff p'\langle L'\rangle\uparrow_{p\langle L'\rangle} a \quad , a \in L\,, a \in \mathcal{A}_{p\langle L'\rangle}\,, q' \xrightarrow{a}_q$$

$$(\text{Regel 9}, q' \xrightarrow{a}_q) \iff (p'\langle L'\rangle \| q')\uparrow_{p\langle L'\rangle\|q} a \quad , a \in L\,, a \in \mathcal{A}_{p\langle L'\rangle}\,, q' \xrightarrow{a}_q$$

$$(\text{Regel 6}, a \neq \tau) \iff (p'\langle L'\rangle \| q')\langle L\rangle\uparrow_2 a \quad , a \in \mathcal{A}_{p\langle L'\rangle}\,, q' \xrightarrow{a}_q$$

Regel 10:

$$(p' \| q')\langle L\rangle\uparrow_1 a \quad , a \in \mathcal{A}_q \setminus \mathcal{A}_p$$

$$(\text{Regel 6}, a \neq \tau) \iff (p' \| q')\uparrow_{p\|q} a \quad , a \in L\,, a \in \mathcal{A}_q \setminus \mathcal{A}_p$$

$$(\text{Regel 10}, a \notin \mathcal{A}_p) \iff q'\uparrow_q a \quad , a \in L\,, a \in \mathcal{A}_q \setminus \mathcal{A}_p$$

$$(\text{Regel 10}, a \notin \mathcal{A}_{p\langle L'\rangle})^5 \iff (p'\langle L'\rangle \| q')\uparrow_{p\langle L'\rangle\|q} a \quad , a \in L\,, a \in \mathcal{A}_q \setminus \mathcal{A}_{p\langle L'\rangle}$$

$$(\text{Regel 6}, a \neq \tau) \iff (p'\langle L'\rangle \| q')\langle L\rangle\uparrow_2 a \quad , a \in \mathcal{A}_q \setminus \mathcal{A}_{p\langle L'\rangle}$$

Regel 11:

$$(p'\|q')\langle L\rangle\uparrow_1 a \quad , a \in \mathcal{A}_q \, , \, p' \xrightarrow{a}_p$$

$$\begin{array}{lcl}
(\text{Regel } 6, a \neq \tau) & \Longleftrightarrow & (p'\|q')\uparrow_{p\|q} a \quad , a \in L \, , \, a \in \mathcal{A}_q \, , \, p' \xrightarrow{a}_p \\
(\text{Regel } 11, p' \xrightarrow{a}_p) & \Longleftrightarrow & q'\uparrow_q a \quad , a \in L \, , \, a \in \mathcal{A}_q \, , \, p' \xrightarrow{a}_p \\
(\text{Regel } 11, p'\langle L'\rangle \xrightarrow{a}_{p\langle L'\rangle})^6 & \Longleftrightarrow & (p'\langle L'\rangle\|q')\uparrow_{p\langle L'\rangle\|q} a \quad , a \in L \, , \, a \in \mathcal{A}_q, \\
 & & \qquad p'\langle L'\rangle \xrightarrow{a}_{p\langle L'\rangle} \\
(\text{Regel } 6, a \neq \tau) & \Longleftrightarrow & (p'\langle L'\rangle\|q')\langle L\rangle\uparrow_2 a \quad , a \in \mathcal{A}_q \, , \, p'\langle L'\rangle \xrightarrow{a}_{p\langle L'\rangle}
\end{array}$$

Regel 12:

$$(p'\|q')\langle L\rangle\uparrow_1 a \quad , a \in \mathcal{A}_p \, , \, a \in \mathcal{A}_q$$

$$\begin{array}{lcl}
(\text{Regel } 6, a \neq \tau) & \Longleftrightarrow & (p'\|q')\uparrow_{p\|q} a \quad , a \in L \, , \, a \in \mathcal{A}_p \, , \, a \in \mathcal{A}_q \\
(\text{Regel } 12) & \Longleftrightarrow & p'\uparrow_p a \, , \, q'\uparrow_q a \quad , a \in L \, , \, a \in \mathcal{A}_p \, , \, a \in \mathcal{A}_q \\
(\text{Regel } 6, a \in L \subseteq L') & \Longleftrightarrow & p'\langle L'\rangle\uparrow_{p\langle L'\rangle} a \, , \, q'\uparrow_q a \quad , a \in L \, , \, a \in \mathcal{A}_{p\langle L'\rangle} \, , \, a \in \mathcal{A}_q \\
(\text{Regel } 12) & \Longleftrightarrow & (p'\langle L'\rangle\|q')\uparrow_{p\langle L'\rangle\|q} a \quad , a \in L \, , \, a \in \mathcal{A}_{p\langle L'\rangle} \, , \, a \in \mathcal{A}_q \\
(\text{Regel } 6, a \neq \tau) & \Longleftrightarrow & (p'\langle L'\rangle\|q')\langle L\rangle\uparrow_2 a \quad , a \in \mathcal{A}_{p\langle L'\rangle} \, , \, a \in \mathcal{A}_q
\end{array}$$

<u>2. Fall: $a = \tau$</u>

<u>„ $\Rightarrow$ “:</u> Betrachte folgende vollständige Fallunterscheidung gemäß den Regeln zur operationellen Semantik (vgl. Definition 1.8):

$(p'\|q')\langle L\rangle\uparrow_1 \tau$

Fall 1:

$$(\text{Regel } 7) \quad \Rightarrow \quad (p'\|q')\uparrow_{p\|q} \tau$$

[5] Beachte: $a \notin \mathcal{A}_p \iff$ (wegen $a \in L \subseteq L'$) $a \notin \mathcal{A}_p \cap L' = \mathcal{A}_{p\langle L'\rangle}$.

[6] Vgl. vorige Fußnote und Regel 1.

Fall 1.1:

(Regel 8) $\Rightarrow \quad p'\uparrow_p\tau$

(Regel 7) $\Rightarrow \quad p'\langle L'\rangle\uparrow_{p\langle L'\rangle}\tau$

(Regel 8) $\Rightarrow \quad (p'\langle L'\rangle\|q')\uparrow_{p\langle L'\rangle\|q}\tau$

(Regel 7) $\Rightarrow \quad (p'\langle L'\rangle\|q')\langle L\rangle\uparrow_2\tau$

Fall 1.2:

(Regel 10) $\Rightarrow \quad q'\uparrow_q\tau$

(Regel 10) $\Rightarrow \quad (p'\langle L'\rangle\|q')\uparrow_{p\langle L'\rangle\|q}\tau$

(Regel 7) $\Rightarrow \quad (p'\langle L'\rangle\|q')\langle L\rangle\uparrow_2\tau$

Fall 1.3:

(Regel 12) $\Rightarrow \quad p'\uparrow_p\tau\,,\ q'\uparrow_q\tau$

(Regel 7) $\Rightarrow \quad p'\langle L'\rangle\uparrow_{p\langle L'\rangle}\tau\,,\ q'\uparrow_q\tau$

(Regel 12) $\Rightarrow \quad (p'\langle L'\rangle\|q')\uparrow_{p\langle L'\rangle\|q}\tau$

(Regel 7) $\Rightarrow \quad (p'\langle L'\rangle\|q')\langle L\rangle\uparrow_2\tau$

Fall 2:

(Regel 7) $\Rightarrow \quad (p'\|q')\uparrow_{p\|q}b \quad ,\ b\notin L\,,\ b\neq\tau$

Fall 2.1:

(Regel 8) $\Rightarrow \quad p'\uparrow_p b \quad ,\ b\notin L\,,\ b\in\mathcal{A}_p\setminus\mathcal{A}_q$

Fall 2.1.1: $b\notin L'$

(Regel 7) $\Rightarrow \quad p'\langle L'\rangle\uparrow_{p\langle L'\rangle}\tau$

(Regel 8) $\Rightarrow \quad (p'\langle L'\rangle\|q')\uparrow_{p\langle L'\rangle\|q}\tau$

$(\text{Regel } 7) \quad \Rightarrow \quad (p'\langle L'\rangle \| q')\langle L\rangle\uparrow_2\tau$

Fall 2.1.2: $b \in L'$

$(\text{Regel } 6) \quad \Rightarrow \quad p'\langle L'\rangle\uparrow_{p\langle L'\rangle} b \quad , b \notin L \, , b \notin \mathcal{A}_q$

$(\text{Regel } 8 , b \notin \mathcal{A}_q) \quad \Rightarrow \quad (p'\langle L'\rangle \| q')\uparrow_{p\langle L'\rangle \| q} b \quad , b \notin L$

