

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Prozessalgebra

Kurzsriptum zur Vorlesung

Sommersemester 2005

F. Kröger

INSTITUT FÜR INFORMATIK

Lehr- und Forschungseinheit für Programmierung und Softwaretechnik

Inhaltsverzeichnis

Literaturhinweise	3
1 Kommunizierende Prozesse	4
1.1 Parallele Prozesse	4
1.2 Prozesskommunikation	5
1.3 Formale Theorien	9
2 Grundlagen der Prozessalgebra	12
2.1 Beschreibung und Verhaltensäquivalenz von Prozessen	12
2.2 Basissprache und Bisimulations-Semantik	14
2.3 Axiomatisierung	17
3 Kommunizierende Prozesse	19
3.1 Parallelität	19
3.2 Kommunikation	21
3.3 Verklemmung und Restriktion	23
3.4 Denotationelle Semantik	26
4 Rekursion	29
4.1 Rekursive Spezifikationen	29
4.2 ACP mit Rekursion	32
4.3 Approximation von Prozessen	37
4.4 Sprachvarianten	39
5 Abstraktion	41
5.1 Unsichtbare Aktion und Abstraktionsoperator	41
5.2 Axiomatisierung	44
5.3 Fairness der Abstraktion	47
5.4 Beispiel: Das Alternating Bit Protocol	49
6 Andere Semantiken	52
6.1 Spur-Semantik	52
6.2 Readiness-Semantik	53
6.3 Failure-Semantik	55

Literaturhinweise

Begleitende Literatur:

Baeten/Weijland: Process Algebra. Cambridge University Press 1990.

Baeten (ed.): Applications of Process Algebra. Cambridge University Press 1990.

Fokkink: Introduction to Process Algebra. Springer 2000.

(zu Kapitel 5:)

Bergstra/Klop: Algebra of communicating processes with abstraction. TCS **37**, 77-121 (1985).

Baeten/Bergstra/Klop: On the consistency of Kooman's fair abstraction rule. TCS **51**, 129-176 (1987).

Eine Auswahl weiterer Lehrbücher:

Ben-Ari: Principles of concurrent and distributed programming. Prentice Hall 1990.

Hennessy: Algebraic Theory of Processes. The MIT Press 1988.

Herrtwich/Hommel: Kooperation und Konkurrenz – nebenläufige, verteilte und echtzeitabhängige Programmsysteme. Springer 1989.

Hoare: Communicating Sequential Processes. Prentice Hall 1985.

Magee/Kramer: Concurrency. Wiley 1999.

Milner: Communication and Concurrency. Prentice Hall 1989.

Kapitel 1

Kommunizierende Prozesse

1.1 Parallele Prozesse

Parallelität

Grundbegriffe

- *(Sequentieller) Prozess*: Folge von *Aktionen* (“Berechnungsschritten”).
- *Sequentielles Programm*: beschreibt Prozesse (“Rechenvorgänge”).
- *Parallele Prozesse*: Prozesse, die *parallel (nebenläufig)* zueinander ablaufen (können).
- *Paralleles Programm*: beschreibt parallele Prozesse.

Vorkommen von Parallelität

- *Multitasking*: Aufteilung von Anwenderprogrammen in parallel ausführbare Teile.
- *Reaktive Systeme*: Auf “Endlosbetrieb” ausgelegte Systeme mit problemgegeben parallelen Prozessen.

Beachte: Neben der hier betrachteten Parallelität “im Programm” auch möglich: Parallelität in der Hardware (Beispiel: Vektorrechner).

Interleaving

Als “paralleler Ablauf” von Prozessen oft nur betrachtet: alle sequentiellen *Verzahnungen* der Prozesse (“Parallelität modelliert als *interleaving*”).

“Kleinste Einheiten”, die verzahnt werden können: *atomare Aktionen* (abhängig vom Abstraktionsgrad der Betrachtung).

Andere gebräuchliche Terminologie:

- “parallel”: nur wenn tatsächlich mehrere Prozessoren zur Verfügung stehen,
- “nebenläufig”: nur für beliebige Verzahnungen.

Zusammenwirken von Prozessen

Grundmuster des Zusammenwirkens von (zueinander parallelen) Prozessen:

- **Kooperation:** Verschiedene Prozesse lösen Teilaufgabe im Rahmen einer Gesamtaufgabe.
- **Konkurrenz:** Aktionen eines Prozesses behindern einen anderen Prozess.

Erforderlich dafür: Koordination (**Synchronisation**) der Prozesse.

Einige Synchronisations-Grundmuster:

- **Erzeuger-Verbraucher**-Synchronisation.
- **Wechselseitiger Ausschluss**.
- **Leser-Schreiber**-Synchronisation.

Einige wünschenswerte Eigenschaften von Synchronisationen:

- **Verklemmungsfreiheit**.
- **Aushungerungsfreiheit**.
- **Fairness**.

1.2 Prozesskommunikation

Kommunikationskonzepte

Beim Zusammenwirken von parallelen Prozessen erfolgt (i. Allg.) ein Datenaustausch (**Kommunikation**) zwischen den Prozessen (problemgegeben und/oder zur Synchronisation).

Beachte: Synchronisation und Kommunikation bedingen sich (i. Allg.) gegenseitig:

- Synchronisation erfordert Kommunikation,
- Kommunikation erfordert (i.Allg.) Synchronisation (von Informationsangabe und -aufnahme).

Zwei wesentliche Konzepte der Kommunikation:

- Kommunikation über **gemeinsame Datenbereiche** (Variablen, Semaphore, Monitore, ...).

- Kommunikation durch *Nachrichtenaustausch*, d.h. über *Nachrichten* (Dateneinheiten die von einem zu einem (oder mehreren) anderen übergeben werden).

Dazu zwei grundlegende Operationen: *Senden* und *Empfangen*.

Bemerkung:

Parallele Programme mit Kommunikation über Nachrichten heißen auch *verteilte Programme*.

Nachrichtenaustausch

Zwei Konzepte für Nachrichtenaustausch:

- **Asynchrone Kommunikation:**

Nachrichte wird gesendet und beim Empfänger in einen Nachrichten-Puffer abgelegt. Der Empfänger holt die Nachricht danach aus dem Puffer. (Sende- und Empfangs-Aktion geschehen zu verschiedenen Zeitpunkten).

Diese Art des Nachrichtenaustauschs entspricht i.w. der Kommunikation über einen gemeinsamen Datenbereich.

- **Synchrone Kommunikation:**

Nachrichtenaustausch (d.h. Ausführung einer Sende- und einer Empfangs-Aktion) als *eine* gleichzeitige (atomare) Aktion.

Kommunizierende Prozesse

Schema der synchronen Kommunikation:

Prozess1: ...;
 α : Sende N ;
 ...

Prozess2: ...;
 β : Empfang N ;
 ...

1. Kommt Prozess1 an die Stelle α , so muss er warten, bis Prozess2 an die Stelle β kommt.
2. Kommt Prozess2 an die Stelle β , so muss er warten, bis Prozess1 an die Stelle α kommt.
3. Sind Prozess1 und Prozess2 an den Stellen α und β , so kann N übertragen werden; anschließend fahren Prozess1 und Prozess2 im Anschluss an die Stellen α und β fort.

Sprechweise: *kommunizierende Prozesse* für:

“synchron über Nachrichten kommunizierende Prozesse”.

CSP

(Prototyp-) Programmiersprache für kommunizierende Prozesse: *CSP* (“Communicating Sequential Processes”).

Zugrundegelegt:

- Prozesse,
- **Kanäle** zwischen jeweils zwei Prozessen.

Wesentliche Sprachelemente:

1. Anweisungen:

- “Leere” Anweisung: nop (keine Wirkung).
- übliche Wertzuweisungen: $x := t$.
- Ausgabe- (Sende-) Anweisungen: $c!t$ (“Sende den Wert von t auf Kanal c ”).
- Eingabe- (Empfange-) Anweisungen: $c?a$ (“Empfange den Wert auf Kanal c und lege diesen auf Variablen a ab”).

Kommunikation kommt zustande zwischen zwei **korrespondierenden** Ein-/Ausgabeanweisungen (d.h.: mit gleichem Kanal und Typübereinstimmung von Aufnahmevariable und gesendetem Wert).

Sprechweise: **Kommunikationsanweisung** für Ein- oder Ausgabeanweisung.

2. Ablaufsteuerung:

- Alternativanweisungen:
$$\left[\begin{array}{l} \text{Wächter}_1 \rightarrow \text{Anweisungsfolge}_1 \\ \square \text{Wächter}_2 \rightarrow \text{Anweisungsfolge}_2 \\ \vdots \\ \square \text{Wächter}_n \rightarrow \text{Anweisungsfolge}_n \\ \end{array} \right]$$

Wächter haben die Form $\text{Bedingung} \{ ; \text{Kommunikationsanweisung} \}_0^1$

Anweisungsfolgen haben die Form $\text{Anweisung}_1; \dots; \text{Anweisung}_k$ ($k \geq 1$).

Nichtdeterminismus: Mehrere Wächter können erfüllt sein.

Abkürzungen:

Anweisungsfolge statt $\text{TRUE} \rightarrow \text{Anweisungsfolge}$,
 $\text{Kommunikationsanweisung}$ statt $\text{TRUE}; \text{Kommunikationsanweisung}$.

- Wiederholungsanweisung: $*\text{Alternativanweisung}$

3. Datenstrukturen, Variablen, Vereinbarungen u.ä.: wie üblich.

4. Programm: $\left[\text{Prozess}_1 \parallel \dots \parallel \text{Prozess}_m \right]$
 ($m \geq 1, \text{Prozess}_1 \dots \text{Prozess}_m$: aufgebaut gemäß 1) – 3)).

Auswertung eines Wächters $B; a$ (a mit Kanal c):

- $B = \text{wahr}$ (und a “kann sofort mit einem Partner kommunizieren”) \rightsquigarrow Wert *wahr*,
- $B = \text{falsch}$ (oder “alle Partner bzgl. c ” sind bereits beendet) \rightsquigarrow Wert *falsch*,

- $B = \text{wahr}$ (und “Partner bzgl. c ” existiert, ist aber nicht “zur Kommunikation bereit”)
 \leadsto keine Auswertung möglich.

Wirkung einer Alternativanweisung:

- Alle Wächter *falsch*: kein Effekt,
- mindestens ein Wächter *wahr*: die Kommunikation in einem der betreffenden Wächter (falls vorhanden) und die entsprechende Anweisungsfolge werden ausgeführt,
- sonst: Prozess muss warten.

Wirkung einer Wiederholungsanweisung $*A$:

- A wird so lange wiederholt, bis alle Wächter in A *falsch* sind.

Beispiele

1. Wechselseitiger Ausschluss zweier Prozesse P1 und P2

```
[ P1:    *[unkritischer_Abschnitt;
          c1! eintritt;
          kritischer_Abschnitt;
          c1! austritt
        ]
||
P2:    *[unkritischer_Abschnitt;
          c2! eintritt;
          kritischer_Abschnitt;
          c2! austritt
        ]
||
Synchr: var a : SIGNAL;
        *[ c1? a → c1? a
          [] c2? a → c1? a
        ]
]
```

2. Erzeuger-Verbraucher-System

```

[ Erzeuger:   *[ erzeuge_Objekt obj;
                c1! obj
              ]
  ||
  Verbraucher: var erhalten : OBJEKTTYP;
                *[ c2? erhalten;
                  verbrauche_Objekt erhalten
                ]
  ||
  Puffer:     var p : PUFFER;
                pobj : OBJEKTTYP;
                go : NAT;   (* initialisiert mit 0*)
                *[ go < maxanz(p); c1? pobj → speichere pobj in p;
                    go := go + 1
                [] go > 0; c2! top(p) → entferne top(p) aus p;
                    go := go - 1
                ]
  ]

```

1.3 Formale Theorien

Eine wahre Begebenheit

Der Clayton-Tunnel in England hatte zwei Gleisspuren, die jeweils in entgegengesetzter Richtung befahren wurden. Der Tunnel wurde 1841 mit einem neuen Signalsystem ausgestattet, um (aus Sicherheitsgründen) zu gewährleisten, dass sich immer nur ein Zug pro Spur im Tunnel befand. An jedem Tunnelende wurde eine Ampel installiert, die automatisch auf rot gesetzt wurde, wenn ein Zug ein grünes Signal passiert hatte. Außerdem wurden beide Ampeln rund um die Uhr durch zwei Wärter überwacht, die ihre jeweilige Ampel wieder auf grün stellten, nachdem sie sich vergewissert hatten, dass Züge, die in den Tunnel eingefahren waren, am anderen Ende wieder aufgetaucht waren. Zu diesem Zweck gab es eine Telegraphenleitung zwischen den beiden Wärtern, auf der einige fest vorgegebene Nachrichten ausgetauscht werden konnten.

Wenn ein Zug in den Tunnel einfuhr, teilte der betreffende Wärter seinem Kollegen dies durch

train_in_tunnel

mit. Sobald er vom Kollegen durch

tunnel_is_free

über die Ausfahrt des Zuges auf dessen Seite benachrichtigt war, setzte er die Ampel wieder auf grün. Darüber hinaus konnte jeder Wärter noch die Anfrage

has_train_left?

an seinen Kollegen richten.

Als zusätzliche Sicherheit war je eine Signalglocke eingebaut, die den Wärter warnte, wenn seine Ampel aufgrund einer Störung bei Zugeinfahrt nicht auf rot wechselte. Er konnte dann eine rote und eine weiße Flagge benutzen, um die Züge entsprechend zu dirigieren.

Im Jahr 1861 geschah folgendes:

- Ein Zug passierte Ampel A, und diese wechselte nicht auf rot. Der durch die Glocke gewarnte Wärter schickte die Meldung *train_in_tunnel* ab und holte die rote Fahne, um den nächsten Zug zu warnen.
- Dieser kam jedoch so schnell, dass er die grüne Ampel passierte. Der Fahrer sah aber gerade noch den heraneilenden Wärter mit der roten Flagge, als er in den Tunnel einfuhr. Ein dritter ankommender Zug wurde rechtzeitig gewarnt und hielt vor dem Tunnel an.
- Wärter A kehrte nun zu seinem Häuschen zurück und telegraphierte noch einmal *train_in_tunnel*, um anzuzeigen, dass zwei Züge im Tunnel waren.
- Um den dritten Zug weiterfahren lassen zu können, musste Wärter A nun erfahren, wann die beiden vorherigen Züge den Tunnel verlassen hatten. Mit der Anfrage *has_train_left?* wollte er seinen Kollegen auf das Problem aufmerksam machen. Der konnte aber, als er den ersten Zug ausfahren sah, gemäß seinen Instruktionen nur *tunnel_is_free* zurücksenden.
- Für Wärter A war nicht klar, ob er nun zwei solche Meldungen abwarten sollte oder ob *tunnel_is_free* wörtlich zu nehmen war, d.h. dass der Tunnel wieder frei war. Er entschied sich für letzteres und ließ mit Hilfe der weißen Flagge den dritten Zug in den Tunnel einfahren. Der Lokführer des zweiten Zuges, der die rote Flagge noch gesehen hatte, brachte seinen Zug in der Mitte des Tunnels zum Stehen und fuhr nach einiger Überlegung, um ganz sicher zu gehen, rückwärts wieder aus dem Tunnel heraus.
- Bei dem nachfolgenden Zusammenstoß starben 21 Personen, 176 wurden verletzt.

Sichere Systeme

Vermeidung (zumindest: Reduzierung) von Systemfehlern erfordert:

- Präzise Semantik der Sytembestandteile.
- Präzise Spezifikation der Systemanforderungen.
- Methoden zur Konstruktion korrekter Systeme.
- Methoden zur Systemverifikation.
- ...

(Mindestens) dazu nötig: Formale Theorien (d.h. formale Modellbildungen und formale Methoden) für Systeme.

Wichtige formale Theorien für Systeme paralleler Prozesse:

- (Petri-) Netze.
- Temporale Logik(en).
- Prozessalgebra.

Prozessalgebra

Spezielle (“algebraische”) Theorien für kommunizierende Prozesse (i.w. drei leicht unterschiedliche Ansätze):

- *CCS* (Calculus of Communicating Systems, Milner 1980).
- *TCSP* (Theoretical CSP, Hoare et al. 1984)
 (“abstrakte Fassung” von CSP, oft auch nur CSP genannt).
- *Prozessalgebra* (Bergstra/Klop 1982).

Kapitel 2

Grundlagen der Prozessalgebra

2.1 Beschreibung und Verhaltensäquivalenz von Prozessen

Algebraische Beschreibungssprachen

Grundlage einer Theorie für Prozesse: Sprache zur Beschreibung (Darstellung) von Prozessen, so dass

- diese formal behandelbar sind,
- abstrahiert wird auf das, was man untersuchen will.

Hier betrachtet: *algebraische* Sprachen.

Charakteristika:

- Zugrundegelegt: “atomare” (“elementare”, “unteilbare”) Aktionen,
- “Zusammensetzen” von Prozessen aus solchen Aktionen mit gewissen Operationen (in abstrakter Nachbildung von Programmkonstruktionen),
- Hauptziel der Theorie dazu: Instrumente zur Analyse der “Gleichheit” solchermaßen beschriebener Prozesse.

Basiskonzepte zur Beschreibung von Prozessen

- **Atomare Aktionen**

Eine atomare Aktion a beschreibt einen Prozess, dessen Verhalten nicht weiter detailliert wird. Das Verhalten besteht darin, dass a ausgeführt wird und dann (“erfolgreich”) anhält.