$(\text{Regel } 7 , b \notin L) \quad \Rightarrow \quad (p'\langle L'\rangle \| q')\langle L\rangle\uparrow_2\tau$

Fall 2.2:

$(\text{Regel } 9) \quad \Rightarrow \quad p'\uparrow_p b \quad , b \notin L \, , b \in \mathcal{A}_p \, , q' \xrightarrow{b}_q$

$(\text{Regel } 6 , b \in L') \quad \Rightarrow \quad p'\langle L'\rangle\uparrow_{p\langle L'\rangle} b \quad , b \notin L \, , b \in \mathcal{A}_p \, , q' \xrightarrow{b}_q$

$(\text{Regel } 9) \quad \Rightarrow \quad (p'\langle L'\rangle \| q')\uparrow_{p\langle L'\rangle \| q} b \quad , b \notin L$

$(\text{Regel } 7 , b \notin L) \quad \Rightarrow \quad (p'\langle L'\rangle \| q')\langle L\rangle\uparrow_2\tau$

Fall 2.3:

$(\text{Regel } 10) \quad \Rightarrow \quad q'\uparrow_q b \quad , b \notin L \, , b \in \mathcal{A}_q \setminus \mathcal{A}_p$

$(\text{Regel } 10)^7 \quad \Rightarrow \quad (p'\langle L'\rangle \| q')\uparrow_{p\langle L'\rangle \| q} b \quad , b \notin L$

$(\text{Regel } 7 , b \notin L) \quad \Rightarrow \quad (p'\langle L'\rangle \| q')\langle L\rangle\uparrow_2\tau$

Fall 2.4:

$(\text{Regel } 11) \quad \Rightarrow \quad q'\uparrow_q b \quad , b \notin L \, , b \in \mathcal{A}_q \, , p' \xrightarrow{b}_p$

$(\text{Regel } 11)^8 \quad \Rightarrow \quad (p'\langle L'\rangle \| q')\uparrow_{p\langle L'\rangle \| q} b \quad , b \notin L$

$(\text{Regel } 7 , b \notin L) \quad \Rightarrow \quad (p'\langle L'\rangle \| q')\langle L\rangle\uparrow_2\tau$

Fall 2.5:

$(\text{Regel } 12) \quad \Rightarrow \quad p'\uparrow_p b \, , q'\uparrow_q b \quad , b \in \mathcal{A}_p \, , b \in \mathcal{A}_q \, , b \notin L$

$(\text{Regel } 6 , b \in L') \quad \Rightarrow \quad p'\langle L'\rangle\uparrow_{p\langle L'\rangle} b \quad , q'\uparrow_q b \, , b \notin L$

$$\text{(Regel 12)} \quad \Rightarrow \quad (p'\langle L'\rangle \| q')\uparrow_{p\langle L'\rangle \| q} b \quad , b \notin L$$

$$\text{(Regel 7, } b \notin L) \quad \Rightarrow \quad (p'\langle L'\rangle \| q')\langle L\rangle\uparrow_2 \tau$$

„ $\Leftarrow$ “: Betrachte folgende vollständige Fallunterscheidung gemäß den Regeln zur operationellen Semantik (vgl. Definition 1.8):

$(p'\langle L'\rangle \| q')\langle L\rangle\uparrow_2 \tau$

Fall 1:

$$\text{(Regel 7)} \quad \Rightarrow \quad (p'\langle L'\rangle \| q')\uparrow_{p\langle L'\rangle \| q} \tau$$

Fall 1.1:

$$\text{(Regel 8)} \quad \Rightarrow \quad p'\langle L'\rangle\uparrow_{p\langle L'\rangle} \tau$$

Fall 1.1.1:

$$\text{(Regel 7)} \quad \Rightarrow \quad p'\uparrow_p \tau$$

$$\text{(Regel 8)} \quad \Rightarrow \quad (p' \| q')\uparrow_{p \| q} \tau$$

$$\text{(Regel 7)} \quad \Rightarrow \quad (p' \| q')\langle L\rangle\uparrow_1 \tau$$

Fall 1.1.2:

$$\text{(Regel 7)} \quad \Rightarrow \quad p'\uparrow_p b \quad , b \notin L'$$

$$\text{(Regel 8, } b \notin \mathcal{A}_q)^9 \quad \Rightarrow \quad (p' \| q')\uparrow_{p \| q} b \quad , b \notin L$$

$$\text{(Regel 7, } b \notin L) \quad \Rightarrow \quad (p' \| q')\langle L\rangle\uparrow_1 \tau$$

[7] Beachte: $b \notin \mathcal{A}_p \Rightarrow b \notin \mathcal{A}_{p\langle L'\rangle}$.

[8] Beachte: $b \in \mathcal{A}_p \cap \mathcal{A}_q \subseteq L'$ impliziert $p'\langle L'\rangle \xrightarrow{b}_{p\langle L'\rangle}$.

Fall 1.2:

$$(\text{Regel } 10) \quad \Rightarrow \quad q'\uparrow_q \tau$$

$$(\text{Regel } 10) \quad \Rightarrow \quad (p' \| q')\uparrow_{p\|q} \tau$$

$$(\text{Regel } 7) \quad \Rightarrow \quad (p' \| q')\langle L\rangle\uparrow_1 \tau$$

Fall 1.3:

$$(\text{Regel } 12) \quad \Rightarrow \quad p'\langle L'\rangle\uparrow_{p\langle L'\rangle} \tau \,,\, q'\uparrow_q \tau$$

Fall 1.3.1:

$$(\text{Regel } 7) \quad \Rightarrow \quad p'\uparrow_p b, q'\uparrow_q \tau \quad , b \notin L'$$

$$(\text{Regel } 10) \quad \Rightarrow \quad (p' \| q')\uparrow_{p\|q} \tau$$

$$(\text{Regel } 7) \quad \Rightarrow \quad (p' \| q')\langle L\rangle\uparrow_1 \tau$$

Fall 1.3.2:

$$(\text{Regel } 7) \quad \Rightarrow \quad p'\uparrow_p \tau, q'\uparrow_q \tau$$

$$(\text{Regel } 12) \quad \Rightarrow \quad (p' \| q')\uparrow_{p\|q} \tau$$

$$(\text{Regel } 7) \quad \Rightarrow \quad (p' \| q')\langle L\rangle\uparrow_1 \tau$$

Fall 2:

$$(\text{Regel } 7) \quad \Rightarrow \quad (p'\langle L'\rangle \| q')\uparrow_{p\langle L'\rangle\|q} b \quad , b \notin L \,,\, b \neq \tau$$

Fall 2.1:

$$(\text{Regel } 8) \quad \Rightarrow \quad p'\langle L'\rangle\uparrow_{p\langle L'\rangle} b \quad , b \notin L \,,\, b \in \mathcal{A}_{p\langle L'\rangle} \,,\, b \notin \mathcal{A}_q$$

$$(\text{Regel } 6,\, b \in \mathcal{A}_{p\langle L'\rangle} \subseteq L') \quad \Rightarrow \quad p'\uparrow_p b \quad , b \notin L \,,\, b \in \mathcal{A}_p \,,\, b \notin \mathcal{A}_q$$

$$(\text{Regel } 8,\, b \notin \mathcal{A}_q) \quad \Rightarrow \quad (p' \| q')\uparrow_{p\|q} b \quad , b \notin L$$

$$(\text{Regel } 7,\, b \notin L) \quad \Rightarrow \quad (p' \| q')\langle L\rangle\uparrow_1 \tau$$

Fall 2.2:

(Regel 9)	$\Rightarrow$	$p'\langle L'\rangle\uparrow_{p\langle L'\rangle} b \quad , b \notin L , b \in \mathcal{A}_{p\langle L'\rangle} , q' \xrightarrow{b}_q$
(Regel 6, $b \in \mathcal{A}_{p\langle L'\rangle} \subseteq L'$)	$\Rightarrow$	$p'\uparrow_p b \quad , b \notin L , b \in \mathcal{A}_p , q' \xrightarrow{b}_q$
(Regel 9, $q' \xrightarrow{b}_q$)	$\Rightarrow$	$(p' \| q')\uparrow_{p\|q} b \quad , b \notin L$
(Regel 7, $b \notin L$)	$\Rightarrow$	$(p' \| q')\langle L\rangle\uparrow_1 \tau$

Fall 2.3:

(Regel 10)	$\Rightarrow$	$q'\uparrow_q b \quad , b \notin L , b \notin \mathcal{A}_{p\langle L'\rangle} , b \in \mathcal{A}_q$
(Regel 10, $b \notin \mathcal{A}_p$)[10]	$\Rightarrow$	$(p' \| q')\uparrow_{p\|q} b \quad , b \notin L$
(Regel 7, $b \notin L$)	$\Rightarrow$	$(p' \| q')\langle L\rangle\uparrow_1 \tau$

Fall 2.4:

(Regel 11)	$\Rightarrow$	$q'\uparrow_q b \quad , b \notin L , b \neq \tau , b \in \mathcal{A}_q , p'\langle L'\rangle \xrightarrow{b}_{p\langle L'\rangle}$
(Regel 1, $b \neq \tau$)	$\Rightarrow$	$q'\uparrow_q b \quad , b \notin L , p' \xrightarrow{b}_p , b \in \mathcal{A}_p , b \in \mathcal{A}_q$
(Regel 11)	$\Rightarrow$	$(p' \| q')\uparrow_{p\|q} b \quad , b \notin L$
(Regel 6, $b \notin L$)	$\Rightarrow$	$(p' \| q')\langle L\rangle\uparrow_1 \tau$

Fall 2.5:

(Regel 12)	$\Rightarrow$	$p'\langle L'\rangle\uparrow_{p\langle L'\rangle} b \quad , q'\uparrow_q b , b \notin L , b \neq \tau$
(Regel 6, $b \in L'$)	$\Rightarrow$	$p'\uparrow_p b , q'\uparrow_q b \quad , b \notin L$
(Regel 12)	$\Rightarrow$	$(p' \| q')\uparrow_{p\|q} b \quad , b \notin L$
(Regel 7, $b \notin L$)	$\Rightarrow$	$(p' \| q')\langle L\rangle\uparrow_1 \tau$

[9] Wäre $b \in \mathcal{A}_q$, so würde (beachte $b \in \mathcal{A}_p$) $b \in \mathcal{A}_p \cap \mathcal{A}_q$ und schließlich $b \in L'$ folgen (Widerspruch).

[10] Wäre $b \in \mathcal{A}_p$, so würde (beachte $b \in \mathcal{A}_q$) $b \in \mathcal{A}_p \cap \mathcal{A}_q \subseteq L'$ folgen und daher $b \in \mathcal{A}_{p\langle L'\rangle}$ gelten (Widerspruch).

Per Induktionsprinzip folgt die Behauptung.

Anhang C

Datenflußanalyse

C.1 Grundlagen

Die *Datenflußanalyse* ist eine spezielle Form der *statischen Analyse* von Programmen. Dabei interessiert man sich i.allg. für Informationen, die an bestimmten Stellen eines Programms (während der Programmausführung) gültig sind. Gegenstand der Datenflußanalyse ist, wie man diese Information zur Übersetzungszeit erhalten kann.
Datenflußanalyse, die auf *abstrakter Interpretation* beruht, ist theoretisch am weitesten erforscht. Ziel der abstrakten Interpretation ist es, die komplexe Semantik eines Programms durch eine einfachere, auf die gewünschten Informationen zugeschnittene Semantik zu ersetzen. Dies geschieht gewöhnlich durch die Definition eines *lokalen abstrakten Semantikfunktionals*, welches jeder Aktion im Programm eine entsprechende Transformation zuordnet. Als formales Hilfsmittel betrachtet man eine *mengenvollständige Halbordnung* $\langle C; \sqsubseteq \rangle$, deren Elemente die Datenflußinformationen repräsentieren, mit deren Hilfe man Aussagen an Programmstellen treffen möchte. Um die interessierenden Aussagen zu erhalten, muß man ausgehend von der lokalen abstrakten Semantik eine *globale abstrakte Semantik* definieren. Eine erste Möglichkeit dazu bietet die operationelle *„join over all paths" (JOP)–Strategie*, welche direkt der Intuition entspricht: Sie bestimmt die Information, die durch das Verfolgen der lokalen abstrakten Semantikfunktionen (ausgehend vom Programmanfang mit einer initialen Datenflußinformation, entlang aller möglichen Programmpfade bis zur beobachteten Stelle im Programm) gegeben ist. Diese Strategie ist jedoch ineffektiv, da es unendlich viele Pfade zu einer bestimmten Stelle im Programm geben kann. Die zweite Möglichkeit bietet die denotationelle *„minimal fixed point" (MFP)–Strategie*. Es wird die kleinste Lösung eines Gleichungssystems betrachtet, welches die Konsistenz zwischen Vor– und Nachbedingungen von Datenflußinformationen beschreibt. Dabei muß die Vorbedingung einer Programmaktion von den Nachbedingungen ihrer Vorgängeraktionen impliziert werden. Die Nachbedingung einer Programmaktion ergibt sich durch die Anwendung ihrer lokalen abstrakten Semantikfunktion auf ihre Vorbedingung. Dieser Ansatz führt unter gewissen Voraussetzungen zu einer effektiven (manchmal auch zur effizienten) algorithmischen Beschreibung der globalen abstrakten Semantik und damit zur effektiven Berechnung der gewünschten Informationen an einer beliebigen Stelle des Programms.

Kam und Ullman [KU77] haben gezeigt, daß die beiden Strategien unter der Voraussetzung *additiver* lokaler abstrakter Semantikfunktionen das gleiche Ergebnis liefern, d.h., daß die sogenannte *JOP–Lösung* und die *MFP–Lösung* übereinstimmen. Diese Aussage ist unter dem Begriff *Koinzidenztheorem* bekannt.

Prozesse und endliche Pfade

Zur statischen Analyse werden Programme hier als Prozesse dargestellt.[1] Wie in Kapitel 1, Definition 1.2 definiert, ist ein Prozeß ein Paar bestehend aus einem Transitionssystem und einem ausgezeichneten Startzustand.[2] Bei dieser Darstellung wird im Gegensatz zu Struktogrammen von den für die Analyse unwichtigen Details eines Programms abstrahiert. Entscheidend sind bei der statischen Analyse nur die einzelnen Aktionen eines Programms und deren Zusammenspiel, d.h. die Verzweigungsstruktur. Um entscheidbar zu bleiben, wird die Verzweigungsstruktur nichtdeterministisch betrachtet.

Im folgenden sei $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p)$ ein fester, aber beliebiger Prozeß. Für eine formale Behandlung der Datenflußanalyse werden die folgenden Definitionen benötigt:

Definition C.1 (Vorgänger und Nachfolger)
Für $k = (q, \cdot, q') \in \longrightarrow_p$ bezeichnet $pred(k) =_{df} \{ l \mid l = (\cdot, \cdot, q) \in \longrightarrow_p \}$ die Menge aller Vorgänger und $succ(k) =_{df} \{ l \mid l = (q', \cdot, \cdot) \in \longrightarrow_p \}$ die Menge aller Nachfolger der Transition k.

Definition C.2 (Endliche Pfade)
Ein endlicher Pfad pf in einem (erweiterten) Transitionssystem $(S, \mathcal{A} \cup \{\tau\}, \longrightarrow, \uparrow)$ ist eine Folge $pf =_{df} (k \equiv k_1, ..., k_q \equiv l)$, $q \geq 0$, $k_i \in \longrightarrow$ $(1 \leq i \leq q)$, von Transitionen mit $k_j \in pred(k_{j+1})$ für $1 \leq j \leq q-1$. Die Länge $lgth(pf)$ des Pfades pf ist q, wobei ϵ der eindeutig bestimmte Pfad der Länge 0 ist. Die Menge aller endlichen Pfade von k nach l wird mit $P[k, l]$ bzw. jene von k zu einem Vorgänger von l mit $P[k, l)$ bezeichnet.

Aus beweistechnischen Gründen nehmen wir von dem Prozeß p o.B.d.A. an, daß zu seinem Startzustand keine Transition führt und daß von seinem Startzustand nur eine τ–Transition, bezeichnet mit k_{start}, ausgeht. Ist diese Bedingung von einem Prozeß $q = ((S_q, \mathcal{A}_q \cup \{\tau\}, \longrightarrow_q, \uparrow_q), q)$ nicht erfüllt, so betrachte den dazu $\approx^d$–äquivalenten Prozeß $q' = ((S_{q'}, \mathcal{A}_q \cup \{\tau\}, \longrightarrow_{q'}, \uparrow_q), q')$ mit $S_{q'} =_{df} S_q \cup \{q'\}$, $q' \notin S_q$, und $\longrightarrow_{q'} =_{df} \longrightarrow_q \cup \{(q', \tau, q)\}$.

Lokale abstrakte Semantik

Jede Transition eines Transitionssystems repräsentiert, wie oben erwähnt, eine Aktion des betrachteten Programms. Die Semantik einer Transition läßt sich somit als die Transformationsfunktion definieren, welche der Transition entspricht.