Beispiel: Schokoladenautomat

Atomare Aktionen: 1EUR (Einwerfen einer 1-EUR-Münze),
SCHOKO (Ausgabe einer Tafel Schokolade).

- **Konkatenation (sequentielle Komposition)**

p, q Prozesse: $p \cdot q$ beschreibt den Prozess, der sich zuerst so verhält wie p und anschließend so wie q .

Beispiele: (w.o.)

1EUR · SCHOKO :

Automat, der nach Einwurf einer 1-EUR-Münze eine Tafel Schokolade ausgibt (und dann nicht mehr weiterarbeitet).

$(1\text{EUR} \cdot \text{SCHOKO}) \cdot (1\text{EUR} \cdot \text{SCHOKO}) :$

Automat, der in gleicher Weise 2 Kunden (nacheinander) bedient
(und dann nicht mehr weiterarbeitet).

- **Auswahl (alternative Komposition)**

p, q Prozesse: $p + q$ beschreibt den Prozess, der sich entweder so verhält wie p
oder so wie q (nichtdeterministisch).

Beispiele: (w.o., zusätzlich: atomare Aktion 2EUR (Einwerfen einer 2-EUR-Münze))

$1\text{EUR} \cdot \text{SCHOKO} + 2\text{EUR} \cdot (\text{SCHOKO} \cdot \text{SCHOKO})$

(Klammern weggelassen gemäß "Punkt vor Strich"):

Automat, der nach Einwurf einer 1-EUR-Münze eine Tafel Schokolade
ausgibt oder nach Einwurf einer 2-EUR-Münze nacheinander zwei Tafeln
Schokolade ausgibt
(und dann nicht mehr weiterarbeitet).

$(1\text{EUR} \cdot 1\text{EUR} + 2\text{EUR}) \cdot (\text{SCHOKO} \cdot \text{SCHOKO}) :$

Automat, der nach Einwurf von zwei 1-EUR-Münzen oder einer
2-EUR-Münze nacheinander zwei Tafeln Schokolade ausgibt
(und dann nicht mehr weiterarbeitet).

- Sprachmittel für **Parallelität**: siehe später.

Untersuchungsgegenstand

Formal untersucht werden soll die "Gleichheit" $p = q$ von Prozessen p und q .

Anwendungen:

- Vergleichbarkeit von Konzepten (Operationen) und "Programmen".
- "Gleichheit" von **Spezifikation** und **Implementierung** (beide beschrieben als Prozesse),
d.h.: **Korrektheit** der Implementierung (bzgl. der Spezifikation).

Beispiel: (w.o.)

$\text{SCH} \equiv 1\text{EUR} \cdot \text{SCHOKO} + 2\text{EUR} \cdot (\text{SCHOKO} \cdot \text{SCHOKO}) :$

Spezifikation (vom Benutzer gewünschtes Verhalten) des Automaten.

$\text{IMPSCH} :$

"Interne" Funktionsweise (Realisierung) des Automaten.

$\text{SCH} = \text{IMPSCH} :$

Korrektes Funktionieren der Realisierung bzgl. der Spezifikation.

- Nachweis von (ebenfalls in der Prozesssprache darstellbaren) Eigenschaften von Prozessen.

Informelle Bedeutung von $p = q$: p und q zeigen "gleichwertiges Verhalten".

Beispiele:

Plausibel erscheinen: $p + q = q + p,$
 $(p + q) \cdot r = p \cdot r + q \cdot r.$

Fraglich: $p \cdot (q + r) \stackrel{?}{=} p \cdot q + p \cdot r.$

(In verschiedenen Anwendungen: “=” oder “≠” plausibel,
 je nachdem, ob der Zeitpunkt von Entscheidungen zwischen
 Alternativen mitberücksichtigt werden soll oder nicht.)

Festlegung der Gleichwertigkeit des Verhaltens von Prozessen:

Unterschiedliche Sichtweisen (Modellvorstellungen, *Semantiken*) möglich, eventuell abhängig vom beabsichtigten Anwendungsbereich.

2.2 Basissprache und Bisimulations-Semantik

Die Sprache BSP

Gegeben sei: Menge \mathcal{A} (Elemente heißen *atomare Aktionen*, Bezeichnung: a, b, c, \dots).

Definiere: Sprache BSP (“Basissprache für Prozesse”) (über \mathcal{A}).

Induktive Definition der Menge \mathcal{P}_0 der *Prozesse* (über \mathcal{A}):

(BSP1) Jedes Element aus \mathcal{A} ist ein Prozess.

(BSP2) Sind p und q Prozesse, so sind $(p \cdot q)$ und $(p + q)$ Prozesse.

Schreibweisen: “Punkt vor Strich”-Konvention zur Klammerersparnis,
 äußerste Klammern weglassen,
 oft pq statt $p \cdot q$.

Operationelle Semantik von BSP

Zur formalen Bedeutungsdefinition (und Definition der Verhaltensäquivalenz auf \mathcal{P}_0):
operationelle Semantik (formale Festlegung des Ablaufverhaltens von Prozessen).

Definition. Ein *markiertes Transitionssystem* (mTS) (S, Tr, \longrightarrow) besteht aus:

S : einer Menge von *Zuständen*,

Tr : einer Menge von *Transitionen*,

\longrightarrow : einer Relation über $S \times Tr \times S$.

Schreibweise: $s_1 \xrightarrow{t} s_2$ statt $(s_1, t, s_2) \in \longrightarrow$.

(“Transition t überführt Zustand s_1 in Zustand s_2 ”).

Operationelle Semantik von BSP: mTS mit $S = \mathcal{P}_0 \cup \{\checkmark\}$, $Tr = \mathcal{A}$ und \longrightarrow induktiv definiert wie folgt (a beliebig aus \mathcal{A}):

- (T1) $a \xrightarrow{a} \checkmark$ (\checkmark : Zustand “erfolgreicher” Terminierung).
- (T2) Falls $p \xrightarrow{a} p'$, so $p + q \xrightarrow{a} p'$ und $q + p \xrightarrow{a} p'$.
- (T3) Falls $p \xrightarrow{a} p'$ und $p' \not\equiv \checkmark$, so $pq \xrightarrow{a} p'q$.
- (T4) Falls $p \xrightarrow{a} \checkmark$, so $pq \xrightarrow{a} q$.

Induktive Erweiterung von $p \xrightarrow{a} q$ auf $p \xrightarrow{\sigma} q$ für $\sigma \in \mathcal{A}^+$:

- (T⁺1) $p \xrightarrow{\sigma} q$ gegeben durch (T1) – (T4).
- (T⁺2) Falls $p \xrightarrow{\sigma} q$ und $q \xrightarrow{\rho} r$, so $p \xrightarrow{\sigma\rho} r$.

Informelle Bedeutung:

- $p \xrightarrow{\sigma} q$: “ p kann die Aktionsfolge σ ausführen und verhält sich anschließend wie q ”.
- $p \xrightarrow{\sigma} \checkmark$: “ p kann die Aktionsfolge σ ausführen und ist dann (erfolgreich) beendet”.

Beispiele:

Es gilt:

- $(a + b)c \xrightarrow{a} c$,
- $(a + b)c \xrightarrow{b} c$,
- $ac + bc \xrightarrow{a} c$,
- $ac + bc \xrightarrow{b} c$,
- $a(b + c) \xrightarrow{a} b + c$,
- $ab + ac \xrightarrow{a} b$,
- $ab + ac \xrightarrow{a} c$,
- $(a + b)c \xrightarrow{ac} \checkmark$,
- $a(b + c) \xrightarrow{ab} \checkmark$,
- $a(b + c) \xrightarrow{ac} \checkmark$,
- $ab + ac \xrightarrow{ab} \checkmark$,
- $ab + ac \xrightarrow{ac} \checkmark$,
- $(ab)c \xrightarrow{a} bc$,
- $(ab)c \xrightarrow{abc} \checkmark$.

Bisimulationen

Gleichwertiges Verhalten von Prozessen: Sichtweise zunächst “stark differenzierend” (Verhalten gegeben durch ausführbare Aktionsfolgen + “Verzweigungsstruktur”).

Grundlage dazu:

Definition. Eine zweistellige Relation β auf \mathcal{P}_0 heißt **Bisimulation** (auf \mathcal{P}_0), wenn für $p, q \in \mathcal{P}_0$ und $a \in \mathcal{A}$ gilt:

(BS1) Falls $\beta(p, q)$ und $p \xrightarrow{a} p'$ ($p' \neq \checkmark$), so gibt es $q' \in \mathcal{P}_0$ mit $q \xrightarrow{a} q'$ und $\beta(p', q')$.

(BS2) Falls $\beta(p, q)$ und $q \xrightarrow{a} q'$ ($q' \neq \checkmark$), so gibt es $p' \in \mathcal{P}_0$ mit $p \xrightarrow{a} p'$ und $\beta(p', q')$.

(BS3) Falls $\beta(p, q)$, so: $p \xrightarrow{a} \checkmark$ genau dann, wenn $q \xrightarrow{a} \checkmark$.

$p, q \in \mathcal{P}_0$ heißen **bisimilar** (in Zeichen: $p \Leftrightarrow q$), wenn es eine Bisimulation β auf \mathcal{P}_0 gibt, so dass $\beta(p, q)$ gilt.

Beispiele:

1. $\beta = \{((a + b)c, ac + bc), (c, c)\}$ (geschrieben in Mengenschreibweise) ist Bisimulation.

2. $a(b + c)$ und $ab + ac$ sind nicht bisimilar.

Satz 2.2.1 \Leftrightarrow ist eine Äquivalenzrelation.

Satz 2.2.2 Falls $p_1 \Leftrightarrow q_1$ und $p_2 \Leftrightarrow q_2$, so gilt $p_1 p_2 \Leftrightarrow q_1 q_2$ und $p_1 + p_2 \Leftrightarrow q_1 + q_2$.

Verhaltensäquivalenz von Prozessen

Definition. Zwei Prozesse p und q mit $p \Leftrightarrow q$ heißen **verhaltensäquivalent** (kurz: **gleich**) (in der **Bisimulations-Semantik**). Schreibweise: $p = q$.

Satz 2.2.3 Für alle $p, q, r \in \mathcal{P}_0$ gilt:

- $p + q = q + p$.
- $(p + q) + r = p + (q + r)$.
- $p + p = p$.
- $(p + q)r = pr + qr$.
- $(pq)r = p(qr)$.

Bemerkungen:

1. Es gilt i.a. nicht: $p(q + r) = pq + pr$.

2. Wegen Satz 2.2.3 b) und e): Im Folgenden häufig klammerfreie Schreibweisen, z.B.:

$$\begin{array}{l} p + q + r \quad \text{statt} \quad (p + q) + r \quad \text{oder} \quad p + (q + r), \\ pqr \quad \quad \quad \text{statt} \quad (pq)r \quad \text{oder} \quad p(qr) \end{array}$$

(und ebenso mit mehr als 3 Prozessen).

2.3 Axiomatisierung

Rechnen mit Gleichheitsbeziehungen von Prozessen

Folgerung aus den Sätzen 2.2.1 und 2.2.2:

Die Gleichheitsrelation $=$ ist eine **Kongruenzrelation** auf \mathcal{P}_0 (bzgl. der Operationen \cdot und $+$), man kann also so mit ihr “rechnen”, wie man das “üblicherweise” für Gleichheitsrelationen gewohnt ist.

Damit: Zwei Möglichkeiten zum Nachweis einer Gleichheitsbeziehung $p = q$:

- “Semantisch”: Zeige $p \Leftrightarrow q$.
- (Wünschenswert:) Durch “Rechnen”, ohne Rückgriff auf die semantische Definition.

Systematisierung:

- (1) Einige Gleichheitsbeziehungen werden als **Axiome** ausgezeichnet.
- (2) Weitere Gleichheitsbeziehungen $p = q$ können aus den Axiomen mit Hilfe der “üblichen Regeln” für die Gleichheitsrelation (**Gleichheitslogik**) (und eventuell weiteren Regeln) “errechnet” werden. $p = q$ heißt dann in der somit definierten (**axiomatischen**) **Theorie (Axiomatisierung)** Σ **herleitbar**. (Schreibweise: $\Sigma \vdash p = q$.)

Die Theorie Σ_{BSP}

Geeignete Axiome für die Gleichheit von Prozessen in BSP (Theorie Σ_{BSP}):

- (P1) $p + q = q + p$.
 (P2) $(p + q) + r = p + (q + r)$.
 (P3) $p + p = p$.
 (P4) $(p + q)r = pr + qr$.
 (P5) $(pq)r = p(qr)$.

Beispiel:

Es gilt $\Sigma_{\text{BSP}} \vdash ((p + q) + p)p = pp + qp$ (für beliebige $p, q \in \mathcal{P}_0$) gemäß der “Rechnung” (**Herleitung**)

$$\begin{aligned}
 ((p + q) + p)p &= ((p + (p + q))p && \text{(P1)} \\
 &= ((p + p) + q)p && \text{(P2)} \\
 &= (p + q)p && \text{(P3)} \\
 &= pp + qp && \text{(P4)}
 \end{aligned}$$

Satz 2.3.1 (Korrektheit von Σ_{BSP})

Für $p, q \in \mathcal{P}_0$ gilt: Falls $\Sigma_{\text{BSP}} \vdash p = q$, so $p = q$.

Präfix-Konkatenation

Induktive Definition der Prozessmenge $\mathcal{P}_{\text{prä}}$:

1. Jedes Element von \mathcal{A} ist in $\mathcal{P}_{\text{prä}}$.
2. Falls $a \in \mathcal{A}$ und $q \in \mathcal{P}_{\text{prä}}$, so ist $(a \cdot q) \in \mathcal{P}_{\text{prä}}$ (Schreibweise wieder: aq) (**Präfix-Konkatenation**).
3. Falls $p, q \in \mathcal{P}_{\text{prä}}$, so ist $(p + q) \in \mathcal{P}_{\text{prä}}$.

Bemerkung:

Aus der Definition folgt: Jedes Element aus $\mathcal{P}_{\text{prä}}$ hat die Gestalt

$$p_1 + p_2 + \dots + p_n \quad (n \geq 1)$$

mit $p_i \equiv a_i q_i$, $a_i \in \mathcal{A}$, $q_i \in \mathcal{P}_{\text{prä}}$

oder $p_i \equiv b_i$, $b_i \in \mathcal{A}$.

($1 \leq i \leq n$).

Vollständigkeit von Σ_{BSP}

Satz 2.3.2 Zu jedem $p \in \mathcal{P}_0$ gibt es ein $p' \in \mathcal{P}_{\text{prä}}$ mit $p = p'$, und $p = p'$ ist in Σ_{BSP} herleitbar.

Lemma 2.3.3 Sei $p \in \mathcal{P}_{\text{prä}}$. Dann gilt:

$$\text{a) } p \xrightarrow{a} q, q \neq \checkmark \Rightarrow q \in \mathcal{P}_{\text{prä}}$$

und

$$p = aq \text{ oder es gibt } r \in \mathcal{P}_{\text{prä}} \text{ mit } p = aq + r.$$

$$\text{b) } p \xrightarrow{a} \checkmark \Rightarrow p = a \text{ oder es gibt } r \in \mathcal{P}_{\text{prä}} \text{ mit } p = a + r.$$

(Und diese Gleichungen sind in Σ_{BSP} herleitbar.)

Satz 2.3.4 (Vollständigkeit von Σ_{BSP})

Für $p, q \in \mathcal{P}_0$ gilt: Falls $p = q$, so $\Sigma_{\text{BSP}} \vdash p = q$.

Kapitel 3

Kommunizierende Prozesse

3.1 Parallelität

Die Sprache SPP

Weitere Sprachmittel zur Beschreibung von Prozessen (jetzt für Parallelität):

- **Paralleloperator (free merge operator)**
 p, q Prozesse: $p \parallel q$ Prozess; informelle Bedeutung: Parallelausführung von p und q (modelliert als interleaving der atomaren Aktionen von p und q).
- **Links-Verzahnung** (“Hilfs”-Operator, benötigt zur Axiomatisierung)
 p, q Prozesse: $p \ll q$ Prozess; informelle Bedeutung: wie $p \parallel q$, aber die erste ausgeführte Aktion stammt von p .

Sprache SPP (“Sprache für parallele Prozesse”):

Erweiterung der Sprache BSP durch zusätzliche Syntaxregel (zu (BSP1),(BSP2)):

(SPP) Sind p und q Prozesse, so sind $(p \parallel q)$ und $(p \ll q)$ Prozesse.

\mathcal{P}_1 : Menge aller Prozesse von SPP (über vorgegebenem \mathcal{A} , wie bisher).

Bindungsstärke der Operatoren: \cdot “vor” \parallel, \ll “vor” $+$.

Operationelle Semantik von SPP

Die operationelle Semantik von SPP ergibt sich durch Erweiterung von (T1)-(T4) um:

- (T5) Falls $p \xrightarrow{a} p'$ und $p' \not\equiv \checkmark$, so $p \parallel q \xrightarrow{a} p' \parallel q$ und $q \parallel p \xrightarrow{a} q \parallel p'$.
- (T6) Falls $p \xrightarrow{a} \checkmark$, so $p \parallel q \xrightarrow{a} q$ und $q \parallel p \xrightarrow{a} q$.
- (T7) Falls $p \xrightarrow{a} p'$ und $p' \not\equiv \checkmark$, so $p \ll q \xrightarrow{a} p' \parallel q$.
- (T8) Falls $p \xrightarrow{a} \checkmark$, so $p \ll q \xrightarrow{a} q$.