[1] Die Darstellung von Programmen durch Prozesse ist in der Datenflußanalyse unüblich, aber für die im Rahmen dieser Arbeit auftretenden Anwendung notwendig. Vielmehr werden in der Datenflußanalyse Programme durch Flußgraphen modelliert.

[2] Das Undefiniertheitsprädikat eines Prozesses spielt in diesem Zusammenhang keine Rolle.

Formal benötigt man zunächst eine mengenvollständige Halbordnung $\langle C; \sqsubseteq \rangle$, deren Elemente die für die Analyse relevanten Datenflußinformationen darstellen.
Die lokale abstrakte Semantik eines Prozesses $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p)$ ist durch das Semantikfunktional $[\![\cdot]\!] : (\longrightarrow_p) \rightarrow (C \rightarrow C)$ definiert, welches jeder Transition $k \in \longrightarrow_p$ eine Transformation in C zuordnet.
$[\![\cdot]\!]$ läßt sich für endliche Pfade einfach erweitern. Definiere dazu für jeden endlichen Pfad $pf = (k_1, ..., k_q)$, $q \geq 0$:

$$[\![pf]\!] =_{df} [\![k_q]\!] \circ ... \circ [\![k_1]\!].$$

Das Semantikfunktional eines Pfades ist als die Hintereinanderausführung der Semantikfunktionen auf dem Pfad (in natürlicher Reihenfolge) definiert.

Gesucht ist nun, ausgehend vom lokalen abstrakten Semantikfunktional, eine Definition einer globalen abstrakten Semantik. Im folgenden werden zwei unterschiedliche Ansätze dazu vorgestellt: zum einen die operationelle JOP–Strategie (Vereinigung über alle Pfade) und zum anderen die denotationelle MFP–Strategie (minimaler Fixpunkt).

JOP–Strategie

Die JOP („join over all paths“)–Strategie entspricht direkt der Intuition. Ausgehend von der für die Starttransition k_{start} gültigen Information c_0 wird für jeden endlichen Pfad im betrachteten Transitionssystem von der Starttransition k_{start} bis vor die betrachtete Transition k diejenige Information berechnet, die vor der Ausführung der zu k gehörigen Aktion gilt. Vereinigt man diese für jeden Pfad erhaltenen Informationen, so erhält man die (Datenfluß–)Information $JOP_{c_0}(k)$, die bzgl. der Transition k gilt, und zwar unabhängig entlang welchen Pfades man k erreicht hat. Gibt es also einen Pfad, der bzgl. der Transition k die Information c liefert, so ist c in $JOP_{c_0}(k)$ enthalten ($c \sqsubseteq JOP_{c_0}(k)$). Diese Semantik wird üblicherweise als *Aufsammelsemantik* bezeichnet.
Formal sieht dies für die mengenvollständige Halbordnung $\langle C; \sqsubseteq \rangle$ wie folgt aus:

$$\forall\, k \in \longrightarrow_p \ \forall\, c_0 \in C.\ \ JOP_{c_0}(k) =_{df} \bigsqcup \{[\![pf]\!](c_0) \mid pf \in P[k_{start}, k)\}$$

Der Nachteil dieser Strategie ist ihre Ineffektivität: Graphentheoretisch gesprochen kann ein Transitionssystem einen Zykel enthalten, so daß es dann von der Starttransition k_{start} aus unendlich viele endliche Pfade zu der betrachteten Transition k gibt; je nachdem wie oft man diesen Zykel durchläuft, erhält man jedesmal einen anderen endlichen Pfad.

MFP-Strategie

Die MFP („minimal fixed point“)–Strategie entspricht nur indirekt der Intuition. Dabei geht man von einem Gleichungssystem aus, welches die Konsistenz zwischen Vor– und

Nachbedingungen von Datenflußinformationen an den Transitionen des betrachteten Transitionssystems beschreibt.
Die Vorbedingung einer Transition k, d.h. die (Datenfluß–)Information, die vor Ausführung der zu k gehörigen Aktion gültig ist, ist für die Starttransition k_{start} die initiale Information c_0 und für alle anderen Transitionen die Vereinigung der Nachbedingungen der Vorgängertransitionen von k (also die vor k gültige Information, egal von welchem Vorgänger aus man gekommen ist). Die Nachbedingung einer Transition k, d.h. die (Datenfluß–)Information, die nach Ausführung der zu k gehörigen Aktion gültig ist, ergibt sich durch Anwendung der lokalen Semantikfunktion $[\![k]\!]$ auf die Vorbedingung von k.
Formal betrachtet man ein Gleichungssystem, welches für $k \in \longrightarrow_p$ aus folgenden Gleichungen entsteht:

$$pre(k) = \begin{cases} c_0 & \text{falls} \quad k = k_{start} \\ \bigsqcup\{post(l) \mid l \in pred(k)\} & \text{sonst} \end{cases}$$

$$post(k) = [\![k]\!](pre(k))$$

Die kleinste Lösung $(pre_{c_0}, post_{c_0})$ des Gleichungssystems beinhaltet die gesuchte MFP-Lösung $MFP_{c_0}(k)$ bzgl. der initialen Information c_0 und der betrachteten Transition k:

$$\forall\, k \in \longrightarrow_p \ \forall\, c_0 \in C.\ \ MFP_{c_0}(k) =_{df} pre_{c_0}(k).$$

Der Vorteil dieser Strategie liegt in der effektiven Berechenbarkeit des kleinsten Fixpunktes des Gleichungssystems, falls die Semantikfunktionen monoton sind und die maximale Kettenlänge in der Halbordnung $\langle C; \sqsubseteq\rangle$ endlich ist.

Korrektheit und Optimalität

In den vorangegangenen beiden Abschnitten wurden zwei Strategien, nämlich die JOP– und die MFP–Strategie, vorgestellt. Beide spiegeln das Ziel wider, für jede Transition eines beliebigen, aber festen Prozesses ausgehend von der Startinformation c_0 die dort gültige Menge an (Datenfluß–)Informationen zu bestimmen.
Die JOP–Strategie benutzt aufgrund der Ausdehnung des Semantikfunktionals $[\![\cdot]\!]$ auf endliche Pfade eine operationelle Semantikdefinition, welche direkt die Intuition wiedergibt, aber i.allg. nicht zu effektiven Algorithmen führt.
Die MFP–Strategie führt aufgrund der Definition über den kleinsten Fixpunkt eines Gleichungssystems zu einer unter den genannten Voraussetzungen effektiven algorithmischen Beschreibung, trifft die Intuition aber nur indirekt.
Ideal wäre es nun, wenn JOP– und MFP–Lösung übereinstimmen würden, womit das Ausgangsproblem effektiv lösbar wäre. Somit stellt sich unmittelbar die Frage nach der *Korrektheit* (Sicherheit) und der *Vollständigkeit* (Optimalität) solcher Algorithmen zur Berechnung der MFP–Lösung im Vergleich zur JOP–Lösung. Zentral sind dabei die Begriffe *Monotonie* und *Additivität*.

Das Koinzidenztheorem

Sind die lokalen abstrakten Semantikfunktionen $[\![k]\!]$, $k \in \longrightarrow_p$, bzgl. $\langle C; \sqsubseteq \rangle$ additiv, so lassen sich die Korrektheit und die Optimalität allgemein zeigen.[3] Es gilt folgendes Resultat:

Theorem C.3 (Koinzidenztheorem)
Gegeben sei ein (erweitertes) Transitionssystem $(S, \mathcal{A} \cup \{\tau\}, \longrightarrow, \uparrow)$.
Sind alle Semantikfunktionen $[\![k]\!]$ *(für* $k \in \longrightarrow$*) additiv, so ist die MFP–Lösung korrekt und optimal in bezug auf die JOP–Lösung, d.h. es gilt:*

$$\forall\, k \in \longrightarrow\ \forall\, c_0 \in C.\ \ JOP_{c_0}(k) = MFP_{c_0}(k).$$

Beweis
Sei $(S, \mathcal{A} \cup \{\tau\}, \longrightarrow, \uparrow)$ ein beliebiges (erweitertes) Transitionssystem, $\langle C; \sqsubseteq \rangle$ eine beliebige mengenvollständige Halbordnung und $[\![l]\!] : C \longrightarrow C$ für $l \in \longrightarrow$ additiv sowie $c_0 \in C$ und $k \in \longrightarrow$ beliebig.

„$\sqsupseteq$“: Zu zeigen ist $JOP_{c_0}(k) \sqsupseteq MFP_{c_0}(k)$.

Beweis mittels vollständiger Induktion über die Länge i eines Pfades pf von k_{start} nach k.