Bisimulationen, $p \Leftrightarrow q$, Gleichheit $p = q$ auf \mathcal{P}_1 : wie bei BSP (d.h.: $p = q \iff p \Leftrightarrow q$).

Satz 3.1.1 Die Sätze 2.2.1 und 2.2.2 sowie die Axiome (P1) – (P5) von Σ_{BSP} gelten auch für Prozesse aus \mathcal{P}_1 .

Satz 3.1.2 Seien $p_1, p_2, q_1, q_2 \in \mathcal{P}_1$. Falls $p_1 = q_1$ und $p_2 = q_2$, so gilt $p_1 \parallel p_2 = q_1 \parallel q_2$ und $p_1 \ll p_2 = q_1 \ll q_2$.

Folgerung: \equiv ist Kongruenzrelation auch auf \mathcal{P}_1 .

Axiomatisierung

Satz 3.1.3 Für alle $p, q, r \in \mathcal{P}_1$, $a \in \mathcal{A}$ gilt:

- a) $p \parallel q = p \parallel q + q \parallel p$.
- b) $a \parallel q = aq$.
- c) $ap \parallel q = a(p \parallel q)$.
- d) $(p + q) \parallel r = p \parallel r + q \parallel r$.

Theorie Σ_{SPP}

(Axiome für Gleichheit von Prozessen in SPP, Korrektheit folgt aus 3.1.1, 3.1.2, 3.1.3.):

Alle Axiome (P1)-(P5) von Σ_{BSP} sowie zusätzlich ($a \in \mathcal{A}$):

- (P6) $p \parallel q = p \parallel q + q \parallel p$.
- (P7) $a \parallel q = aq$.
- (P8) $ap \parallel q = a(p \parallel q)$.
- (P9) $(p + q) \parallel r = p \parallel r + q \parallel r$.

Satz 3.1.4 (Eliminationssatz)

Zu jedem Prozess $p \in \mathcal{P}_1$ gibt es einen Prozess $p' \in \mathcal{P}_0$ mit $p = p'$
(und $p = p'$ ist in Σ_{SPP} herleitbar).

Satz 3.1.5 (Vollständigkeit von Σ_{SPP})

Für $p, q \in \mathcal{P}_1$ gilt: Falls $p = q$, so $\Sigma_{\text{SPP}} \vdash p = q$.

Einige weitere Gleichheitsbeziehungen

Satz 3.1.6 Für $p, q, r \in \mathcal{P}_1$ gilt:

- a) $p \parallel q = q \parallel p$.
- b) $(p \parallel q) \parallel r = p \parallel (q \parallel r)$.
- c) $(p \parallel q) \parallel r = p \parallel (q \parallel r)$.

Schreibweise: $p_1 \parallel p_2 \parallel p_3$ statt $(p_1 \parallel p_2) \parallel p_3$ oder $p_1 \parallel (p_2 \parallel p_3)$
(ebenso für mehr als 3 Prozesse).

Satz 3.1.7 Für $p_1, \dots, p_n \in \mathcal{P}_1$, $n \geq 2$, gilt:

$$p_1 \parallel p_2 \parallel \dots \parallel p_n = p_1 \parallel (p_2 \parallel \dots \parallel p_n) + \dots + p_i \parallel (p_1 \parallel \dots \parallel p_{i-1} \parallel p_{i+1} \parallel \dots \parallel p_n) + \dots + p_n \parallel (p_1 \parallel \dots \parallel p_{n-1}).$$

3.2 Kommunikation

Kommunikations- und Übertragungsaktionen

Operator \parallel (bis jetzt): Parallelität ohne Kommunikation zwischen den beteiligten Prozessen.

Erweiterung: Synchrone Kommunikation

(über einen Kanal zwischen je zwei Prozessen, vgl. Abschnitt 1.2).

Dazu:

- \mathcal{A} enthalte drei ausgezeichnete disjunkte Teilmengen:

\mathcal{C} ($a \in \mathcal{C}$: “Senden (einer bestimmten Nachricht) auf Kanal a ”);

$\bar{\mathcal{C}} = \{\bar{a} \mid a \in \mathcal{C}\}$ (\bar{a} : “Empfangen auf Kanal a ”);

$\mathcal{C}_\gamma = \{a_\gamma \mid a \in \mathcal{C}\}$ (a_γ : “Synchrone Ausführung von a und \bar{a} ”).

$c \in \mathcal{C} \cup \bar{\mathcal{C}}$ heißt **Kommunikationsaktion**.

$a, b \in \mathcal{A}$ heißen **kommunizierend**, wenn $a \in \mathcal{C}, b \equiv \bar{a}$ oder $b \in \mathcal{C}, a \equiv \bar{b}$

(in beiden Fällen: Bezeichnung $\gamma(a, b)$ für a_γ bzw. b_γ).

$c \in \mathcal{C}_\gamma$ heißt **Übertragung(saktion)**.

- **Kommunikationsoperator (communication merge operator)** (zur Anpassung von \parallel)
 p, q Prozesse: $p|q$ Prozess; informelle Bedeutung: Verzahnung von p und q ,
wobei die erste Aktion eine (passende) Übertragung ist.

Die Sprache SKP

Sprache SKP (“Sprache für kommunizierende Prozesse”):

Erweiterung der Sprache SPP durch zusätzliche Syntaxregel (zu den Regeln von SPP; \mathcal{A} wie oben):

(SKP) Sind p und q Prozesse, so ist $(p|q)$ ein Prozess.

\mathcal{P}_2 : Menge aller Prozesse von SKP.

Bindungsstärke der Operatoren: \cdot “vor” $\parallel, \underline{\parallel}, |$ “vor” $+$.

Operationelle Semantik von SKP

Die operationelle Semantik von SKP ergibt sich durch Erweiterung von (T1)-(T8) um:

(T9) Falls $p \xrightarrow{a} p', q \xrightarrow{b} q', p' \not\equiv \checkmark, q' \not\equiv \checkmark, a, b$ kommunizierend,
so $p \parallel q \xrightarrow{\gamma(a,b)} p' \parallel q'$ und $p|q \xrightarrow{\gamma(a,b)} p'|q'$.

(T10) Falls $p \xrightarrow{a} p', q \xrightarrow{b} \checkmark, p' \not\equiv \checkmark, a, b$ kommunizierend,
so $p \parallel q \xrightarrow{\gamma(a,b)} p', p|q \xrightarrow{\gamma(a,b)} p', q \parallel p \xrightarrow{\gamma(a,b)} p'$ und $q|p \xrightarrow{\gamma(a,b)} p'$.

(T11) Falls $p \xrightarrow{a} \checkmark, q \xrightarrow{b} \checkmark, a, b$ kommunizierend,
so $p \parallel q \xrightarrow{\gamma(a,b)} \checkmark$ und $p|q \xrightarrow{\gamma(a,b)} \checkmark$.

Bisimulationen, $p \Leftrightarrow q$, Gleichheit $p = q$ auf \mathcal{P}_2 : wie bei SPP (d.h.: $p = q \iff p \Leftrightarrow q$).

Beispiele:

Es gilt:

$$ab|\bar{a}c \xrightarrow{a_\gamma} b||c,$$

$$ab|\bar{a} \xrightarrow{a_\gamma} b,$$

$$a|\bar{a} \xrightarrow{a_\gamma} \checkmark.$$

Satz 3.2.1 \Leftrightarrow ist Kongruenzrelation auf \mathcal{P}_2 , und die Axiome (P1) – (P5) und (P7) – (P9) von Σ_{SPP} gelten auch für Prozesse aus \mathcal{P}_2 .

Axiomatisierung

Satz 3.2.2 Für alle $p, q, r \in \mathcal{P}_2$, $a, b \in \mathcal{A}$ gilt:

- a) $p||q = p||q + q||p + p|q$.
- b) $a|b = \gamma(a, b)$, falls a und b kommunizierend.
- c) $ap|b = (a|b)p$.
- d) $a|bp = (a|b)p$.
- e) $ap|bq = (a|b) \cdot (p||q)$.
- f) $(p + q)|r = p|r + q|r$.
- g) $p|(q + r) = p|q + p|r$.

Theorie Σ_{SKP}

(Axiome für Gleichheit von Prozessen in SKP, Korrektheit folgt aus 3.2.1, 3.2.2):

Alle Axiome (P1)-(P5) und (P7)-(P9) von Σ_{SPP} sowie zusätzlich ($a, b \in \mathcal{A}$):

$$(P6_c) \quad p||q = p||q + q||p + p|q.$$

$$(P10) \quad a|b = \gamma(a, b), \quad \text{falls } a \text{ und } b \text{ kommunizierend.}$$

$$(P11) \quad ap|b = (a|b)p.$$

$$(P12) \quad a|bp = (a|b)p.$$

$$(P13) \quad ap|bq = (a|b) \cdot (p||q).$$

$$(P14) \quad (p + q)|r = p|r + q|r.$$

$$(P15) \quad p|(q + r) = p|q + p|r.$$

Bemerkungen:

1. Der Eliminationssatz gilt in SKP (i. Allg.) nicht.
2. Σ_{SKP} ist nicht vollständig.

3.3 Verklemmung und Restriktion

Weitere wünschenswerte Sprachelemente

- $a|b$ ($a, b \in \mathcal{A}$, nicht kommunizierend): Prozess, der nicht “erfolgreich” beendet ist, aber auch nicht fortfahren kann (Verklemmung).

Zur Beschreibung: \mathcal{A} enthalte ein ausgezeichnetes Element δ (**Verklemmung, deadlock**) (mit: $\delta \notin \mathcal{C} \cup \bar{\mathcal{C}} \cup \mathcal{C}_\gamma$).

Intention:

$$a|b = \delta, \text{ falls } a, b \text{ nicht kommunizierend, und } \delta \not\rightarrow \dots$$

- $a \in \mathcal{C}$ und $\bar{a} \in \bar{\mathcal{C}}$ sind auch “einzeln ausführbar”.

Zur Beschreibung, dass dies nicht möglich ist: Neuer Operator ∂ .

Intention:

$$\partial_{\{a, \bar{a}\}}(ab || c\bar{a}) \xrightarrow{ca_\gamma b} \checkmark \quad (\text{und sonst keine “Abläufe” nach } \checkmark).$$

Allgemein:

- **Restriktion (encapsulation)**
 p Prozess, $\mathcal{R} \subseteq \mathcal{A} \setminus \{\delta\}$: $\partial_{\mathcal{R}}(p)$ Prozess; informelle Bedeutung: In p sind Aktionen aus \mathcal{R} nicht (“einzeln”) ausführbar.

Die Sprache ACP

Sprache ACP (“Algebra of Communicating Processes”):

Erweiterung der Sprache SKP durch zusätzliche Syntaxregel (zu den Regeln von SKP; \mathcal{A} wie oben):

(ACP) Ist p ein Prozess, $\mathcal{R} \subseteq \mathcal{A} \setminus \{\delta\}$, so ist $\partial_{\mathcal{R}}(p)$ ein Prozess.

\mathcal{P}_e : Menge aller Prozesse von ACP.

Operationelle Semantik von ACP

Die operationelle Semantik von SKP ergibt sich durch Einschränkung auf $a \in \mathcal{A} \setminus \{\delta\}$ in (T1) und Erweiterung von (T1)-(T11) um:

(T12) Falls $p \xrightarrow{a} p'$, $p' \not\equiv \checkmark$ und $a \notin \mathcal{R}$, so $\partial_{\mathcal{R}}(p) \xrightarrow{a} \partial_{\mathcal{R}}(p')$.

(T13) Falls $p \xrightarrow{a} \checkmark$ und $a \notin \mathcal{R}$, so $\partial_{\mathcal{R}}(p) \xrightarrow{a} \checkmark$.

Bisimulationen, $p \Leftrightarrow q$, Gleichheit $p = q$ auf \mathcal{P}_e : wie bei SKP (d.h.: $p = q \iff p \Leftrightarrow q$).

Satz 3.3.1 \Leftrightarrow ist Kongruenzrelation auf \mathcal{P}_e , und die Axiome (P1) – (P5), (P6_c), (P7) – (P15) von Σ_{SKP} gelten auch für Prozesse aus \mathcal{P}_e .

Axiomatisierung

Satz 3.3.2 Für alle $p, q \in \mathcal{P}_e$, $a \in \mathcal{A}$, $\mathcal{R} \subseteq \mathcal{A} \setminus \{\delta\}$ gilt:

- a) $p + \delta = p$.
- b) $\delta p = \delta$.
- c) $a|b = \delta$, falls a und b nicht kommunizierend.
- d) $\partial_{\mathcal{R}}(a) = a$ für $a \notin \mathcal{R}$.
- e) $\partial_{\mathcal{R}}(a) = \delta$ für $a \in \mathcal{R}$.
- f) $\partial_{\mathcal{R}}(p + q) = \partial_{\mathcal{R}}(p) + \partial_{\mathcal{R}}(q)$.
- g) $\partial_{\mathcal{R}}(pq) = \partial_{\mathcal{R}}(p) \cdot \partial_{\mathcal{R}}(q)$.

Theorie Σ_{ACP}

(Axiome für Gleichheit von Prozessen in ACP, Korrektheit folgt aus 3.3.1, 3.3.2):

Alle Axiome (P1)-(P5), (P6_c), (P7)-(P15) von Σ_{SKP} sowie zusätzlich ($a \in \mathcal{A}$, $\mathcal{R} \subseteq \mathcal{A} \setminus \{\delta\}$):

- (P16) $p + \delta = p$.
- (P17) $\delta p = \delta$.
- (P18) $a|b = \delta$, falls a und b nicht kommunizierend.
- (P19) $\partial_{\mathcal{R}}(a) = a$ für $a \notin \mathcal{R}$.
- (P20) $\partial_{\mathcal{R}}(a) = \delta$ für $a \in \mathcal{R}$.
- (P21) $\partial_{\mathcal{R}}(p + q) = \partial_{\mathcal{R}}(p) + \partial_{\mathcal{R}}(q)$.
- (P22) $\partial_{\mathcal{R}}(pq) = \partial_{\mathcal{R}}(p) \cdot \partial_{\mathcal{R}}(q)$.

Satz 3.3.3 (Eliminationssatz)

Zu jedem Prozess $p \in \mathcal{P}_e$ gibt es einen Prozess $p' \in \mathcal{P}_0$ mit $p = p'$ (und $p = p'$ ist in Σ_{ACP} herleitbar).

Satz 3.3.4 (Vollständigkeit von Σ_{ACP})

Für $p, q \in \mathcal{P}_e$ gilt: Falls $p = q$, so $\Sigma_{ACP} \vdash p = q$.

Einige weitere Gleichheitsbeziehungen

Satz 3.3.5 Für $p, q, r \in \mathcal{P}_e$ gilt:

- a) $p|q = q|p$.
- b) $p||q = q||p$.
- c) $p|\delta = \delta$.
- d) $p||\delta = p||\delta = p\delta$.
- e) $\delta||p = \delta$.

- f) $(p|q)|r = p|(q|r) = \delta$.
g) $(p \parallel q) \parallel r = p \parallel (q \parallel r)$.
h) $p|(q \parallel r) = (p|q) \parallel r$.
i) $(p \parallel q) \parallel r = p \parallel (q \parallel r)$.

Klammerfreie Schreibweise (wie in SPP): $p_1 \parallel p_2 \parallel \dots \parallel p_n$.

Satz 3.1.7 gilt in ACP im Allgemeinen nicht, statt dessen:

Satz 3.3.6 (Expansionssatz)

Für $p_1, \dots, p_n \in \mathcal{P}_e$, $n \geq 3$ gilt:

$$\begin{aligned}
p_1 \parallel p_2 \parallel \dots \parallel p_n &= p_1 \parallel (p_2 \parallel \dots \parallel p_n) \\
&+ \dots + \\
&p_i \parallel (p_1 \parallel \dots \parallel p_{i-1} \parallel p_{i+1} \parallel \dots \parallel p_n) \\
&+ \dots + \\
&p_n \parallel (p_1 \parallel \dots \parallel p_{n-1}) + \\
&(p_1|p_2) \parallel (p_3 \parallel \dots \parallel p_n) \\
&+ \dots + \\
&(p_i|p_j) \parallel (p_1 \parallel \dots \parallel p_{i-1} \parallel p_{i+1} \parallel \dots \parallel p_{j-1} \parallel p_{j+1} \parallel \dots \parallel p_n) \quad [i < j] \\
&+ \dots + \\
&(p_{n-1}|p_n) \parallel (p_1 \parallel \dots \parallel p_{n-2}).
\end{aligned}$$

Beispiel

System mit

Prozess P_1 : Sendet Bit 0 oder 1,

Prozess P_2 : Empfängt dieses Bit (auf dem Verbindungskanal).

Spezifikation von P_1 : $\text{sende}(0) + \text{sende}(1)$.

Spezifikation von P_2 : $\text{empfang}(0) + \text{empfang}(1)$.

Dabei: $\text{empfang}(0) \equiv \overline{\text{sende}(0)}$,
 $\text{empfang}(1) \equiv \overline{\text{sende}(1)}$.

Sei: $\text{übertr}(0) \equiv \gamma(\text{sende}(0), \text{empfang}(0))$,
 $\text{übertr}(1) \equiv \gamma(\text{sende}(1), \text{empfang}(1))$.

Spezifikation des Gesamtsystems:

$$P \equiv \partial_{\{\text{sende}(0), \text{sende}(1), \text{empfang}(0), \text{empfang}(1)\}}(P_1 \parallel P_2).$$

Es gilt:

$$P = \text{übertr}(0) + \text{übertr}(1).$$

Allgemeine Form von Spezifikationen

(Systeme von) Prozesse(n) werden in ACP typischerweise in der Form

$$\partial_{\mathcal{R}}(P_1 \parallel P_2 \parallel \dots \parallel P_n)$$

mit

$$\mathcal{R} = \{a, \bar{a} \mid \text{es gibt } P_i, \text{ der } a \text{ enthält und } P_j, j \neq i, \text{ der } \bar{a} \text{ enthält}\}$$

beschrieben.

Wirkung von $\partial_{\mathcal{R}}$: Aktionen a und \bar{a} in den P_k , die miteinander kommunizieren sollen, werden “eingekapselt”, ihre jeweilige “alleinige” Wirkung auf die “Außenwelt” wird ausgeschlossen.

3.4 Denotationelle Semantik

Prozessgraphen

Definition. Ein *verwurzelter gerichteter Multigraph* (vgMG) $G = (V, E, w)$ ist gegeben durch

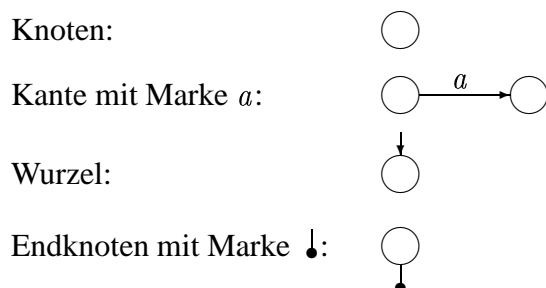
- eine Menge V von **Knoten**,
- eine Multimenge E von geordneten Paaren $(v_1, v_2) \in V \times V$ (**Kanten**),
- ein ausgezeichnetes Element $w \in V$ (**Wurzel**).

$v \in V$ heißt **Endknoten** von G , wenn es keine Kante $(v, \dots) \in E$ gibt.

Definition. Ein *Prozessgraph* ist ein vgMG, in dem jeder Kante ein Element aus \mathcal{A} (als **Marke**) zugeordnet ist (gleichen Kanten nicht notwendig gleiche Elemente) und ein Teil der Endknoten (einschließlich keiner oder alle) mit der Marke \downarrow markiert ist.

Schreibweisen: $v_1 \xrightarrow{a} v_2$ für: $(v_1, v_2) \in E$, markiert mit a ,
 $v \downarrow$ für: Endknoten v ist mit \downarrow markiert.

Graphische Darstellung von Prozessgraphen:




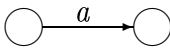

Definition. $G = (V_G, E_G, w_G)$ und $H = (V_H, E_H, w_H)$ seien Prozessgraphen. Das *kartesische Produkt* $G \times H$ von G und H ist ein Prozessgraph $(V_{G \times H}, E_{G \times H}, w_{G \times H})$ mit:

- $V_{G \times H} = V_G \times V_H$.
- $(v_1, v_2) \downarrow$ für $(v_1, v_2) \in V_{G \times H} \iff v_1 \downarrow$ oder $v_2 \downarrow$.
- $(v_1, v_2) \xrightarrow{a} (v'_1, v_2)$ in $E_{G \times H} \iff v_1 \xrightarrow{a} v'_1$ in E_G .
- $(v_1, v_2) \xrightarrow{a} (v_1, v'_2)$ in $E_{G \times H} \iff v_2 \xrightarrow{a} v'_2$ in E_H .
- $w_{G \times H} = (w_G, w_H)$.


Der Prozessgraph eines ACP-Prozesses

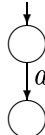
Intention ($p \in \mathcal{P}_e$): $p \mapsto^{\mathcal{D}}$ Prozessgraph $\mathcal{D}(p) = \llbracket p \rrbracket$ (*denotationelle Semantik*).

Dabei:

-  $\hat{=}$ Anfangszustand,
-  $\hat{=}$ möglicher Zustandsübergang,
-  $\hat{=}$ Verklemmungszustand,
- Endknoten ohne \downarrow $\hat{=}$ Endzustand \neq Verklemmung.

Induktive Definition von $\llbracket p \rrbracket$ für $p \in \mathcal{P}_e$:

(D1) $\llbracket \delta \rrbracket =$ 

(D2) $\llbracket a \rrbracket =$  für $a \in \mathcal{A} \setminus \{\delta\}$.

(D3) $p, q \in \mathcal{P}_e$; Konstruktion von $\llbracket p + q \rrbracket$ aus $\llbracket p \rrbracket$ und $\llbracket q \rrbracket$:

- Identifiziere die beiden Wurzeln von $\llbracket p \rrbracket$ und $\llbracket q \rrbracket$.
- Markiere die jetzt gemeinsame Wurzel mit \downarrow genau dann, wenn beide Wurzeln von $\llbracket p \rrbracket$ und $\llbracket q \rrbracket$ mit \downarrow markiert sind.

(D4) $p, q \in \mathcal{P}_e$; Konstruktion von $\llbracket pq \rrbracket$ aus $\llbracket p \rrbracket$ und $\llbracket q \rrbracket$:

- Identifiziere Endknoten von $\llbracket p \rrbracket$, die nicht mit \downarrow markiert sind, mit der Wurzel w von $\llbracket q \rrbracket$ (mit Übernahme aller Kanten, die auf die Endknoten führen, und der eventuell vorhandenen Marke \downarrow von w).
- Hat $\llbracket p \rrbracket$ keine derartigen Endknoten, so ist $\llbracket p \rrbracket$ das Ergebnis.

(D5) $p, q \in \mathcal{P}_e$; Konstruktion von $\llbracket p \parallel q \rrbracket$ aus $\llbracket p \rrbracket$ und $\llbracket q \rrbracket$:

- Bilde $\llbracket p \rrbracket \times \llbracket q \rrbracket$ ($=: G$).
- Sind $(v_1, v_2) \xrightarrow{a} (v'_1, v_2)$ und $(v_1, v_2) \xrightarrow{b} (v_1, v'_2)$ Kanten in G , a und b kommutierend, so füge in G eine Kante $(v_1, v_2) \xrightarrow{\gamma^{(a,b)}} (v'_1, v'_2)$ ein.

(D6) $p, q \in \mathcal{P}_e$; Konstruktion von $\llbracket p \parallel q \rrbracket$ aus $\llbracket p \rrbracket$ und $\llbracket q \rrbracket$:

- Bilde $\llbracket p \parallel q \rrbracket$ ($=: G$).
- Ist (v_1, v_2) die Wurzel von G , so entferne alle Kanten $(v_1, v_2) \xrightarrow{a} (v_1, v'_2)$ und $(v_1, v_2) \xrightarrow{c} (v'_1, v'_2)$ mit $c \in C_\gamma$ sowie alle Teile von G , die dadurch von (v_1, v_2) aus auf keiner Kantenfolge erreichbar sind. Wird (v_1, v_2) dadurch Endknoten, so markiere (v_1, v_2) mit \downarrow .

(D7) $p, q \in \mathcal{P}_e$; Konstruktion von $\llbracket p | q \rrbracket$ aus $\llbracket p \rrbracket$ und $\llbracket q \rrbracket$:

- Bilde $\llbracket p | q \rrbracket$ ($=: G$, Wurzel w).
- Entferne in G alle Kanten $w \xrightarrow{a} v$ mit $a \notin C_\gamma$ sowie alle dadurch nicht mehr erreichbaren Teile von G . Wird w dadurch Endknoten, so markiere w mit \downarrow .

(D8) $p, q \in \mathcal{P}_e$; Konstruktion von $\llbracket \partial_{\mathcal{R}}(p) \rrbracket$ aus $\llbracket p \rrbracket$:

- Entferne alle Kanten $v \xrightarrow{a} v'$ mit $a \in \mathcal{R}$ und alle dadurch unerreichbar gewordenen Teile von $\llbracket p \rrbracket$. Wird v dadurch Endknoten, so markiere v mit \downarrow .

Bisimulationen auf Prozessgraphen

Definition. $G = (V_G, E_G, w_G)$ und $H = (V_H, E_H, w_H)$ seien Prozessgraphen. Eine binäre Relation ϱ auf $V_G \times V_H$ heißt **Bisimulation** (zwischen G und H), wenn gilt:

(BG1) $\varrho(w_G, w_H)$.

(BG2) Falls $s, s' \in V_G, t \in V_H, s \xrightarrow{a} s', \varrho(s, t)$, so gibt es $t \xrightarrow{a} t'$ in H mit $\varrho(s', t')$.

(BG3) Falls $t, t' \in V_H, s \in V_G, t \xrightarrow{a} t', \varrho(s, t)$, so gibt es $s \xrightarrow{a} s'$ in G mit $\varrho(s', t')$.

(BG4) Falls $s \in V_G, t \in V_H, s, t$ Endknoten, $\varrho(s, t)$, so: $s \downarrow \iff t \downarrow$.

G und H heißen **bisimilar** ($G \simeq H$), wenn es eine Bisimulation ϱ zwischen G und H gibt.

Satz 3.4.1 Für $p, q \in \mathcal{P}_e$ gilt: $p = q \iff \llbracket p \rrbracket \simeq \llbracket q \rrbracket$.

Kapitel 4

Rekursion

4.1 Rekursive Spezifikationen

Das Rekursionskonzept

Ziel: Beschreibung von Prozessen, die (auch) “endlos laufen” können.

Beispiel:

Schokoladenautomat (vgl. Abschnitt 2.1) SCH, der den durch 1EUR · SCHOKO beschriebenen Vorgang “endlos” wiederholen kann:

$$\begin{aligned} \text{SCH} &= (1\text{EUR} \cdot \text{SCHOKO})^\omega \\ &= (1\text{EUR} \cdot \text{SCHOKO}) \cdot (1\text{EUR} \cdot \text{SCHOKO}) \cdot (1\text{EUR} \cdot \text{SCHOKO}) \cdot \dots \quad (\text{ad infinitum}). \end{aligned}$$

Formale Beschreibung von SCH durch Rekursion:

$$\text{SCH} = (1\text{EUR} \cdot \text{SCHOKO}) \cdot \text{SCH}$$

(Automat, der nach Einwurf einer 1-EUR-Münze eine Tafel Schokolade ausgibt und sich anschließend erneut so verhält wie SCH).

Anders ausgedrückt: SCH ist “Lösung” (für “Variable” x) der (“Rekursions”-) Gleichung

$$x = (1\text{EUR} \cdot \text{SCHOKO}) \cdot x.$$

Allgemein: Weiteres Konzept zur Beschreibung von Prozessen:

- **Rekursion**
Beschreibung von Prozessen durch Rekursionsgleichungen.

Prozessterme

Gegeben sei: \mathcal{A} wie bisher,
(abzählbar unendliche) Menge \mathcal{X} von (**Prozess-**) **Variablen**.

Induktive Definition der **Prozessterme** t (über \mathcal{A} und \mathcal{X}) und der Mengen $FV(t) \subseteq \mathcal{X}$ ihrer **freien Variablen**:

- (PT1) Jedes $a \in \mathcal{A}$ ist ein Prozessterm mit $FV(a) = \emptyset$.
- (PT2) Jedes $x \in \mathcal{X}$ ist ein Prozessterm mit $FV(x) = \{x\}$.

(PT3) Sind t und s Prozessterme, $\mathcal{R} \subseteq \mathcal{A} \setminus \{\delta\}$,

so sind $(t \cdot s)$, $(t + s)$, $(t \parallel s)$, $(t \ll s)$, $(t|s)$ und $\partial_{\mathcal{R}}(t)$ Prozessterme mit

$$\begin{aligned} FV(t \cdot s) &= FV(t + s) = FV(t \parallel s) = FV(t \ll s) = FV(t|s) = FV(t) \cup FV(s), \\ FV(\partial_{\mathcal{R}}(t)) &= FV(t). \end{aligned}$$

(PT4) (siehe später).

Schreibweisen: wie bisher, außerdem:

t, r_1, \dots, r_m Prozessterme : $t^{[y_1/r_1, \dots, y_m/r_m]}$ für den Prozessterm, der aus t entsteht, wenn jedes y_j (falls es in t vorkommt) durch r_j ersetzt wird ($j = 1, \dots, m$).

Rechnen mit Prozesstermen

Herleitung von Prozesstermgleichungen $t = t'$: wie bisher (in Σ_{ACP}), wobei alle Axiome und Regeln auf Prozessterme angewendet werden (und dabei Variablen wie Prozesse behandelt werden).

Beispiel: (vgl. Abschnitt 2.3)

$$\begin{aligned} ((x + q) + x)x &= (x + (x + q))x \\ &= ((x + x) + q)x \\ &= (x + q)x \\ &= xx + qx \end{aligned}$$

Prozessdefinition durch Rekursion

Definition. Sei $n \in \mathbb{N}$, $n \geq 1$. Eine **rekursive Spezifikation** $S_n = (X, T)$ ist gegeben durch eine Folge $X = (x_1, \dots, x_n)$ von Variablen und eine Folge $T = (t_1, \dots, t_n)$ von Prozesstermen mit $FV(t_i) \subseteq \{x_1, \dots, x_n\}$ für alle $i = 1, \dots, n$.

Bemerkungen:

1. Eine rekursive Spezifikation (X, T) repräsentiert ein Gleichungssystem

$$\begin{aligned} x_1 &= t_1, \\ x_2 &= t_2, \\ &\vdots \\ x_n &= t_n. \end{aligned}$$

2. Intention: Eine rekursive Spezifikation (X, T) “definiert” die Prozesse, die “Lösungen” für x_1, \dots, x_n des repräsentierten Gleichungssystems sind, d.h. Prozesse p_1, \dots, p_n mit

$$\begin{aligned} p_1 &= t_1^{[x_1/p_1, \dots, x_n/p_n]}, \\ p_2 &= t_2^{[x_1/p_1, \dots, x_n/p_n]}, \\ &\vdots \\ p_n &= t_n^{[x_1/p_1, \dots, x_n/p_n]}. \end{aligned}$$

Bewachte rekursive Spezifikationen

Problem noch: Nicht jede rekursive Spezifikation definiert eindeutig (bzgl. Verhaltensgleichheit) Prozesse.

Daher: Beschränkung auf rekursive Spezifikationen mit eindeutigen Lösungen.

Im Folgenden: Eine mögliche (syntaktische) Beschränkung.

Definition. Ein Prozessterm t heißt *in Präfixform*, wenn t von der Gestalt

$$t_1 + t_2 + \dots + t_n \quad (n \geq 1) \quad \text{mit: } t_i \equiv a_i \cdot s_i, a_i \in \mathcal{A} \setminus \{\delta\}, s_i \text{ Prozessterme} \\ \text{oder } t_i \equiv b_i, b_i \in \mathcal{A} \\ \text{für } i = 1, \dots, n$$

ist.

Definition. Eine rekursive Spezifikation $((x_1, \dots, x_n), (t_1, \dots, t_n))$ heißt *bewacht*, wenn es für jedes $t_i, i = 1, \dots, n$, einen Prozessterm t'_i in Präfixform gibt, so dass $t_i = t'_i$ herleitbar ist in Σ_{ACP} unter eventuell zusätzlicher Verwendung der Gleichungen $x_j = t_j$ ($j = 1, \dots, n$).

Rekursiv definierte Prozessterme

Vervollständigung der Syntaxdefinition der Prozessterme:

(PT4) Ist $S_n = (X, T)$ eine bewachte rekursive Spezifikation, so sind $\mu_i X(T)$ für alle $i = 1, \dots, n$ Prozessterme mit $FV(\mu_i X(T)) = \emptyset$.

Schreibweisen: $X = (x_1, \dots, x_n), T = (t_1, \dots, t_n) : \mu_i x_1 \dots x_n(t_1, \dots, t_n),$
 $n = 1 : \mu x(t).$

Bemerkungen:

1. Informelle Bedeutung von $\mu_i X(T)$: "Lösung für x_i im durch (X, T) gegebenen Gleichungssystem" (i. Allg.: neuartiger, nicht in ACP enthaltener Prozess).
2. Terme in T müssen keine Variablen enthalten, z.B.: $\mu x(a + b)$ ($= a + b$, bereits in ACP enthalten).
3. Die Variablen x_1, \dots, x_n in $\mu_i X(T)$ heißen *gebunden*. Sie sind beliebig durch andere ("unverbrauchte") ersetzbar.
4. Die Prozessterme t_1, \dots, t_n in $\mu_i X(t_1, \dots, t_n)$ können selbst wieder von der Gestalt $\mu_j \dots$ sein.

4.2 ACP mit Rekursion

Die Sprache ACPR

ACPR (“ACP mit Rekursion”): Erweiterung der Sprache ACP.

Definition. Ein *Prozess* (von ACPR) ist ein Prozessterm p mit $FV(p) = \emptyset$. \mathcal{P} bezeichne die Menge dieser Prozesse.

Operationelle Semantik von ACPR: Erweiterung von (T1)-(T13) um

(T14) Falls $t_i^{[x_1/\mu_1 X(T), \dots, x_n/\mu_n X(T)]} \xrightarrow{a} q$, so $\mu_i X(T) \xrightarrow{a} q$ ($i = 1, \dots, n$)

($X = (x_1, \dots, x_n)$, $T = (t_1, \dots, t_n)$).