Induktionsanfang: $(i = 0)$
Hier gilt $k = k_{start}$ und deswegen
$JOP_{c_0}(k_{start}) = \bigsqcup\{[\![pf]\!](c_0) \mid pf \in P[k_{start}, k_{start})\} = [\![\epsilon]\!](c_0) = c_0 = MFP_{c_0}(k_{start})$

Gelte also $JOP_{c_0}(l) \sqsupseteq MFP_{c_0}(l)$ für alle Kanten l, für die es einen Pfad pf' von k_{start} nach l der Länge i gibt.

Induktionsschluß: $(i \longrightarrow i + 1)$
Habe pf die Gestalt $pf' \cdot l$.[4] Dann gilt:

$$\begin{array}{lcl} & & JOP_{c_0}(k) \\ (\text{Definition von } JOP) & = & \bigsqcup\{[\![pf]\!](c_0) \mid pf \in P[k_{start}, k)\} \\ (\text{siehe Fußnote}^5) & \sqsupseteq & \bigsqcup\{[\![l]\!]([\![pf']\!](c_0)) \mid pf' \in P[k_{start}, l)\} \\ (\text{Additivität von } [\![l]\!]) & = & [\![l]\!](\bigsqcup\{[\![pf']\!](c_0) \mid pf' \in P[k_{start}, l)\}) \end{array}$$

[3] Die Korrektheit benötigt nur die Voraussetzung monotoner lokaler abstrakter Semantikfunktionen.

[4] Dabei bezeichnet $pf' \cdot l$ den Pfad, welcher durch das Anhängen der Transition l an den Pfad pf' entsteht; vorausgesetzt, daß der Pfad pf' mit einer Vorgängertransition von l endet.

$$
\begin{array}{lcl}
(\text{Definition von } JOP) & = & [\![l]\!](JOP_{c_0}(l)) \\
(\text{Induktionsannahme \& Monotonie}) & \sqsupseteq & [\![l]\!](MFP_{c_0}(l)) \\
(\text{Definition von } MFP) & \sqsupseteq & MFP_{c_0}(k)
\end{array}
$$

Per Induktionsprinzip folgt die Behauptung.

„$\sqsubseteq$“: Zu zeigen ist $MFP_{c_0}(k) \sqsupseteq JOP_{c_0}(k)$.

Nach Definition von $JOP_{c_0}(k)$ genügt es, folgende äquivalente Aussage zu zeigen:

$$(*) \qquad \forall pf \in P[k_{start}, k).\ MFP_{c_0}(k) \sqsupseteq [\![pf]\!](c_0)$$

Sei $pf \in P[k_{start}, k)$ beliebig. Der Beweis von $(*)$ erfolgt jetzt mittels vollständiger Induktion über die Länge i des Pfades pf.

Induktionsanfang: $(i = 0)$
Hier gilt $k = k_{start}$ sowie $pf = \epsilon$ und daher $MFP_{c_0}(k_{start}) = c_0 = [\![\epsilon]\!](c_0)$.

Gelte also $MFP_{c_0}(k) \sqsupseteq [\![pf']\!](c_0)$ für alle Pfade pf' mit $lgth(pf') = i$.

Induktionsschluß: $(i \longrightarrow i + 1)$
Sei etwa $pf = pf' \cdot l'$ mit $l' \in pred(k)$, d.h. $pf' \in P[k_{start}, l')$ und $lgth(pf') = i$. Dann gilt:

$$
\begin{array}{lcl}
MFP_{c_0}(k) & & \\
(\text{Definition von } MFP) & = & \bigsqcup\{post_{c_0}(l) \mid l \in pred(k)\} \\
(\text{Wähle speziell } l') & \sqsupseteq & post_{c_0}(l') \\
(\text{Definition von } MFP) & = & [\![l']\!](MFP_{c_0}(l')) \\
(\text{Induktionsannahme \& Monotonie}) & \sqsupseteq & [\![l']\!]([\![pf']\!](c_0)) \\
(\text{Definition von } pf) & = & [\![pf]\!](c_0)
\end{array}
$$

Per Induktionsprinzip folgt die Behauptung.

[5] Betrachte nicht mehr alle Pfade von k_{start} bis vor k, sondern nur solche, die vor der Vorgängerkante l von k enden.

Die Aussage des Koinzidenztheorems folgt nun wegen der Antisymmetrie von „$\sqsubseteq$“.

Aufgrund des Koinzidenztheorems erhält man die i.allg. nicht berechenbare JOP–Lösung über die (unter der zusätzlichen Voraussetzung einer endlichen maximalen Kettenlänge in der Halbordnung $\langle C; \sqsubseteq \rangle$) effektiv zu berechnende MFP–Lösung, falls alle Semantikfunktionen additiv sind.

Ein expliziter Algorithmus

Hier wird ein Algorithmus in PASCAL–ähnlicher Notation angegeben, welcher die MFP–Lösung, also den kleinsten Fixpunkt des MFP–Gleichungssystems, unter den oben erwähnten Voraussetzungen effektiv berechnet.

Eingabe des Algorithmus:

- der betrachtete Prozeß $p = ((S_p, \mathcal{A}_p \cup \{\tau\}, \longrightarrow_p, \uparrow_p), p)$,
- die additiven Semantikfunktionen $[\![k]\!]$ für $k \in \longrightarrow_p$ und
- die Startinformation $c_0 \in C$.

Ausgabe des Algorithmus:

Zu jeder Transition $k \in \longrightarrow_p$ ist nach der Terminierung des Algorithmus die Vorbedingung (d.h. die vor Ausführung der zu k gehörigen Aktion gültige Datenflußinformation) in der Variablen $pre[k]$ und die Nachbedingung (d.h. die nach Ausführung der zu k gehörigen Aktion gültige Datenflußinformation) in der Variablen $post[k]$ gespeichert.

Der Algorithmus beruht auf dem Worksetprinzip, d.h. in einer Variablen *workset* werden diejenigen Transitionen gespeichert, für die sich aufgrund einer Änderung der Datenflußinformation einer Vorgängertransition ebenfalls noch etwas ändern könnte und die deshalb zumindest noch einmal überprüft werden müssen. Die Variable *workset* ist mit der Starttransition k_{start} initialisiert. Diese ist mit der initialen Datenflußinformation c_0 beschriftet. Die Variablen $pre[k]$ und $post[k]$ korrespondieren mit den obigen Bezeichnungen aus dem MFP–Gleichungssystem und sind mit dem $\perp$–Element von C, also dem kleinsten Element der betrachteten mengenvollständigen Halbordnung, initialisiert. Eine Ausnahme bildet wie bereits erwähnt $pre[k_{start}]$, das mit der initialen Datenflußinformation c_0 initialisiert ist. Ansonsten ist der Algorithmus eine direkte Umsetzung des MFP–Gleichungssystems.

(Initialisierung der Felder *pre* und *post* sowie der Variablen *workset*)
FORALL $k \in \longrightarrow$ **DO** $(pre[k], post[k]) := (\perp, \perp)$ **OD**;
$pre[k_{start}] := c_0$;
$workset := \{k_{start}\}$;

```
(Iterative Fixpunktberechnung)
WHILE workset ≠ ∅ DO
   LET k ∈ workset
      BEGIN
         workset := workset \ { k };
         post[k] := [[k]](pre[k]);
         (Aktualisierung der Umgebung von k und der Variablen workset)
         FORALL l ∈ succ(k) DO
            union := pre[l] ⊔ post[k];
            IF union    pre[l]
               THEN
                  pre[l] := union;
                  workset := workset ∪ { l } FI
         OD
      END
OD.
```

C.2 Anwendung

Bei der kompositionellen Minimierung endlicher verteilter Systeme ist die Datenflußanalyse Hilfsmittel für die Berechnung eines reduzierten Prozesses, also für die algorithmische Realisierung des speziellen Reduktionsoperators $\overline{\Pi}$ (vgl. Abschnitt 2.3).
Um die vorgestellte Theorie der Datenflußanalyse auf diesen speziellen Fall anwenden zu können, sind zunächst eine mengenvollständige Halbordnung und additive lokale abstrakte Semantikfunktionen zu definieren sowie die zugehörige JOP– und MFP–Lösung zu charakterisieren.

Sei p ein beliebiger, aber fester Prozeß und $I \in \mathcal{I}(p)$ eine beliebige, aber feste Interfacespezifikation für p.
Wie in Abschnitt 2.3 bereits angedeutet, wird hier die mengenvollständige Halbordnung $\langle \underline{\text{Languages}}; \subseteq \rangle$ für $\underline{\text{Languages}} = \mathcal{P}(\mathcal{A}_I^*)$ (vgl. Lemma A.6) betrachtet. Ebenso sind dort bereits die interessierenden lokalen abstrakten Semantikfunktionen $\mathcal{E}_a^I$ definiert, welche nach Lemma 2.15 additiv sind. Beachte dabei folgendes:
Damit die vorgestellte Technik zur Datenflußanalyse, welche auf der Betrachtung der Transitionen von p basiert, mit der in Abschnitt 2.3 praktizierten Betrachtung der Zustände von p korrespondiert, wird eine Transition $k = (q, a, q') \in \longrightarrow_p$ mit dem Zustand q identifiziert. Das lokale abstrakte Semantikfunktional $[\![\cdot]\!]$ ist somit formal wie folgt definiert:

$$[\![k]\!] =_{df} \mathcal{E}_a^I.$$

Charakterisierung der JOP–Lösung

Die auf Seite 52 vorgestellte Prozedur charakterisiert, wie ein Vergleich mit Abschnitt C.1 zeigt, die MFP–Lösung. Daß $\bigcup\{\mathcal{L}(i) \mid q\|i \in S_{p\|I}\}$ der JOP–Lösung für $q \in S_p$ entspricht, ist weniger einsichtig, denn nach Abschnitt C.1 gilt

$$JOP_{\mathcal{L}(I)}(q) = \bigcup\{[\![pf]\!](\mathcal{L}(I)) \mid pf \in [k_{start}, k), k = (q, \cdot, \cdot) \in \longrightarrow_p\}$$

(beachte auch hier wieder die Korrespondenz zwischen Transitionen und Zuständen). Die Korrektheit dieser Charakterisierung garantiert die folgende Proposition:

Proposition C.4 (Charakterisierung der JOP–Lösung)
$JOP_{\mathcal{L}(I)}(q) = \bigcup\{\mathcal{L}(i) \mid q\|i \in S_{p\|I}\}$

Beweis
Zum Beweis sind zwei Inklusionen zu zeigen:

„$\supseteq$“: Hier genügt es, für ein beliebiges $q \in S_p$ und $k = (q, \cdot, \cdot)$ folgende stärkere Behauptung zu zeigen:

$$\forall\, q\|i \in S_{p\|I}\ \exists\, pf \in P[k_{start}, k).\ \mathcal{L}(i) \subseteq [\![pf]\!](\mathcal{L}(I)).$$

Seien also $q \in S_p$ und $q\|i \in S_{p\|I}$ beliebig. Der Beweis der Behauptung wird mittels vollständiger Induktion über die Länge j eines Pfades von $p\|I$ nach $q\|i$ geführt:

Induktionsanfang: $(j = 0)$
Hier gilt $q \equiv p$ und $i \equiv I$. Mit der Wahl $pf = \epsilon$ folgt
$\mathcal{L}(i) = \mathcal{L}(I) \subseteq [\![\epsilon]\!](\mathcal{L}(I))$.

Induktionsschluß: $(j \longrightarrow j+1)$
Sei nun pf :[6] $p\|I \longrightarrow^j p'\|i' \xrightarrow{a} q\|i$ mit $k' =_{df} (p', a, q)$ und $pf =_{df} pf' \cdot k'$. Dann hat der Pfad pf' von $p\|I$ nach $p'\|i'$ die Länge j. Folglich gilt nach Induktionsvoraussetzung $\mathcal{L}(i') \subseteq [\![pf']\!](\mathcal{L}(I))$.
Für den Induktionsschluß unterscheidet man nun zwei Fälle:

Fall 1: $a \in \mathcal{A}_I$, d.h. (vgl. Definition 1.8 (5), $\mathcal{A}_I \subseteq \mathcal{A}_p$) $p' \xrightarrow{a} q$ und $i' \xrightarrow{a} i$.

$$\begin{aligned} & \mathcal{L}(i) \\ (\text{Definition 1.7, } i' \xrightarrow{a} i) \qquad & \subseteq \mathcal{L}(i')_a \end{aligned}$$

[6] Diese Bezeichnung soll suggerieren, daß es sich bei diesem Pfad schon um den Pfad pf aus obiger Behauptung mit der gewünschten Eigenschaft handelt.

$$\begin{array}{lcl}
(\text{Definition von } [\![k']\!],\ a \in \mathcal{A}_I) & = & [\![k']\!](\mathcal{L}(i')) \\
(\text{Induktionsvor. \& Monotonie}) & \subseteq & [\![k']\!]([\![pf']\!](\mathcal{L}(I))) \\
(\text{Definition von } pf) & = & [\![pf]\!](\mathcal{L}(I))
\end{array}$$

Fall 2: $a \notin \mathcal{A}_I$, d.h. (vgl. Definition 1.8 (3)) $p' \xrightarrow{a} q$ und $i' = i$.

$$\begin{array}{lcl}
\mathcal{L}(i) & = & \mathcal{L}(i') \\
(\text{Induktionsvoraussetzung}) & \subseteq & [\![pf']\!](\mathcal{L}(I)) \\
(\text{Definition von } [\![k']\!],\ a \notin \mathcal{A}_I) & = & [\![k']\!]([\![pf']\!](\mathcal{L}(I))) \\
(\text{Definition von } pf) & = & [\![pf]\!](\mathcal{L}(I))
\end{array}$$

Per Induktionsprinzip folgt die Behauptung.

„$\subseteq$“: Hier genügt es, für einen beliebigen Pfad $pf \in P[k_{start}, k)$ mit $k = (q, \cdot, \cdot) \in \longrightarrow_p$ folgendes zu zeigen:

$$w \in [\![pf]\!](\mathcal{L}(I)) \quad \text{impliziert} \quad \exists\, q\|i \in S_{p\|I}.\ w \in \mathcal{L}(i).$$

Die Behauptung wird mittels vollständiger Induktion über die Länge j des Pfades pf bewiesen.

Induktionsanfang: $(j = 0)$
Hier ist $pf = \epsilon$, sowie $k \equiv k_{start}$ und $q \equiv p$. Wähle $i \equiv I$, dann gilt $p\|i \in S_{p\|I}$, und die Behauptung folgt mit $[\![\epsilon]\!](\mathcal{L}(I)) = \mathcal{L}(i)$.

Induktionsschluß: $(j \longrightarrow j+1)$
Sei nun $pf \in P[k_{start}, k)$ ein Pfad der Länge $j+1$ mit $pf =_{df} pf' \cdot k'$. Dabei sei etwa $k' =_{df} (p', a, q)$, und der Pfad $pf' \in P[k_{start}, k')$ hat die Länge j. Nach Induktionsvoraussetzung gibt es zu einem beliebigen Wort $w' \in [\![pf']\!](\mathcal{L}(I))$ einen Zustand $p'\|i' \in S_{p\|I}$ mit $w' \in \mathcal{L}(i')$. Unterscheide nun zwei Fälle:

Fall 1: $a \in \mathcal{A}_I$. Dann gilt:

$$\begin{array}{lcl}
 & & w \in [\![pf]\!](\mathcal{L}(I)) \\
(\text{Definition von } pf) & \Rightarrow & w \in [\![k']\!]([\![pf']\!](\mathcal{L}(I))) \\
(\text{Definition von } [\![k']\!],\ a \in \mathcal{A}_I) & \Rightarrow & w \in ([\![pf']\!](\mathcal{L}(I)))_a
\end{array}$$

$$\begin{array}{lcl}
(\text{Induktionsvor. \& Monotonie}) & \Rightarrow & w \in \mathcal{L}(i')_a \\
(\text{Definition 1.7}^7) & \Rightarrow & \exists\, q\|i \in S_{p\|I}.\ w \in \mathcal{L}(i)
\end{array}$$

Fall 2: $a \notin \mathcal{A}_I$, d.h. (vgl. Definition 1.8 (3)) $p'\|i' \xrightarrow{a}_{p\|I} q\|i'$. Mit $i =_{df} i'$ folgt:

$$\begin{array}{lcl}
 & & w \in [\![pf]\!](\mathcal{L}(I)) \\
(\text{Definition von } pf) & \Rightarrow & w \in [\![k']\!]([\![pf']\!](\mathcal{L}(I))) \\
(\text{Definition von } [\![k']\!],\ a \notin \mathcal{A}_I) & \Rightarrow & w \in [\![pf']\!](\mathcal{L}(I)) \\
(\text{Induktionsvoraussetzung}) & \Rightarrow & w \in \mathcal{L}(i') \\
(i = i') & \Rightarrow & w \in \mathcal{L}(i)
\end{array}$$

Per Induktionsprinzip folgt die Behauptung.

Mit „$\subseteq$“ und „$\supseteq$“ folgt die Aussage der Proposition.