Bisimulationen, $p \simeq q$, Gleichheit $p = q$ auf \mathcal{P} : wie bei BSP (d.h.: $p = q \iff p \simeq q$).

Bemerkung:

Für den Spezialfall $n = 1$ lautet (T14):

- Falls $t^{[x/\mu x(t)]} \xrightarrow{a} q$, so $\mu x(t) \xrightarrow{a} q$.

Beispiele:

Es gilt:

1. $\mu x(ax) \xrightarrow{a} \mu x(ax)$,
 $\mu x(ax + b) \xrightarrow{b} \checkmark$,
 $\mu x(ax + b) \xrightarrow{a} \mu x(ax + b)$.
2. $\mu_1 x_1 x_2(ax_1 + ax_2, ax_2 + ax_2) = \mu y(ay)$.

Satz 4.2.1 \simeq ist Kongruenzrelation auf \mathcal{P} , und die Axiome (P1) – (P5), (P6_c), (P7) – (P22) von Σ_{ACP} gelten auch für Prozesse aus \mathcal{P} .

Spezifikationsbeispiele

1. Schokoladenautomat (jetzt formal)

$$\text{SCH: } \mu x(1\text{EUR} \cdot \text{SCHOKO} \cdot x).$$

In “Gleichungsschreibweise”:

$$\text{SCH} = 1\text{EUR} \cdot \text{SCHOKO} \cdot \text{SCH}.$$

2. (Triviales) Erzeuger-Verbraucher-System (ohne Pufferung)

Atomare Aktionen:

- e (Erzeugen eines Objekts),
- s (Senden eines Objekts),
- \bar{s} (Empfangen eines Objekts),
- v (Verbrauchen eines Objekts).

Erzeuger und Verbraucher (jeweils in Gleichungsschreibweise):

$$\begin{aligned} E &= e s E, \\ V &= \bar{s} v V. \end{aligned}$$

Gesamtsystem:

$$EV = \partial_{\{s, \bar{s}\}}(E \| V).$$

3. Werkbank

Informelle Spezifikation:

Zwei Arbeiter bearbeiten an einer Werkbank fortwährend angelieferte Werkstücke, die danach wieder ausgeliefert werden. Sie bearbeiten jeweils ein Werkstück und legen es anschließend (etwa auf einem Förderband) ab. Zur Bearbeitung benutzt jeder Arbeiter entweder ein Werkzeug W_1 oder ein Werkzeug W_2 . Beide Werkzeuge liegen auf der Werkbank bereit und werden jeweils nach Benutzung wieder zurückgelegt.

Atomare Aktionen:

- aufn (Aufnehmen eines angelieferten Werkstücks),
- abl (Ablegen des bearbeiteten Werkstücks),
- arb (Arbeit am Werkstücks),
- $\overline{nimm}_{1/2}$ (Aufnehmen von $W_{1/2}$, synchronisiert mit $\overline{nimm}_{1/2}$),
- $\overline{zurück}_{1/2}$ (Zurücklegen von $W_{1/2}$, synchronisiert mit $\overline{zurück}_{1/2}$).

Arbeiter und Werkzeuge (ebenfalls als Prozesse):

$$\begin{aligned} A &= \text{aufn} \cdot \text{BEARB} \cdot \text{abl} \cdot A, \\ \text{BEARB} &= \overline{nimm}_1 \cdot \text{arb} \cdot \overline{zurück}_1 + \overline{nimm}_2 \cdot \text{arb} \cdot \overline{zurück}_2, \\ W1 &= \overline{nimm}_1 \cdot \overline{zurück}_1 \cdot W1 \\ W2 &= \overline{nimm}_2 \cdot \overline{zurück}_2 \cdot W2. \end{aligned}$$

Gesamtsystem:

$$B = \partial_{\mathcal{R}}(A \| A \| W1 \| W2)$$

mit

$$\mathcal{R} = \{\overline{nimm}_1, \overline{nimm}_2, \overline{nimm}_1, \overline{nimm}_2, \overline{zurück}_1, \overline{zurück}_2, \overline{zurück}_1, \overline{zurück}_2\}.$$

Denotationelle Semantik

Definition. $G = (V_G, E_G, w_G)$ und $H = (V_H, E_H, w_H)$ seien Prozessgraphen. G heißt **Unter-Prozessgraph** von H (in Zeichen: $G \subseteq H$), wenn gilt: $V_G \subseteq V_H$, $E_G \subseteq E_H$, $w_G = w_H$, Kanten in $E_G \cap E_H$ haben die gleiche Marke, und ein Endknoten in $V_G \cap V_H$ ist in G genau dann mit \downarrow markiert, wenn er in H mit \downarrow markiert ist.

Ist $G_1, G_2, G_3, \dots, G_i = (V_i, E_i, w_i)$ eine unendliche Folge von Prozessgraphen mit $G_i \subseteq G_{i+1}$ für $i = 1, 2, 3, \dots$, so ist $\bigcup_{i=1}^{\infty} G_i$ der Prozessgraph (V, E, w) mit $V = \bigcup_{i=1}^{\infty} V_i$, $E = \bigcup_{i=1}^{\infty} E_i$, $w = w_1$ und den aus G_i übernommenen Kanten- und Knotenmarkierungen.

Denotationelle Semantik von ACPR: Erweiterung von (D1)-(D8) um

(D9) Konstruktion von $\llbracket p \rrbracket$ für $p \equiv \mu_i X(T)$ ($X = (x_1, \dots, x_n)$, $T = (t_1, \dots, t_n)$):

- Für $k \geq 1$ und $i = 1, \dots, n$ seien

$$t_i^{(1)} \equiv t_i,$$

$$t_i^{(k+1)} \equiv t_i^{(k)}[x_1/t_1, \dots, x_n/t_n].$$

Weiter seien $G_k = \llbracket t_i^{(k)} \rrbracket$, wobei jede Variable in $t_i^{(k)}$ als atomare Aktion (und $t_i^{(k)}$ damit als Prozess) aufgefasst wird, und G'_k der Prozessgraph, der aus G_k entsteht, wenn alle mit Variablen markierten Kanten und dadurch unerreichbar gewordenen Knoten gestrichen werden. (Es gilt: $G'_k \subseteq G'_{k+1}$ für $k \geq 1$.) Dann ist $\llbracket p \rrbracket = \bigcup_{k=1}^{\infty} G'_k$.

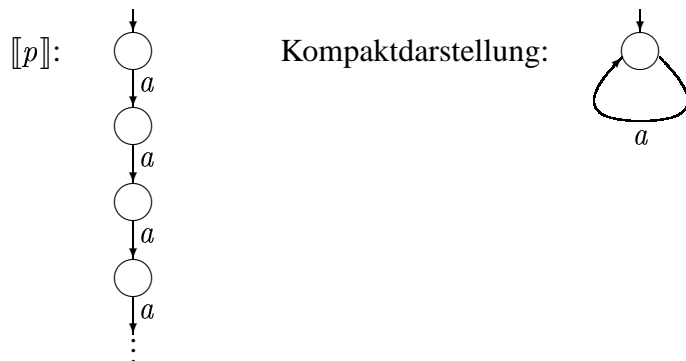
Bisimulationen, $G \Leftrightarrow H$: wie bisher (Abschnitt 3.4).

Bemerkung:

Der Prozessgraph $\llbracket p \rrbracket$ eines Prozesses p gemäß der angegebenen Konstruktion ist im Allgemeinen unendlich.

Statt $\llbracket p \rrbracket$ (als "Prozessgraph von p ") manchmal verwendet: endlicher Prozessgraph G mit $G \Leftrightarrow \llbracket p \rrbracket$ ("Kompaktdarstellung").

Beispiel: $p \equiv \mu x(ax)$.



Axiomatisierung

Satz 4.2.2 Sei $S_n = (X, T)$, $X = (x_1, \dots, x_n)$, $T = (t_1, \dots, t_n)$, eine bewachte rekursive Spezifikation. Für $1 \leq i \leq n$ gilt: $\mu_i X(T) = t_i^{[x_1/\mu_1 X(T), \dots, x_n/\mu_n X(T)]}$.

Bemerkung:

Satz 4.2.2: “Rekursives Definitions-Prinzip” (RDP).

Informell: $\mu_i X(T)$ ist Lösung für x_i des Gleichungssystems $x_1 = t_1, \dots, x_n = t_n$.

Lemma 4.2.3 $S_n = (X, T)$ und $S'_n = (X, R)$ seien bewachte rekursive Spezifikationen mit $X = (x_1, \dots, x_n)$, $T = (t_1, \dots, t_n)$, $R = (r_1, \dots, r_n)$. Für jedes $i = 1, \dots, n$ sei $t_i = r_i$ herleitbar in Σ_{ACP} unter eventuell zusätzlicher Verwendung der Gleichungen $x_j = t_j$ ($j = 1, \dots, n$). Dann gilt für beliebige $p_1, \dots, p_n \in \mathcal{P}$: Falls $p_i = t_i^{[x_1/p_1, \dots, x_n/p_n]}$ für alle $i = 1, \dots, n$, so $p_i = r_i^{[x_1/p_1, \dots, x_n/p_n]}$ für alle $i = 1, \dots, n$.

Satz 4.2.4 Sei $S_n = (X, T)$, $X = (x_1, \dots, x_n)$, $T = (t_1, \dots, t_n)$, eine bewachte rekursive Spezifikation. Für $1 \leq i \leq n$ gilt: Sind $p_1, \dots, p_n \in \mathcal{P}$ Prozesse mit $p_j = t_j^{[x_1/p_1, \dots, x_n/p_n]}$ für alle $j = 1, \dots, n$, so ist $p_i = \mu_i X(T)$.

Theorie Σ_{ACPR}

(Axiome + Regel für Gleichheit von Prozessen in ACPR, Korrektheit folgt aus 4.2.1, 4.2.2, 4.2.4):

Alle Axiome (P1) – (P5), (P6_c), (P7) – (P22) von Σ_{ACP} sowie zusätzlich ($S_n = (X, T)$, $X = (x_1, \dots, x_n)$, $T = (t_1, \dots, t_n)$, $1 \leq i \leq n$):

(P23) $\mu_i X(T) = t_i^{[x_1/\mu_1 X(T), \dots, x_n/\mu_n X(T)]}$.

(P24) Falls $p_j = t_j^{[x_1/p_1, \dots, x_n/p_n]}$ für alle $j = 1, \dots, n$, so $p_i = \mu_i X(T)$.

Satz 4.2.5 $S_n = (X, T)$ und $S'_n = (X, R)$ seien bewachte rekursive Spezifikationen mit $X = (x_1, \dots, x_n)$, $T = (t_1, \dots, t_n)$, $R = (r_1, \dots, r_n)$. Für jedes $i = 1, \dots, n$ sei $t_i = r_i$ herleitbar in Σ_{ACP} unter eventuell zusätzlicher Verwendung der Gleichungen $x_j = t_j$ ($j = 1, \dots, n$). Dann gilt für $1 \leq i \leq n$: $\mu_i X(T) = \mu_i X(R)$.

Bemerkungen:

1. Satz 4.2.5 kann aufgefasst werden als Erweiterung der “Kongruenzeigenschaft” von \Leftrightarrow , z.B. für $n = 1$: $t = r \Rightarrow \mu x(t) = \mu x(r)$.
2. Im praktischen Gebrauch (beim Rechnen mit Gleichungssystemen): Benutzung der Variablen bzw. Lösungsprozesse vermischt.
3. Die Sätze 3.3.5 und 3.3.6 gelten auch in ACPR.
4. Σ_{ACPR} ist nicht vollständig.

Beispiel

Puffer der Kapazität 1 ($D = \{d_1, d_2, \dots\}$: endliche Datenmenge)

Atomare Aktionen (jeweils für alle $d \in D$):

$$\begin{aligned} \bar{e}(d) & \text{ (Empfangen von } d), \\ a(d) & \text{ (Ausgabe von } d). \end{aligned}$$

Spezifikation des Puffers:

$$B_1 = \sum_{d \in D} \bar{e}(d) \cdot a(d) \cdot B_1$$

(Schreibweise für: $\bar{e}(d_1) \cdot a(d_1) \cdot B_1 + \bar{e}(d_2) \cdot a(d_2) \cdot B_1 + \dots$).

Zusammensetzung zweier solcher Puffer B_1 und B'_1

Atomare Aktionen (jeweils für alle $d \in D$): wie oben und zusätzlich

$$\begin{aligned} t(d) & \text{ (Ausgabe von } d \text{ von } B_1), \\ \bar{e}(d) & \text{ (Empfangen von } d \text{ durch } B'_1), \\ t_\gamma(d) & \text{ (Übertragungsaktion zwischen } B_1 \text{ und } B'_1). \end{aligned}$$

Spezifikation von B_1 und B'_1 :

$$\begin{aligned} B_1 & = \sum_{d \in D} \bar{e}(d) \cdot t(d) \cdot B_1, \\ B'_1 & = \sum_{d \in D} \bar{e}(d) \cdot a(d) \cdot B'_1. \end{aligned}$$

Spezifikation der Zusammensetzung:

$$B = \partial_{\{t(d), \bar{e}(d) \mid d \in D\}}(B_1 \parallel B'_1).$$

Es gilt:

$$\begin{aligned} B & = \sum_{d \in D} \bar{e}(d) \cdot t_\gamma(d) \cdot B_d, \\ B_d & = a(d) \cdot B + \sum_{d' \in D} \bar{e}(d') \cdot a(d) \cdot t_\gamma(d') \cdot B_{d'} \quad (\text{für alle } d \in D). \end{aligned}$$

4.3 Approximation von Prozessen

Ausgezeichnete Präfixformen

Gemäß Definition in Abschnitt 4.1: Präfixform von Prozessen aus \mathcal{P} :

$$p_1 + p_2 + \dots + p_n \quad (n \geq 1) \quad \text{mit: } p_i \equiv a_i q_i, a_i \in \mathcal{A} \setminus \{\delta\}, q_i \in \mathcal{P} \\ \text{oder } p_i \equiv b_i, b_i \in \mathcal{A} \\ \text{für } i = 1, \dots, n$$

Für jede bewachte rekursive Spezifikation $(X, (t_1, \dots, t_m))$ seien im Folgenden t'_i fest gewählte Terme in Präfixform mit $t_i = t'_i$ ($i = 1, \dots, m$).

Induktive Definition der ausgezeichneten Präfixform $\varphi(p)$ für jedes $p \in \mathcal{P}$:

$$(\varphi 1) \quad \varphi(a) \equiv a \quad \text{für } a \in \mathcal{A}.$$

$$(\varphi 2) \quad \varphi(pq) \equiv r_1 + \dots + r_n, \quad \text{wobei } \varphi(p) \equiv p_1 + \dots + p_n, \\ r_i \equiv \begin{cases} a_i(p'_i q), & \text{falls } p_i \equiv a_i p'_i, \\ b_i q, & \text{falls } p_i \equiv b_i \in \mathcal{A} \setminus \{\delta\}, \\ \delta, & \text{falls } p_i \equiv \delta, \end{cases} \\ (i = 1, \dots, n).$$

$$(\varphi 3) \quad \varphi(p + q) \equiv \varphi(p) + \varphi(q).$$

$$(\varphi 4) \quad \varphi(p \parallel q) \equiv r_1 + \dots + r_n, \quad \text{wobei } \varphi(p) \equiv p_1 + \dots + p_n, \\ r_i \equiv \begin{cases} a_i(p'_i \parallel q), & \text{falls } p_i \equiv a_i p'_i, \\ b_i q, & \text{falls } p_i \equiv b_i \in \mathcal{A} \setminus \{\delta\}, \\ \delta, & \text{falls } p_i \equiv \delta, \end{cases} \\ (i = 1, \dots, n).$$

$$(\varphi 5) \quad \varphi(p|q) \equiv r_{11} + r_{12} + \dots + r_{ij} + \dots + r_{nm}, \quad \text{wobei} \\ \varphi(p) \equiv p_1 + \dots + p_n, \quad \varphi(q) \equiv q_1 + \dots + q_m, \\ r_{ij} \equiv \begin{cases} \gamma(a_i, b_j) \cdot (p'_i \parallel q'_j), & \text{falls } p_i \equiv a_i p'_i, q_j \equiv b_j q'_j, a_i, b_j \text{ kommunizierend,} \\ \gamma(a_i, b_j) \cdot p'_i, & \text{falls } p_i \equiv a_i p'_i, q_j \equiv b_j, a_i, b_j \text{ kommunizierend,} \\ \gamma(a_i, b_j) \cdot q'_j, & \text{falls } p_i \equiv a_i, q_j \equiv b_j q'_j, a_i, b_j \text{ kommunizierend,} \\ \gamma(a_i, b_j), & \text{falls } p_i \equiv a_i, q_j \equiv b_j, a_i, b_j \text{ kommunizierend,} \\ \delta & \text{sonst,} \end{cases} \\ (i = 1, \dots, n, j = 1, \dots, m).$$

$$(\varphi 6) \quad \varphi(p \parallel q) \equiv \varphi(p \parallel q) + \varphi(q \parallel p) + \varphi(p|q).$$

$$(\varphi 7) \quad \varphi(\partial_{\mathcal{R}}(p)) \equiv r_1 + \dots + r_n, \quad \text{wobei } \varphi(p) \equiv p_1 + \dots + p_n, \\ r_i \equiv \begin{cases} a_i \partial_{\mathcal{R}}(p'_i), & \text{falls } p_i \equiv a_i p'_i, a_i \notin \mathcal{R} \\ b_i, & \text{falls } p_i \equiv b_i \notin \mathcal{R}, \\ \delta, & \text{sonst,} \end{cases} \\ (i = 1, \dots, n).$$

$$(\varphi 8) \quad \varphi(\mu_i X(T)) \equiv t'_i[x_1/\mu_1 X(T), \dots, x_m/\mu_m X(T)], \quad \text{wobei } X = (x_1, \dots, x_m), T = (t_1, \dots, t_m).$$

Beispiele:

1. $\varphi((a + bc)d) \equiv ad + b(cd)$.
2. $\varphi(\mu x(ax)) \equiv a\mu x(ax)$ (Annahme: $t' \equiv ax$).
3. $\varphi(\mu_1 x_1 x_2(ax_1 + ax_2, ax_2 + aax_2)) \equiv a\mu_1 x_1 x_2(\dots) + a\mu_2 x_1 x_2(\dots)$.
4. $\varphi(\mu_2 x_1 x_2(ax_1 + ax_2, ax_2 + aax_2)) \equiv a\mu_2 x_1 x_2(\dots) + aa\mu_2 x_1 x_2(\dots)$.
5. $\varphi(ab \parallel (c + d)) \equiv \varphi(ab \parallel (c + d)) + \varphi((c + d) \parallel ab) + \varphi(ab | (c + d))$
 $\equiv a(b \parallel (c + d)) + cab + dab + \delta + \delta$
 (a, c und a, d nicht kommunizierend).

Lemma 4.3.1 Für beliebige $p \in \mathcal{P}$ gilt:

- a) $\varphi(p)$ ist in Präfixform.
- b) p in Präfixform $\Rightarrow p \equiv \varphi(p)$.
- c) $p = \varphi(p)$.

Projektionen

Definition. Sei $p \in \mathcal{P}$, $k \in \mathbb{N}$. Die **k -Projektion** $\pi_k(p)$ von p ist gegeben wie folgt:

- (π_1) $a \in \mathcal{A}$: $\pi_0(a) \equiv \delta$,
 $\pi_{k+1}(a) \equiv a$.
- (π_2) $a \in \mathcal{A} \setminus \{\delta\}$, $q \in \mathcal{P}$: $\pi_0(aq) \equiv \delta$,
 $\pi_{k+1}(aq) \equiv a\pi_k(q)$.
- (π_3) $\varphi(p) \equiv p_1 + \dots + p_n$: $\pi_k(p) \equiv \pi_k(p_1) + \dots + \pi_k(p_n)$.

Beispiele:

1. Für $p \equiv (a + bc)d$ gilt:

$$\begin{aligned} \pi_0(p) &\equiv \pi_0(ad) + \pi_0(b(cd)) \equiv \delta + \delta, \\ \pi_1(p) &\equiv \pi_1(ad) + \pi_1(b(cd)) \equiv a\pi_0(d) + b\pi_0(cd) \equiv a\delta + b\delta, \\ \pi_2(p) &\equiv a\pi_1(d) + b\pi_1(cd) \equiv ad + bc\delta, \\ \pi_3(p) &\equiv a\pi_2(d) + b\pi_2(cd) \equiv ad + bcd, \\ \pi_k(p) &\equiv ad + bcd \quad \text{für } k > 3. \end{aligned}$$

2. Für $p \equiv ab \parallel (c + d)$ gilt:

$$\begin{aligned} \pi_0(p) &= \delta, \\ \pi_1(p) &= a\delta + c\delta + d\delta, \\ \pi_2(p) &= a(b\delta + c\delta + d\delta) + ca\delta + da\delta, \\ \pi_3(p) &= a(b(c + d) + cb + db) + cab + dab. \end{aligned}$$

Lemma 4.3.2 Für $p \in \mathcal{P}$ und alle $k \in \mathbb{N}$ gilt:

- a) $\pi_k(p) \equiv \pi_k(\varphi(p))$.
- b) $\pi_k(p) \in \mathcal{P}_e$.

Lemma 4.3.3 Für $p, q \in \mathcal{P}$ und alle $k \in \mathbb{N}$ gilt: $\pi_{k+1}(p) = \pi_{k+1}(q) \Rightarrow \pi_k(p) = \pi_k(q)$.

Satz 4.3.4 Für $p, q \in \mathcal{P}$ gilt: $p = q \iff \pi_k(p) = \pi_k(q)$ für alle $k \in \mathbb{N}$.

Eine alternative Axiomatisierung von ACPR

Theorie Σ_{ACPR}^+ (Axiome + Regel für Gleichheit von Prozessen in ACPR):

Alle Axiome (P1) – (P5), (P6_c), (P7) – (P22) von Σ_{ACP} sowie zusätzlich (P23) und (P25) Falls $\pi_k(p) = \pi_k(q)$ für alle $k \in \mathbb{N}$, so $p = q$.

Satz 4.3.5 (Korrektheit und Vollständigkeit von Σ_{ACPR}^+)
Für $p, q \in \mathcal{P}$ gilt: $\Sigma_{\text{ACPR}}^+ \vdash p = q \iff p = q$.

4.4 Sprachvarianten

Projektionen als Operationen in der Sprache

Die Projektionen π_k ($k \in \mathbb{N}$) können auch als weitere Operationen (als Spracherweiterung) in ACPR eingeführt werden.

Operationelle Semantik: Weitere Transitionsregeln:

- Falls $p \xrightarrow{a} p'$ ($p' \not\equiv \checkmark$), so $\pi_{k+1}(p) \xrightarrow{a} \pi_k(p')$.
- Falls $p \xrightarrow{a} \checkmark$, so $\pi_{k+1}(p) \xrightarrow{a} \checkmark$.

Zusätzliche Axiome:

- $\pi_0(a) = \delta$.
- $\pi_{k+1}(a) = a$ ($a \in \mathcal{A}$).
- $\pi_0(ap) = \delta$.
- $\pi_{k+1}(ap) = a\pi_k(p)$ ($a \in \mathcal{A} \setminus \{\delta\}$).
- $\pi_k(p + q) = \pi_k(p) + \pi_k(q)$.

Teilsprachen von ACP mit Rekursion

- Das Rekursionskonzept kann (in gleicher Weise) auch in den Teilsprachen BSP, SPP und SKP eingeführt werden (mit gleicher Semantik und den Axiomen (P23) und (P24)).
- Zur Benutzung der Projektionen π_k und der Regel (P25) müssen diese Sprachen dann auch um die Verklemmung δ erweitert sein.

Lineare Rekursion

Eine rekursive Spezifikation $S_n = (X, T)$ heißt **linear**, wenn jeder Term von T von der Gestalt

$$t_1 + t_2 + \dots + t_n \quad (n \geq 1) \quad \text{mit: } t_i \equiv a_i \cdot x_i, a_i \in \mathcal{A} \setminus \{\delta\}, x_i \in X \\ \text{oder } t_i \equiv b_i, b_i \in \mathcal{A} \\ \text{für } i = 1, \dots, n$$

ist.

Es gilt: Σ_{ACPR} ist vollständig, wenn man in ACPR nur lineare rekursive Spezifikationen zulässt.

Unendliche Spezifikationen

Mögliche Erweiterung von ACPR: Rekursive Spezifikationen $S_\infty = (X, T)$ mit unendlichen Folgen $X = (x_1, x_2, \dots)$ und $T = (t_1, t_2, \dots)$.

Semantik und Axiome/Regeln: wie bisher.

Bemerkung:

Der Eliminationssatz (Satz 3.3.3) für ACPR (d.h. für $p \in \mathcal{P}$ und bezogen auf BSP+Rekursion) gilt, falls unendliche Spezifikationen erlaubt sind, nicht jedoch in ACPR ohne diese Erweiterung.

Beispiel

Zähler Z ("auf einer Zählvariablen x , initialisiert mit 0") mit den atomaren Aktionen

$$\text{plus} \quad ("x := x + 1"), \\ \text{min} \quad ("x := x - 1, \text{ nur möglich, wenn } x \neq 0").$$

Unendliche Spezifikation ($Z \hat{=} Z_0$):

$$Z_0 = \text{plus } Z_1, \\ Z_1 = \text{plus } Z_2 + \text{min } Z_0, \\ Z_2 = \text{plus } Z_3 + \text{min } Z_1, \\ \vdots \\ Z_i = \text{plus } Z_{i+1} + \text{min } Z_{i-1}, \\ \vdots$$

Endliche Spezifikation (d.h. in ACPR):

$$Z = \text{plus } YZ, \\ Y = \text{min} + \text{plus } YY.$$

Spezifikation durch eine einzige Gleichung:

$$C = \text{plus}(C \parallel \text{min}).$$

Kapitel 5

Abstraktion

5.1 Unsichtbare Aktion und Abstraktionsoperator

Abstraktion von Aktionen

Beispiel:

Puffer der Kapazität 2

(Datenmenge D , atomare Aktionen $\bar{e}(d)$, $a(d)$: wie im Puffer-Beispiel in Abschnitt 4.2)

$$B_2 = \sum_{d \in D} \bar{e}(d) \cdot \tilde{B}_d,$$

$$\tilde{B}_d = a(d) \cdot B_2 + \sum_{d' \in D} \bar{e}(d') \cdot a(d) \cdot \tilde{B}_{d'} \quad (\text{für alle } d \in D).$$

Zusammensetzung von 2 Puffern der Kapazität 1 (Abschnitt 4.2):

$$B = \sum_{d \in D} \bar{e}(d) \cdot \tau_\gamma(d) \cdot B_d,$$

$$B_d = a(d) \cdot B + \sum_{d' \in D} \bar{e}(d') \cdot a(d) \cdot \tau_\gamma(d') \cdot B_{d'} \quad (\text{für alle } d \in D).$$

“Nach außen sichtbar”: B_2 und B verhalten sich gleich. ($\tau_\gamma(d)$ und $\tau_\gamma(d')$ sind “interne Aktionen”.)

Wünschenswert (zur formalen Behandlung solcher Phänomene): Sprachmittel zur Abstraktion (“Unsichtbarmachung”, “Verbergen”) von Aktionen (im Beispiel: interner Aktionen).

Die Sprache $ACP^\tau R$

Grundidee:

Abstraktion von der Aktion a geschieht durch Ersetzen von a durch eine ausgezeichnete (“unsichtbare”) Aktion τ

Erweiterung der Sprache $ACPR$ zur Sprache $ACP^\tau R$:

1. \mathcal{A} enthalte ein ausgezeichnetes Element τ (**unsichtbare Aktion, silent action**) mit $\tau \neq \delta$ und $\tau \notin \mathcal{C} \cup \overline{\mathcal{C}} \cup \mathcal{C}_\gamma$.
2. Einschränkung bei Prozesstermen $\partial_{\mathcal{R}}(p): \mathcal{R} \subseteq \mathcal{A} \setminus \{\delta, \tau\}$.
3. Zusätzliche Syntaxregel:

(PT5) Ist t ein Prozessterm, $\mathcal{I} \subseteq \mathcal{A} \setminus \{\delta, \tau\}$, so ist $\tau_{\mathcal{I}}(t)$ ein Prozessterm.

($\tau_{\mathcal{I}}$ heißt **Abstraktionsoperator**;
informelle Bedeutung von $\tau_{\mathcal{I}}(t)$: Ersetze in t alle $a \in \mathcal{I}$ durch τ .)

4. Modifikation der Rekursion: (siehe später).
5. **Prozess** (von $ACP^{\tau}R$): Prozessterm p mit $FV(p) = \emptyset$.
6. \mathcal{P}_{τ} : Menge aller Prozesse von $ACP^{\tau}R$.
7. $\mathcal{P}_0, \mathcal{P}_{\text{prä}}, \mathcal{P}_e$: wie früher (aber mit $\tau \in \mathcal{A}$).

Operationelle Semantik von $ACP^{\tau}R$:

1. Verallgemeinerung der Regeln (T9)-(T11) zu

- (T9 $_{\tau}$) Falls $p \xrightarrow{\tau^n a} p', q \xrightarrow{\tau^m b} q', n, m \geq 0, p' \not\equiv \checkmark, q' \not\equiv \checkmark, a, b$ kommunizierend,
so $p \parallel q \xrightarrow{\gamma(a,b)} p' \parallel q'$ und $p|q \xrightarrow{\gamma(a,b)} p'|q'$.
- (T10 $_{\tau}$) Falls $p \xrightarrow{\tau^n a} p', q \xrightarrow{\tau^m b} \checkmark, n, m \geq 0, p' \not\equiv \checkmark, a, b$ kommunizierend,
so $p \parallel q \xrightarrow{\gamma(a,b)} p', p|q \xrightarrow{\gamma(a,b)} p', q \parallel p \xrightarrow{\gamma(a,b)} p'$ und $q|p \xrightarrow{\gamma(a,b)} p'$.
- (T11 $_{\tau}$) Falls $p \xrightarrow{\tau^n a} \checkmark, q \xrightarrow{\tau^m b} \checkmark, n, m \geq 0, a, b$ kommunizierend,
so $p \parallel q \xrightarrow{\gamma(a,b)} \checkmark$ und $p|q \xrightarrow{\gamma(a,b)} \checkmark$.

2. Erweiterung von (T1)-(T8),(T9 $_{\tau}$)-(T11 $_{\tau}$),(T12)-(T14) um

- (T15) Falls $p \xrightarrow{a} p', p' \not\equiv \checkmark$ und $a \in \mathcal{I}$, so $\tau_{\mathcal{I}}(p) \xrightarrow{\tau} \tau_{\mathcal{I}}(p')$.
- (T16) Falls $p \xrightarrow{a} p', p' \not\equiv \checkmark$ und $a \notin \mathcal{I}$, so $\tau_{\mathcal{I}}(p) \xrightarrow{a} \tau_{\mathcal{I}}(p')$.
- (T17) Falls $p \xrightarrow{a} \checkmark$ und $a \in \mathcal{I}$, so $\tau_{\mathcal{I}}(p) \xrightarrow{\tau} \checkmark$.
- (T18) Falls $p \xrightarrow{a} \checkmark$ und $a \notin \mathcal{I}$, so $\tau_{\mathcal{I}}(p) \xrightarrow{a} \checkmark$.

Beispiele:

Es gilt:

$$\begin{aligned} a\tau + \tau_{\{b\}}(bc) &\xrightarrow{a} \tau \xrightarrow{\tau} \checkmark, \\ a\tau + \tau_{\{b\}}(bc) &\xrightarrow{\tau} \tau_{\{b\}}(c) \xrightarrow{c} \checkmark, \\ \tau\tau a|\bar{a} &\xrightarrow{a\gamma} \checkmark. \end{aligned}$$

Denotationelle Semantik von $ACP^{\tau}R$:

1. Modifikation von (D7)

(D7 $_{\tau}$) $p, q \in \mathcal{P}_{\tau}$; Konstruktion von $\llbracket p|q \rrbracket$ aus $\llbracket p \rrbracket$ und $\llbracket q \rrbracket$:

- Bilde $\llbracket p \parallel q \rrbracket$ (= G_0 , Wurzel w_0).
- w_1, w_2, \dots seien die Knoten von G_0 (evtl. unendlich viele), die von w_0 aus durch eine Folge von mit τ markierten Kanten erreichbar sind. G_1, G_2, \dots seien die Unterprozessgraphen von G_0 mit Wurzeln w_1, w_2, \dots
- Für alle $i = 0, 1, 2, \dots$ bilde G'_i wie folgt: Entferne in G_i alle Kanten $w_i \xrightarrow{a} v$ mit $a \notin \mathcal{C}_{\gamma}$ sowie alle dadurch nicht mehr erreichbaren Teile von G_i . Wird w_i dadurch Endknoten, so markiere w_i mit \downarrow .
- Bilde " $G'_0 + G'_1 + G'_2 + \dots$ " wie in (D3), d.h. identifiziere alle Wurzeln. Die gemeinsame Wurzel erhält Marke \downarrow , wenn alle Wurzeln vorher mit \downarrow markiert sind.

2. Erweiterung der induktiven Definition (D1)-(D6),(D7_τ), (D8),(D9) von $\llbracket p \rrbracket$ um

(D10) $p \in \mathcal{P}_\tau$; Konstruktion von $\llbracket \tau_{\mathcal{I}}(p) \rrbracket$ aus $\llbracket p \rrbracket$:

- Ersetze in allen Kanten $v_1 \xrightarrow{a} v_2$ mit $a \in \mathcal{I}$ die Markierung a durch τ .

Beobachtungsäquivalenz

Schreibweisen ($p \in \mathcal{P}_\tau, p' \in \mathcal{P}_\tau \cup \{\checkmark\}, a \in \mathcal{A} \setminus \{\delta, \tau\}$):

$p \xrightarrow{\tau} p'$ für: $p \equiv p'$ oder $p \xrightarrow{\tau^n} p'$ mit $n \geq 1$,

$p \not\xrightarrow{\tau} p'$ für: nicht $p \xrightarrow{\tau} p'$

$p \xrightarrow{\tau}_+ p'$ für: $p \xrightarrow{\tau^n} p'$ mit $n \geq 1$,

$p \xrightarrow{a} p'$ für: $p \xrightarrow{\tau} p_1 \xrightarrow{a} p_2 \xrightarrow{\tau} p'$.