Die Korrektheit des Algorithmus (vgl. Satz 2.16) folgt nun unmittelbar aus dem Koinzidenztheorem C.3. Daß der Algorithmus terminiert, ist nicht unmittelbar ersichtlich, da die maximale Länge einer Kette in der Halbordnung $\langle \underline{\text{Languages}}; \subseteq \rangle$ unendlich sein kann. Klarheit verschafft erst eine genauere Betrachtung dieser Datenflußanalyse im nächsten Abschnitt.

Komplexität

Hier soll eine *Zeitkomplexitätsanalyse* (worst–case–analysis) für den in Abschnitt C.1 vorgestellten Workset–Algorithmus bzgl. obiger Anwendung durchgeführt werden. Dabei sei $p = (S_p, \mathcal{A}_p, \longrightarrow_p, \uparrow_p), p)$ ein beliebiger, aber fester Prozeß und $I = ((S_I, \mathcal{A}_I, \longrightarrow_I, \emptyset), I)$ eine beliebige, aber feste Interfacespezifikation.

Als Maß für die (Zeit–)Komplexität des Algorithmus dient die maximale Anzahl seiner elementaren Rechenschritte (bis zu seiner Termination). Dabei wird hier das „Betrachten“ einer Transition von p als elementarer Rechenschritt aufgefaßt. Im folgenden bezeichnet m die Anzahl der Transitionen von p, d.h. $m =_{df} \mid \longrightarrow_p \mid$, und n die Anzahl der Zustände der Interfacespezifikation I, also $n =_{df} \mid S_I \mid$. Ferner seien p und I beschränkt verzweigend, d.h. für alle $k \in \longrightarrow_p$ gilt: $|succ(k)| \leq$ konst. Weiterhin sei der Aufwand für die Anwendung einer lokalen abstrakten Semantikfunktion $[\![k]\!]$ für $k \in \longrightarrow_p$ konstant, da die Funktionen

[7] Beachte: $w \in \mathcal{L}(i')_a \Rightarrow aw \in \mathcal{L}(i') \Rightarrow w \in \mathcal{L}(i)$ für ein i mit $i' \xrightarrow{a} i$ und zusammen mit Definition 1.8 (5) $p'\|i' \xrightarrow{a}_{p\|I} q\|i$, also insbesondere $q\|i \in S_{p\|I}$.

einmalig vor Ausführung des Algorithmus mit einem Aufwand in $O(m \cdot n)$ berechnet werden können. Dabei genügt es, sich das Ergebnis der Anwendung einer solchen Funktion auf die Sprachmengen $\mathcal{L}(i)$ für $i \in S_I$ zu merken. Die Effizienz kann unter obigen Annahmen durch eine spezielle Darstellung der betrachteten Sprachmengen erreicht werden, die, wie unten erläutert wird, auf einer surjektiven Abbildung zwischen der Potenzmenge von S_I und den Sprachmengen beruht.

Die Komplexität des Workset–Algorithmus setzt sich summativ aus der Komplexität für seine Initialisierung und jener für die Ausführung seiner While–Schleife zusammen.
Bei der Initialisierung wird jede Transition mit einer Sprachmenge beschriftet. Folglich wird jede Transition genau einmal betrachtet, so daß die Komplexität für die Initialisierung in $O(m)$ liegt.

Um die Komplexität für die While–Schleife zu bestimmen, muß gezählt werden, wie oft die While–Schleife höchstens durchlaufen wird. Desweiteren ist der Aufwand für einen Schleifendurchlauf des Schleifenrumpfes nach oben abzuschätzen. Das Produkt aus beidem ergibt dann den Aufwand für die Schleife.
Der Aufwand für eine einmalige Ausführung des Schleifenrumpfes ist konstant, da der Aufwand für die Anwendung von $[\![\cdot]\!]$ nach Voraussetzung konstant und p beschränkt verzweigend ist.

Weitaus schwieriger als diese Betrachtungen ist es, festzustellen, wie oft die While–Schleife höchstens durchlaufen wird. Präziser muß gefragt werden: Wie oft kann eine Transition $k \in \longrightarrow_p$ maximal in das Workset gelangen? Eine Transition k mit momentaner Beschriftung $\mathcal{L}$ kommt gemäß des Algorithmus genau dann in das Workset, wenn für eine Vorgängertransition l mit Beschriftung $\mathcal{L}'$ $[\![l]\!](\mathcal{L}') \cup \mathcal{L} \supset \mathcal{L}$ gilt, d.h. falls sich die Beschriftung von k geändert hat (beachte dabei die Monotonie der Funktion $[\![l]\!]$). Anders formuliert lautet nun die Frage: Was ist die maximale Länge einer (möglichen) Kette bzgl. der mengenvollständigen Halbordnung $\langle \underline{\text{Languages}}; \subseteq \rangle$? Dabei ist zu berücksichtigen, daß die betrachteten Sprachen bzw. Vereinigungen von Sprachen eine Struktur gemäß Definition 1.7 und insbesondere die Eigenschaft der Präfixabgeschlossenheit besitzen. Folglich wird für die hier analysierte spezielle Datenflußanalyse nicht die Menge aller Sprachen betrachtet, die über dem Alphabet $\mathcal{A}_I$ gebildet werden können, sondern lediglich eine Teilmenge davon. Mit anderen Worten ist nicht die mengenvollständige Halbordnung $\langle \underline{\text{Languages}}; \subseteq \rangle$ Grundlage für die Datenflußanalyse, sondern eine mengenvollständige Halbordnung $\langle H; \subseteq \rangle$ mit $H \subset \underline{\text{Languages}}$. Diese Beobachtung ist sehr wichtig, da es in $\langle \underline{\text{Languages}}; \subseteq \rangle$ unendlich lange Ketten gibt,[8] und soll im folgenden exakt formuliert werden.

Nach Definition 1.7 der Sprache eines Prozesses und nach Definition des lokalen abstrakten Semantikfunktionals $[\![\cdot]\!]$ sowie des Algorithmus enthält die Menge H alle Sprachen $\mathcal{L}(i)$ für $i \in S_I$ und die Mengen $\bigcup \{\mathcal{L}(i) \mid i \in S_I'\}$ für $S_I' \subseteq S_I$, die aus der Vereinigung solcher Sprachen entstehen. Die in H enthaltenen Sprachen lassen sich durch sog. *Multiprozesse*

[8] Unter der Voraussetzung, daß es unendlich lange Ketten gibt, ist der Workset–Algorithmus ineffektiv.

bzgl. I charakterisieren.

Definition C.5 (Multiprozesse)
Gegeben sei ein Prozeß $I = ((S_I, \mathcal{A}_I, \longrightarrow_I, \uparrow_I), I)$.
Ein Tupel $P =_{df} ((S_I, \mathcal{A}_I, \longrightarrow_I, \uparrow_I), P)$ *mit* $P \subseteq S_I$ *heißt Multiprozeß bzgl.* I.
Die Menge aller Multiprozesse bzgl. I *wird mit* $MP(I)$ *bezeichnet.*

Ein Multiprozeß bzgl. I besteht also aus demselben erweiterten Transitionssystem wie I und kann im Gegensatz zu I keinen, einen oder mehrere Startzustände besitzen. In natürlicher Weise dehnt sich die *Tracesemantik* eines Prozesses[9] auf Multiprozesse aus: Ist $P = ((S_I, \mathcal{A}_I, \longrightarrow_I, \uparrow_I), P)$ ein Multiprozeß bzgl. I, so ist die Tracesemantik von P definiert durch:

$$\mathcal{L}(P) =_{df} \bigcup \{\mathcal{L}(q) \mid q \in P\}.$$

Folglich ist die hier interessierende Menge H von Sprachen nichts anderes als die Menge der Tracesemantiken aller Multiprozesse bzgl. I, welche mit $MP(I)$ identifiziert werden kann (wir schreiben: $H \hat{=} MP(I)$). Deswegen bildet sie zusammen mit der Ordnung $\subseteq$ eine Halbordnung. Da $H \subset \underline{\text{Languages}}$ gilt und $\langle \underline{\text{Languages}}; \subseteq \rangle$ eine mengenvollständige Halbordnung ist, ist auch $\langle H; \subseteq \rangle$ eine mengenvollständige Halbordnung. Ferner ist nach Definition C.5 ein Multiprozeß P bzgl. I durch seine Startzustandsmenge P eindeutig bestimmt.

Betrachte nun die Abbildung $\phi : \mathcal{P}(S_I) \longrightarrow H \hat{=} MP(I)$ definiert durch $P \mapsto \mathcal{L}(P)$, wobei $\langle \mathcal{P}(S_I); \subseteq \rangle$ nach Lemma A.6 ebenfalls eine mengenvollständige Halbordnung ist. Die Abbildung ϕ ist aufgrund ihrer Definition zusammen mit den Definitionen von H und von $\mathcal{L}(P)$ für $P \in \mathcal{P}(S_I)$ surjektiv. Daher erlaubt sie eine effiziente Darstellung der Elemente von H, die i.allg. Sprachmengen mit unendlich vielen Worten sind, durch (endliche) Mengen von Zuständen des Prozesses I.