Definition. Eine zweistellige Relation β auf \mathcal{P}_τ heißt **Beobachtungsäquivalenz (observation equivalence)**, wenn für $p, q \in \mathcal{P}_\tau$ und $a \in \mathcal{A}$ gilt:

(τBS1) Falls $\beta(p, q)$ und $p \xrightarrow{a} p', p' \not\xrightarrow{\tau} \checkmark$, so gibt es $q' \in \mathcal{P}_\tau$ mit $q \xrightarrow{a} q'$ und $\beta(p', q')$.

(τBS2) Falls $\beta(p, q)$ und $q \xrightarrow{a} q', q' \not\xrightarrow{\tau} \checkmark$, so gibt es $p' \in \mathcal{P}_\tau$ mit $p \xrightarrow{a} p'$ und $\beta(p', q')$.

(τBS3) Falls $\beta(p, q)$, so: $p \xrightarrow{a} \checkmark$ genau dann, wenn $q \xrightarrow{a} \checkmark$.

$p, q \in \mathcal{P}_\tau$ heißen **beobachtungsäquivalent** (in Zeichen: $p \simeq_0 q$), wenn es eine Beobachtungsäquivalenz β auf \mathcal{P}_τ gibt, so dass $\beta(p, q)$ gilt.

Satz 5.1.1 \simeq_0 ist eine Äquivalenzrelation auf \mathcal{P}_τ .

Verhaltensgleichheit

Definition. $p, q \in \mathcal{P}_\tau$ heißen **τ-bisimilar** (in Zeichen: $p \simeq_\tau q$), wenn $p \simeq_0 q$ und zusätzlich gilt:

(τBS4) Falls $p \xrightarrow{\tau} p', p' \not\xrightarrow{\tau}_+ \checkmark$, so gibt es $q' \in \mathcal{P}_\tau$ mit $q \xrightarrow{\tau}_+ q'$ und $p' \simeq_0 q'$.

(τBS5) Falls $q \xrightarrow{\tau} q', q' \not\xrightarrow{\tau}_+ \checkmark$, so gibt es $p' \in \mathcal{P}_\tau$ mit $p \xrightarrow{\tau}_+ p'$ und $p' \simeq_0 q'$.

Satz 5.1.2 \simeq_τ ist eine Kongruenzrelation auf \mathcal{P}_τ (bzgl. der Operationen $\cdot, +, \llbracket _ \rrbracket, \|_ \|_ , \mid, \partial_{\mathcal{R}}, \tau_{\mathcal{I}}$).

Definition. $p, q \in \mathcal{P}_\tau$ heißen **verhaltensäquivalent (gleich, $p = q$)** (in der Bisimulations-Semantik), wenn gilt: $p \simeq_\tau q$.

Satz 5.1.3 Für alle $p, q \in \mathcal{P}_\tau, a \in \mathcal{A} \setminus \{\delta, \tau\}$ gilt:

a) $p\tau = p$.

b) $\tau p = \tau p + p$.

c) $a(\tau p + q) = a(\tau p + q) + ap$.

Rekursion

Übernahme aus den Abschnitten 2.2, 2.3, 4.1: $\mathcal{P}_0, \mathcal{P}_{\text{prä}}$, Präfixform (jetzt mit τ).

Definition. Ein Prozessterm

$$t \equiv t_1 + t_2 + \dots + t_n$$

in Präfixform heißt **abstraktionsfrei**, wenn t keinen Abstraktionsoperator enthält und wenn für $i = 1, \dots, n$ gilt: Ist $t_i \equiv a_i s_i$, so ist $a_i \neq \tau$.

Modifikation der Rekursion: Ersetze die Definition bewachter rekursiver Spezifikationen (Abschnitt 4.1) durch:

Eine rekursive Spezifikation $((x_1, \dots, x_n), (t_1, \dots, t_n))$ heißt **bewacht**, wenn es für jedes t_i , $i = 1, \dots, n$, einen abstraktionsfreien Prozessterm t'_i in Präfixform gibt, so dass $t_i = t'_i$ herleitbar ist in Σ_{ACP} unter eventuell zusätzlicher Verwendung der Gleichungen $x_j = t_j$ ($j = 1, \dots, n$).

Beispiel:

Der Prozess τ^ω kann nicht “direkt” als $\mu \dots$ definiert werden, aber:

$$\tau^\omega = \tau_{\{a\}}(\mu x(ax)).$$

5.2 Axiomatisierung

Bisherige Axiome

Es gilt:

- Die Axiome (P1)-(P5), (P6_c), (P7)-(P10), (P14)-(P24) gelten auch für Prozesse aus \mathcal{P}_τ .

Modifikation der Axiome (P11)-(P13):

$$(P11_\tau) \quad ap|b = (a|b)p \quad (a, b \in \mathcal{A} \setminus \{\tau\}).$$

$$(P12_\tau) \quad a|bp = (a|b)p \quad (a, b \in \mathcal{A} \setminus \{\tau\}).$$

$$(P13_\tau) \quad ap|bq = (a|b) \cdot (p||q) \quad (a, b \in \mathcal{A} \setminus \{\tau\}).$$

Weitere Axiome

Axiome für τ und $\tau_{\mathcal{I}}$:

1. Zusammenspiel von τ und $|$:

$$(P26) \quad \tau|p = \delta.$$

$$(P27) \quad p|\tau = \delta.$$

$$(P28) \quad \tau p|q = p|q.$$

$$(P29) \quad p|\tau q = p|q.$$

2. “Charakteristische” Axiome für τ :

$$(P30) \quad p\tau = p.$$

$$(P31) \quad \tau p = \tau p + p.$$

$$(P32) \quad a(\tau p + q) = a(\tau p + q) + ap \quad (a \in \mathcal{A} \setminus \{\delta, \tau\}).$$

3. Axiome für $\tau_{\mathcal{I}}$:

$$(P33) \quad \tau_{\mathcal{I}}(a) = a \quad \text{für } a \notin \mathcal{I}.$$

$$(P34) \quad \tau_{\mathcal{I}}(a) = \tau \quad \text{für } a \in \mathcal{I}.$$

$$(P35) \quad \tau_{\mathcal{I}}(p + q) = \tau_{\mathcal{I}}(p) + \tau_{\mathcal{I}}(q).$$

$$(P36) \quad \tau_{\mathcal{I}}(pq) = \tau_{\mathcal{I}}(p) \cdot \tau_{\mathcal{I}}(q).$$

Die Theorien $\Sigma_{\text{ACP}\tau}$ und $\Sigma_{\text{ACP}\tau\text{R}}$

Theorie $\Sigma_{\text{ACP}\tau}$ (Axiome für Gleichheit von Prozessen in $\text{ACP}^{\tau}\text{R}$ ohne Rekursion):

(P1)-(P5),(P6_c),(P7)-(P10),(P11 _{τ})-(P13 _{τ}),(P14)-(P22),(P26)-(P36).

Es gilt:

1. Zu jedem $p \in \mathcal{P}_0$ gibt es ein $p' \in \mathcal{P}_{\text{prä}}$ mit $p = p'$
(und $p = p'$ kann in $\Sigma_{\text{ACP}\tau}$ allein mit (P1)-(P5),(P16),(P17) hergeleitet werden).

2. Eliminationssatz

Zu jedem Prozess $p \in \mathcal{P}_{\tau}$, der keinen Rekursionsoperator enthält, gibt es einen Prozess $p' \in \mathcal{P}_0$ mit $p = p'$
(und $p = p'$ kann in $\Sigma_{\text{ACP}\tau}$ hergeleitet werden).

3. Sind p und q Prozesse aus \mathcal{P}_{τ} , die keinen Rekursionsoperator enthalten, so gilt:
 $p = q \Rightarrow \Sigma_{\text{ACP}\tau} \vdash p = q.$

Theorie $\Sigma_{\text{ACP}\tau\text{R}}$ (Axiome + Regel für Gleichheit von Prozessen in $\text{ACP}^{\tau}\text{R}$):

Alle Axiome von $\Sigma_{\text{ACP}\tau}$ und zusätzlich (P23) und (P24).

Approximationen

Präfixform $\varphi(p)$ für $p \in \mathcal{P}_\tau$:

1. Übernahme der Definition von $\varphi(p)$ wie bisher mit folgender Modifikation von ($\varphi 6$):

$$\begin{aligned}
 (\varphi 6_\tau) \quad \varphi(p|q) &\equiv r_{11} + r_{12} + \dots + r_{ij} + \dots + r_{nm}, \quad \text{wobei} \\
 \varphi(p) &\equiv p_1 + \dots + p_n, \quad \varphi(q) \equiv q_1 + \dots + q_m, \\
 r_{ij} &\equiv \begin{cases} \delta, & \text{falls } p_i \equiv \tau \text{ oder } q_j \equiv \tau, \\ \varphi(p'_i|q_j), & \text{falls } p_i \equiv \tau p'_i, q_j \not\equiv \tau, \\ \varphi(p_i|q'_j), & \text{falls } p_i \not\equiv \tau, q_j \equiv \tau q'_j, \\ \text{wie bisher} & \text{sonst,} \end{cases} \\
 &(i = 1, \dots, n, j = 1, \dots, m).
 \end{aligned}$$

2. Definition von $\varphi(\tau_{\mathcal{I}}(p))$:

$$\begin{aligned}
 (\varphi 9) \quad \varphi(\tau_{\mathcal{I}}(p)) &\equiv r_1 + \dots + r_n, \quad \text{wobei } \varphi(p) \equiv p_1 + \dots + p_n, \\
 r_i &\equiv \begin{cases} \tau \cdot \tau_{\mathcal{I}}(p'_i), & \text{falls } p_i \equiv a_i p'_i, a_i \in \mathcal{I}, \\ \tau, & \text{falls } p_i \equiv b_i \in \mathcal{I}, \\ a \cdot \tau_{\mathcal{I}}(p'_i), & \text{falls } p_i \equiv a_i p'_i, a_i \notin \mathcal{I}, \\ b_i, & \text{falls } p_i \equiv b_i \notin \mathcal{I}, \end{cases} \\
 &(i = 1, \dots, n).
 \end{aligned}$$

Definition von π_k : wie bisher (mit $a \not\equiv \tau$) und zusätzlich (für $k \geq 0$):

1. $\pi_k(\tau) \equiv \tau$.
2. $\pi_k(\tau p) \equiv \tau \pi_k(p)$.

Lemma 5.2.1 Sei $p \in \mathcal{P}_\tau$ von der Form $p \equiv \mu_i X(T)$. Dann ist $\pi_k(p) \in \mathcal{P}_e$ für alle $k \in \mathbb{N}$.

Lemma 5.2.2 Sei $p, q \in \mathcal{P}_\tau$, $\pi_k(p)$ und $\pi_k(q)$ definiert für alle $k \in \mathbb{N}$. Dann gilt:

$$p = q \iff \pi_k(p) = \pi_k(q) \text{ für alle } k \in \mathbb{N}.$$

Theorie $\Sigma_{\text{ACP}\tau\text{R}}^+$ (Axiome + Regel für Gleichheit von Prozessen in $\text{ACP}\tau\text{R}$):

Alle Axiome von $\Sigma_{\text{ACP}\tau}$ und zusätzlich (P23) und (P25).

Satz 5.2.3 (Korrektheit von $\Sigma_{\text{ACP}\tau\text{R}}^+$)

Für $p, q \in \mathcal{P}_\tau$ gilt: $\Sigma_{\text{ACP}\tau\text{R}}^+ \vdash p = q \Rightarrow p = q$.

Bemerkung: $\Sigma_{\text{ACP}\tau\text{R}}^+$ ist nicht vollständig.

Weitere Gesetze

Satz 5.2.4 Für $p, q \in \mathcal{P}_\tau$, $a \in \mathcal{A} \setminus \{\delta, \tau\}$ gilt:

- a) $p|q = q|p$.
- b) $p||q = q||p$.
- c) $p|\delta = \delta$.

- d) $p \parallel \delta = p \parallel \delta = p\delta$.
- e) $\delta \parallel p = \delta$.
- f) $(p|q)|r = p|(q|r) = \delta$.
- g) $(p \parallel q) \parallel r = p \parallel (q \parallel r)$.
- h) $p|(aq \parallel r) = (p|aq) \parallel r$.
- i) $(p \parallel q) \parallel r = p \parallel (q \parallel r)$.

Satz 5.2.5 (Expansionssatz)

Für $p_1, \dots, p_n \in \mathcal{P}_e$, $n \geq 3$ gilt:

$$p_1 \parallel p_2 \parallel \dots \parallel p_n = \sum_{i=1}^n p_i \parallel (p_1 \parallel \dots \parallel p_{i-1} \parallel p_{i+1} \parallel \dots \parallel p_n) + \sum_{\substack{i,j=1 \\ i < j}}^n (p_i | p_j) \parallel (p_1 \parallel \dots \parallel p_{i-1} \parallel p_{i+1} \parallel \dots \parallel p_{j-1} \parallel p_{j+1} \parallel \dots \parallel p_n).$$

Spezifikationen**Beispiel:**

Für die Pufferspezifikationen in den Abschnitten 4.2 und 5.1 gilt:

$$\tau_{\{\tau_\gamma(d) \mid d \in D\}}(\mathbb{B}) = \mathbb{B}_2.$$

Allgemeine (typische) Form von Spezifikationen
(von Prozessen mit interner Aktionenmenge \mathcal{I}):

- $\partial_{\mathcal{R}}(P_1 \parallel \dots \parallel P_n)$ (interne Aktionen “sichtbar”),
- $\tau_{\mathcal{I}}(\partial_{\mathcal{R}}(P_1 \parallel \dots \parallel P_n))$ (nur mehr mit “Schnittstellen” nach außen).

5.3 Fairness der Abstraktion**Faire Auswahl von unsichtbaren Aktionen****Beispiel:**

Mögliche Abläufe von $\tau_{\{a\}}(\mu x(ax + b))$ ($a \neq b$):

- Nach einer gewissen Zeit (d.h.: endlich vielen unsichtbaren Schritten) erfolgt b .

nicht aber:

- Unendlich oft unsichtbare Schritte (b erfolgt nie).

Allgemein:

Ein Prozess wählt nicht unendlich oft unsichtbare Schritte, wenn auch sichtbare möglich sind (*Fairness* (der Abstraktion)).

Satz 5.3.1 Seien $p, q \in \mathcal{P}_\tau$, $\mathcal{I} \subseteq \mathcal{A} \setminus \{\delta, \tau\}$, $a \in \mathcal{I}$. Dann gilt:

$$p = ap + q \Rightarrow \tau_{\mathcal{I}}(p) = \tau \cdot \tau_{\mathcal{I}}(q).$$

Beispiel:

Es gilt ($a \neq b$): $\tau_{\{a\}}(\mu x(ax + b)) = \tau \cdot \tau_{\{a\}}(b) = \tau \cdot b$.

Koomen's Fair Abstraction Rule

Verallgemeinerung von Satz 5.3.1:

(KFAR_n) Seien $p_1, \dots, p_n, q_1, \dots, q_n \in \mathcal{P}_\tau$, $\mathcal{I} \subseteq \mathcal{A} \setminus \{\delta, \tau\}$, $a_1, \dots, a_n \in \mathcal{I}$ ($n \geq 1$).
Dann gilt:

$$\left. \begin{array}{l} p_1 = a_1 p_2 + q_1 \\ p_2 = a_2 p_3 + q_2 \\ \vdots \\ p_n = a_n p_1 + q_n \end{array} \right\} \Rightarrow \tau_{\mathcal{I}}(p_1) = \tau \cdot (\tau_{\mathcal{I}}(q_1) + \tau_{\mathcal{I}}(q_2) + \dots + \tau_{\mathcal{I}}(q_n)).$$

(KFAR_n) für $n = 1, 2$:

$$n = 1: \quad p = ap + q \Rightarrow \tau_{\mathcal{I}}(p) = \tau \cdot \tau_{\mathcal{I}}(q) \quad (\text{Satz 5.3.1}).$$

$$n = 2: \quad \left. \begin{array}{l} p_1 = a_1 p_2 + q_1 \\ p_2 = a_2 p_1 + q_2 \end{array} \right\} \Rightarrow \tau_{\mathcal{I}}(p_1) = \tau \cdot (\tau_{\mathcal{I}}(q_1) + \tau_{\mathcal{I}}(q_2)).$$

Die Cluster Fair Abstraction Rule

Definition. Seien $p_i \equiv \mu_i x_1 \dots x_n(t_1, \dots, t_n) \in \mathcal{P}_\tau$ ($i = 1, \dots, n$), $\mathcal{I} \subseteq \mathcal{A} \setminus \{\delta, \tau\}$, $V \subseteq X = \{x_1, \dots, x_n\}$. V heißt **Cluster von \mathcal{I} in X** , wenn für jedes $x_i \in V$ gilt:

$$t_i \equiv a_1 x_{i_1} + \dots + a_m x_{i_m} + x_{j_1} + \dots + x_{j_k}$$

mit $m \geq 1$, $k \geq 0$, $a_1, \dots, a_m \in \mathcal{I} \cup \{\tau\}$, $x_{i_1}, \dots, x_{i_m} \in V$, $x_{j_1}, \dots, x_{j_k} \in X \setminus V$.

Für $x_i, x_j \in X$ sei

$$x_i \curvearrowright x_j \iff x_j \text{ ist in } t_i \text{ enthalten.}$$

\curvearrowright^* sei die reflexive, transitive Hülle von \curvearrowright .