Für die Komplexitätsanalyse ist die Abbildung ϕ ein wichtiges Hilfsmittel, um zu zeigen, daß die maximale Länge einer Kette in $\langle H; \subseteq \rangle$ und in $\langle \mathcal{P}(S_I); \subseteq \rangle$ gleich ist.

Lemma C.6
Die maximalen Kettenlängen der Halbordnungen $\langle \mathcal{P}(S_I); \subseteq \rangle$ *und* $\langle H; \subseteq \rangle$ *sind gleich.*

Beweis
Zum Beweis sind zwei Inklusionen zu zeigen:

1. Die maximale Kettenlänge von $\langle H; \subseteq \rangle$ ist mindestens so groß wie die von $\langle \mathcal{P}(S_I); \subseteq \rangle$, da ϕ ordnungserhaltend ist, denn:

[9] Die Sprache eines Prozesses definiert seine Tracesemantik.

Seien $P, P' \in \mathcal{P}(S_I)$ mit $P \subseteq P'$ beliebig. Zeige $\phi(P) \subseteq \phi(P')$:

$$
\begin{array}{lcl}
 & & \phi(P) \\
(\text{Definition von } \phi) & = & \mathcal{L}(P) \\
(\text{Definition von } \mathcal{L}(P)) & = & \bigcup\{\mathcal{L}(q) \mid q \in P\} \\
(P \subseteq P') & \subseteq & \bigcup\{\mathcal{L}(q) \mid q \in P'\} \\
(\text{Definition von } \mathcal{L}(P')) & = & \mathcal{L}(P') \\
(\text{Definition von } \phi) & = & \phi(P')
\end{array}
$$

2. Die maximale Kettenlänge von $\langle H; \subseteq\rangle$ ist höchstens so groß wie die von $\langle \mathcal{P}(S_I); \subseteq\rangle$, da gilt:

$$
\begin{array}{l}
\forall\, n \geq 0.\ \ \mathcal{L}_1 \subset \mathcal{L}_2 \subset \cdots \subset \mathcal{L}_n \ \text{ impliziert} \\
\qquad \exists\, P_1, P_2, \cdots, P_n.\ P_1 \subset P_2 \subset \cdots \subset P_n \ \ \wedge \ \ \phi(P_i) = \mathcal{L}_i,\ 1 \leq i \leq n
\end{array}
$$

Beweis mittels Induktion über n:

Induktionsanfang ($n = 0$):

Für $n = 0$ ist nichts zu zeigen.

Induktionsschluß ($n \longrightarrow n + 1$):

Gelte $\mathcal{L}_1 \subset \cdots \subset \mathcal{L}_n \subset \mathcal{L}_{n+1}$. Nach Induktionsannahme existieren P_i, $1 \leq i \leq n$, mit $P_1 \subset P_2 \subset \cdots \subset P_n$ und $\phi(P_i) = \mathcal{L}_i$, $1 \leq i \leq n$. Da ϕ surjektiv ist, existiert ein $\overline{P}_{n+1}$ mit $\phi(\overline{P}_{n+1}) = \mathcal{L}_{n+1}$. Setze nun $P_{n+1} =_{df} P_n \cup \overline{P}_{n+1}$. Dann gilt offensichtlich $P_n \subset P_{n+1}$, $\phi(P_{n+1}) = \phi(\overline{P}_{n+1})$ (vgl. Definition von $\mathcal{L}(\cdot)$) und

$$
\begin{array}{lcl}
 & & \phi(P_{n+1}) \\
(\text{Definition von } \phi) & = & \mathcal{L}(P_{n+1}) \\
(\text{Definition von } P_{n+1}) & = & \mathcal{L}(P_n \cup \overline{P}_{n+1}) \\
(\text{Definition von } \mathcal{L}(\cdot)) & = & \mathcal{L}(P_n) \cup \mathcal{L}(\overline{P}_{n+1}) \\
(\text{Definition von } \phi) & = & \phi(P_n) \cup \phi(\overline{P}_{n+1})
\end{array}
$$

$$\begin{aligned}
(\text{nach Konstruktion}) &= \mathcal{L}_n \cup \mathcal{L}_{n+1} \\
(\mathcal{L}_n \subset \mathcal{L}_{n+1}) &= \mathcal{L}_{n+1}
\end{aligned}$$

Per Induktionsprinzip folgt die Behauptung.

Aus (1) und (2) folgt die Behauptung des Lemmas.

Da die maximale Kettenlänge in $\langle \mathcal{P}(S_I); \subseteq \rangle$ offensichtlich $n+1$ beträgt, besitzt $\langle H; \subseteq \rangle$ auch nur Ketten der Länge höchstens $n+1$.
Nach diesen Ausführungen gelangt eine Transition höchstens $(n+1)$–mal in das Workset, so daß die While–Schleife höchstens $(m \cdot (n+1))$–mal durchlaufen wird. Da der Aufwand für den Schleifenrumpf, wie bereits gezeigt, konstant ist, beträgt die Komplexität der gesamten While–Schleife $O(m \cdot n)$.

Damit ergibt sich für den Gesamtaufwand des Workset–Algorithmus:

$$\underbrace{O(m \cdot n)}_{\text{Berechnung von } [\![\cdot]\!]} + \underbrace{O(m)}_{\text{Initialisierung}} + \underbrace{O(m \cdot n)}_{\text{While–Schleife}} = O(m \cdot n)$$

Fazit: Der Workset–Algorithmus terminiert und ist für die in dieser Arbeit vorgestellte Anwendung effizient.

Literaturverzeichnis

[CLM89] **E. M. Clarke; D. E. Long; K. L. McMillan**: Compositional Model Checking. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science, Computer Society Press* (1989), Seite 353-362.

[CS] **R. Cleaveland; B. Steffen**: A Preorder for Partial Process Specifications. In *Proceedings of CONCUR'90, LNCS 458* (1990), Seite 141-151.

[CS90] **R. Cleaveland; B. Steffen**: When is Partial Adequate? A Logic–Based Proof Technique for Partial Specifications. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science, Computer Society Press* (1990), Seite 108-117.

[Fer88] **J.–C. Fernandez**: Aldébaran: Un Système de Vérification par Réduction de Processus Communicants. *Université de Grenoble* (1988).

[GS91] **S. Graf; B. Steffen**: Compositional Minimization of Finite State Systems. *Aachener Informatik–Berichte Nr. 91–23, RWTH Aachen* (1991).

[KS91] **J. Knoop; B. Steffen**: The Interprocedural Coincidence Theorem. *In Proceedings of the Fourth International Conference on Compiler Construction CC'92, LNCS 641* (1992), Seite 125-140.

[Kn93] **J. Knoop**: Optimal Interprocedural Program Optimization: A new Framework and its Application. *Christian–Albrechts–Universität Kiel* (1993).

[KU77] **J. B. Kam; J. D. Ullman**: Monotone Data Flow Analysis Frameworks. *Acta Informatica 7* (1977), Seite 309-317.

[LT87] **K. G. Larsen; B. Thomsen**: Compositional Proofs by Partial Specification of Processes. *Technical Report 87-20, University of Aalborg* (1987).

[L92] **G. Lüttgen**: Kompositionelle Minimierung endlicher Transitionssysteme. *Ausarbeitung zum Seminar „Analyse und Verifikation". RWTH–Aachen* (Sommersemester 1992).

[L93] **G. Lüttgen**: Interprozedurale Datenflußanalyse. *Ausarbeitung zum Seminar „Analysetechniken zur Systemoptimierung". RWTH–Aachen* (Wintersemester 1992/93).

[Mi89] **R. Milner**: Communication and Concurrency. *Prentice Hall* (1989).

[St91] **B. Steffen**: Theorie verteilter Systeme. *Vorlesungsmitschrift, RWTH Aachen* (Wintersemester 1991/92).

[St92] **B. Steffen**: Analyse und Verifikation. *Vorlesungsmitschrift, RWTH Aachen* (Wintersemester 1992/93).

[St92/2] **B. Steffen**: Das PASSAU–Projekt (Process–oriented Analysis and Synthesis Supporting Abstraction and Unification). *Universität Passau* (1992).

[Wa88] **D. J. Walker**: Bisimulation and Divergence. In *Proceedings of the Third Annual Symposium on Logic in Computer Science, Computer Society Press* (1988), Seite 186-192.