Die Menge $A(V)$ der **Ausgänge** des Clusters V ist gegeben durch

$$A(V) = \{y \in X \setminus V \mid \text{es gibt } x \in V \text{ mit } x \curvearrowright y\}.$$

Das Cluster V heißt **konservativ**, wenn $x \curvearrowright^* y$ gilt für alle $x \in V$, $y \in A(V)$.

Weitere Verallgemeinerung von Satz 5.3.1:

(CFAR) Seien $p_i \equiv \mu_i x_1 \dots x_n(t_1, \dots, t_n) \in \mathcal{P}_\tau$ ($i = 1, \dots, n$), $\mathcal{I} \subseteq \mathcal{A} \setminus \{\delta, \tau\}$, V ein konservatives Cluster von \mathcal{I} in $X = \{x_1, \dots, x_n\}$, $A(V) = \{x_{i_1}, \dots, x_{i_m}\}$, $x_k \in V$.

Dann gilt:

$$\tau_{\mathcal{I}}(p_k) = \tau \cdot (\tau_{\mathcal{I}}(p_{i_1}) + \dots + \tau_{\mathcal{I}}(p_{i_m})).$$

Bemerkung:

(KFAR_n) ist ein Spezialfall von (CFAR).

Beispiel:

Würfeln, bis eine "sechs" gewürfelt ist; Spezifikation:

$$p = \tau_{\mathcal{I}}(p')$$

mit $\mathcal{I} = \{\text{Wurf, eins, zwei, drei, vier, fünf}\}$ und p' Lösung für x in:

$$\begin{aligned} x &= \text{Wurf} \cdot x_1, \\ x_1 &= \text{eins} \cdot x + x_2, \\ x_2 &= \text{zwei} \cdot x + x_3, \\ x_3 &= \text{drei} \cdot x + x_4, \\ x_4 &= \text{vier} \cdot x + x_5, \\ x_5 &= \text{fünf} \cdot x + x_6, \\ x_6 &= \text{sechs}. \end{aligned}$$

Konservatives Cluster von \mathcal{I} in $\{x, x_1, \dots, x_6\}$: $\{x, x_1, x_2, x_3, x_4, x_5\}$;

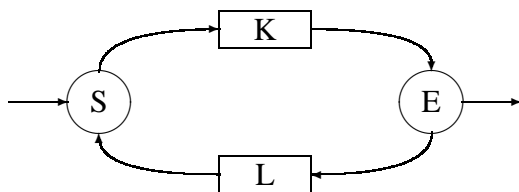
Ausgang: x_6 .

Mit (CFAR) folgt: $p = \tau_{\mathcal{I}}(p') = \tau \cdot \tau_{\mathcal{I}}(\text{sechs}) = \tau \cdot \text{sechs}$.

5.4 Beispiel: Das Alternating Bit Protocol

Informelle Beschreibung

Datenübertragungsschema:



Der "Sender" S soll bei ihm eingehende Datenelemente d über den Kanal K an den "Empfänger" übertragen, wo diese ausgegeben werden. K ist unzuverlässig, Sendungen können zerstört werden.

Lösung im Alternating Bit Protocol:

- Datenelemente werden (als “Paket”) zusammen mit einem Kontrollbit b übertragen; diese Sendung wird wiederholt, bis S das von E (auf einem ebenfalls unzuverlässigen Kanal L) als Bestätigung zurückgesendete b erhalten hat.
- Danach wird die nächste Eingabe gelesen und mit geändertem Kontrollbit ebenso verfahren usw.
- E gibt die Datenelemente unzerstörter Pakete aus und sendet das zugehörige b zurück.
- Zerstörte Sendungen (über K oder L) seien als solche erkennbar.

Formale Spezifikation

D sei eine endliche Menge von (zu übertragenden) Daten;
 \perp bezeichne eine “zerstörte Sendung”.

Atomare Aktionen

($d \in D, b \in \{0, 1\}, b' \in \{0, 1, \perp\}, v \in D \times \{0, 1\}, v' \in D \times \{0, 1\} \cup \{\perp\}$):

$e_S(d)$	Eingabe von d in S (von “außen”),		
$a_{SK}(v)$	Ausgabe von v von S an K ,	$e_{SK}(v)$	Eingabe von v in K ,
$a_{KE}(v')$	Ausgabe von v' von K an E ,	$e_{KE}(v')$	Eingabe von v' in E ,
$a_{EL}(b)$	Ausgabe von b von E an L ,	$e_{EL}(b)$	Eingabe von b in L ,
$a_{LS}(b')$	Ausgabe von b' von L an S ,	$e_{LS}(b')$	Eingabe von b' in S ,
$a_E(d)$	Ausgabe von d bei E (nach “außen”).		

Spezifikation der einzelnen Prozesse ($d \in D, b \in \{0, 1\}$):

Sender:

$$S = S_0 \cdot S_1 \cdot S,$$

$$S_b = \sum_{d \in D} e_S(d) \cdot S_{db},$$

$$S_{db} = a_{SK}(d, b) \cdot B_{db},$$

$$B_{db} = (e_{LS}(1 - b) + e_{LS}(\perp)) \cdot S_{db} + e_{LS}(b).$$

Kanäle:

$$K = \sum_{\substack{d \in D \\ b \in \{0, 1\}}} e_{SK}(d, b) \cdot (u \cdot a_{KE}(d, b) + z \cdot a_{KE}(\perp)) \cdot K,$$

$$L = \sum_{b \in \{0, 1\}} e_{EL}(b) \cdot (u \cdot a_{LS}(b) + z \cdot a_{LS}(\perp)) \cdot L.$$

(u, z : interne Aktionen der Kanäle.)

Empfänger:

$$E = E_0 \cdot E_1 \cdot E,$$

$$E_b = \sum_{d \in D} e_{KE}(d, b) \cdot a_E(d) \cdot a_{EL}(b) +$$

$$\left(\sum_{d \in D} e_{KE}(d, 1 - b) + e_{KE}(\perp) \right) \cdot e_{EL}(1 - b) \cdot E_b.$$

Zur Zusammensetzung des Gesamtprozesses sei:

$$\begin{aligned}\bar{a}_{SK}(v) &\equiv e_{SK}(v), & \mathfrak{t}_{SK}(v) &\equiv \gamma(\mathfrak{a}_{SK}(v), e_{SK}(v)), \\ \bar{a}_{KE}(v') &\equiv e_{KE}(v'), & \mathfrak{t}_{KE}(v') &\equiv \gamma(\mathfrak{a}_{KE}(v'), e_{KE}(v')), \\ \bar{a}_{EL}(b) &\equiv e_{EL}(b), & \mathfrak{t}_{EL}(b) &\equiv \gamma(\mathfrak{a}_{EL}(b), e_{EL}(b)), \\ \bar{a}_{LS}(b') &\equiv e_{LS}(b'), & \mathfrak{t}_{LS}(b') &\equiv \gamma(\mathfrak{a}_{LS}(b'), e_{LS}(b')).\end{aligned}$$

$$\mathcal{R} = \{\mathfrak{a}_{SK}(v), e_{SK}(v), \mathfrak{a}_{KE}(v'), e_{KE}(v'), \mathfrak{a}_{EL}(b), e_{EL}(b), \mathfrak{a}_{LS}(b'), e_{LS}(b') \mid b \in \{0, 1\}, b' \in \{0, 1, \perp\}, v \in D \times \{0, 1\}, v' \in D \times \{0, 1\} \cup \{\perp\}\}.$$

$$\mathcal{I} = \{\mathfrak{t}_{SK}(v), \mathfrak{t}_{KE}(v'), \mathfrak{t}_{EL}(b), \mathfrak{t}_{LS}(b'), u, z \mid b \in \{0, 1\}, b' \in \{0, 1, \perp\}, v \in D \times \{0, 1\}, v' \in D \times \{0, 1\} \cup \{\perp\}\}.$$

$$\begin{aligned}\text{Gesamtprozess: } \quad \text{ABP}' &= \partial_{\mathcal{R}}(S \parallel K \parallel L \parallel E), \\ \text{ABP} &= \tau_{\mathcal{I}}(\text{ABP}').\end{aligned}$$

Korrektheit des Systems

$$\text{Behauptung: } \quad \text{ABP} = \sum_{d \in D} e_S(d) \cdot a_E(d) \cdot \text{ABP}.$$

Beweisidee: Seien

$$\begin{aligned}X &= \partial_{\mathcal{R}}(S \parallel K \parallel L \parallel E), \\ Y &= \partial_{\mathcal{R}}(S_1 \cdot S \parallel K \parallel L \parallel E_1 \cdot E)\end{aligned}$$

Es gilt

$$\begin{aligned}\tau_{\mathcal{I}}(X) &= \sum_{d \in D} e_S(d) \cdot a_E(d) \cdot \tau_{\mathcal{I}}(Y), \\ \tau_{\mathcal{I}}(Y) &= \sum_{d \in D} e_S(d) \cdot a_E(d) \cdot \tau_{\mathcal{I}}(X),\end{aligned}$$

also:

$$\tau_{\mathcal{I}}(X) = \tau_{\mathcal{I}}(Y).$$

Damit erhält man:

$$\begin{aligned}\text{ABP} = \tau_{\mathcal{I}}(X) &= \sum_{d \in D} e_S(d) \cdot a_E(d) \cdot \tau_{\mathcal{I}}(Y) \\ &= \sum_{d \in D} e_S(d) \cdot a_E(d) \cdot \tau_{\mathcal{I}}(X) \\ &= \sum_{d \in D} e_S(d) \cdot a_E(d) \cdot \text{ABP}.\end{aligned}$$

Kapitel 6

Andere Semantiken

6.1 Spur-Semantik

Die Sprache BSP^δ

Seien BSP^δ die Sprache BSP mit $\delta \in \mathcal{A}$, \mathcal{P}_0^δ die Menge aller Prozesse von BSP^δ .

Axiome (P1)-(P5),(P16),(P17): korrekte und vollständige Axiomatisierung der Gleichheit (gemäß Bisimulations-Semantik) von Prozessen aus \mathcal{P}_0^δ .

Alle folgenden Betrachtungen: nur für BSP^δ .

Grundidee

Bisher: Bisimulations-Semantik.
 Andere Sichtweise: Verhalten eines Prozesses gegeben durch "mögliche (Teil-) Abläufe" (*Spuren, traces*).

Beispiel:

(Informell:) Spuren von $(a + bc)d$:

- | | | |
|---|---------------------|---------------|
| 1. "Gar nichts ausführen" | beschreibbar durch: | ε |
| 2. "a ausführen" | | a |
| 3. "Erst a, danach d ausführen" | | ad |
| 4. "b ausführen" | | b |
| 5. "Erst b, danach c ausführen" | | bc |
| 6. "Erst b, danach c, danach d ausführen" | | bcd |

Grundidee: $p = q \iff \{\text{Spuren von } p\} = \{\text{Spuren von } q\}$.

Erweiterte Spuren und Verhaltensäquivalenz

Definition. Sei $\downarrow \notin \mathcal{A}$. Ist $\sigma \in (\mathcal{A} \setminus \{\delta\})^*$, so heißen σ und $\sigma\downarrow$ ($\in ((\mathcal{A} \setminus \{\delta\}) \cup \{\downarrow\})^*$) *erweiterte Spuren* (über \mathcal{A}).

Induktive Definition der Menge $tr(p)$ von erweiterten Spuren für $p \in \mathcal{P}_0^\delta$:

- (tr1) $tr(\delta) = \{\varepsilon\}$.
- (tr2) $tr(a) = \{\varepsilon, a, a\downarrow\}$ für $a \in \mathcal{A} \setminus \{\delta\}$.
- (tr3) $tr(pq) = \{\sigma \in tr(p) \mid \sigma \text{ enthält kein } \downarrow\} \cup \{\sigma\varrho \mid \sigma\downarrow \in tr(p), \varrho \in tr(q)\}$.
- (tr4) $tr(p + q) = tr(p) \cup tr(q)$.

Beispiele:

$$tr(\delta a) = \{\varepsilon\}.$$

$$tr((a + bc)d) = \{\varepsilon, a, b, bc, ad, ad\downarrow, bcd, bcd\downarrow\}.$$

$$tr(a(b + c)) = \{\varepsilon, a, ab, ac, ab\downarrow, ac\downarrow\}.$$

$$tr(ab + ac) = \{\varepsilon, a, ab, ac, ab\downarrow, ac\downarrow\}.$$

Bemerkung:

Offenbar gilt: $\varepsilon \in tr(p)$ für alle $p \in \mathcal{P}_0^\delta$.

Definition. $p, q \in \mathcal{P}_0^\delta$ heißen *verhaltensäquivalent in der Spur-Semantik* ($p = q$), wenn gilt: $tr(p) = tr(q)$.

Axiomatisierung

Satz 6.1.1 Die Axiome (P1)-(P5),(P16),(P17) gelten auch in der Spur-Semantik.

Satz 6.1.2 In der Spur-Semantik gilt für alle $p, q, r \in \mathcal{P}_0^\delta$:

$$(TR) \quad p(q + r) = pq + pr.$$

Theorie $\Sigma_{\text{BSP}}^{\text{tr}}$: Axiome (P1)-(P5),(P16),(P17) und (TR).

Satz 6.1.3 (Korrektheit und Vollständigkeit von $\Sigma_{\text{BSP}}^{\text{tr}}$)

Für $p, q \in \mathcal{P}_0^\delta$ gilt: $\Sigma_{\text{BSP}}^{\text{tr}} \vdash p = q \iff p = q$ in der Spur-Semantik.

Bemerkung:

Die Spur-Semantik ist “wenig differenzierend”, insbesondere in Bezug auf Verklemmungen. Z.B. gilt:

$$ab = a(b + \delta) = ab + a\delta.$$

Somit: Spur-Semantik nicht geeignet in Anwendungen, in denen man am Verklemmungsverhalten von Prozessen interessiert ist.

6.2 Readiness-Semantik**Ready-Mengen und Verhaltensäquivalenz**

Grundidee: $p \longmapsto$ Menge von erweiterten Spuren + zusätzliche Information über Verklemmungsverhalten

Definition. Eine *Ready-Menge* ist eine Menge von Paaren (σ, M) mit $\sigma \in (\mathcal{A} \setminus \{\delta\})^*$ und $M \subseteq (\mathcal{A} \setminus \{\delta\}) \cup \{\downarrow\}$.

Intention: σ : Spur

(keine erweiterten Spuren notwendig, Information über \downarrow steckt in M);

M : Menge der Aktionen, die nach der Aktionsfolge σ ausführbar sind

($\downarrow \in M$: Terminierung möglich).

Induktive Definition der Ready-Menge $rm(p)$ für $p \in \mathcal{P}_0^\delta$:

$$(rm1) \quad rm(\delta) = \{(\varepsilon, \emptyset)\}.$$

$$(rm2) \quad rm(a) = \{(\varepsilon, \{a\}), (a, \{\downarrow\})\} \quad \text{für } a \in \mathcal{A} \setminus \{\delta\}.$$

$$(rm3) \quad rm(pq) = \{(\sigma, M) \in rm(p) \mid \downarrow \notin M\} \cup \\ \{(\sigma, M \cup M_q) \mid (\sigma, M \cup \{\downarrow\}) \in rm(p) \text{ mit } \downarrow \notin M, (\varepsilon, M_q) \in rm(q)\} \cup \\ \{(\sigma \varrho, M') \mid \varrho \neq \varepsilon \text{ und es gibt } M \text{ mit} \\ (\sigma, M \cup \{\downarrow\}) \in rm(p) \text{ und } (\varrho, M') \in rm(q)\}.$$

$$(rm4) \quad rm(p + q) = (rm(p) \setminus \{(\varepsilon, M_p) \in rm(p)\}) \cup \\ (rm(q) \setminus \{(\varepsilon, M_q) \in rm(q)\}) \cup \\ \{(\varepsilon, M_p \cup M_q) \mid (\varepsilon, M_p) \in rm(p), (\varepsilon, M_q) \in rm(q)\}.$$

Beispiele:

$$rm(ab) = \{(\varepsilon, \{a\}), (a, \{b\}), (ab, \{\downarrow\})\}.$$

$$rm(a\delta) = \{(\varepsilon, \{a\}), (a, \emptyset)\}.$$

$$rm(ab + a\delta) = \{(\varepsilon, \{a\}), (a, \{b\}), (ab, \{\downarrow\}), (a, \emptyset)\}.$$

Definition. $p, q \in \mathcal{P}_0^\delta$ heißen *verhaltensäquivalent in der Readiness-Semantik* ($p = q$), wenn gilt: $rm(p) = rm(q)$.

Axiomatisierung

Satz 6.2.1 Die Axiome (P1)-(P5),(P16),(P17) gelten auch in der Readiness-Semantik.

Satz 6.2.2 In der Readiness-Semantik gilt für alle $p, q, r, s \in \mathcal{P}_0^\delta$, $a, b \in \mathcal{A} \setminus \{\delta\}$:

$$(RE1) \quad a(bp + r) + a(bq + s) = a(bp + bq + r) + a(bp + bq + s).$$

$$(RE2) \quad a(b + r) + a(bq + s) = a(b + bq + r) + a(b + bq + s).$$

Theorie $\Sigma_{\text{BSP}}^{\text{re}}$: Axiome (P1)-(P5),(P16),(P17) und (RE1),(RE2).

Satz 6.2.3 (Korrektheit und Vollständigkeit von $\Sigma_{\text{BSP}}^{\text{re}}$)

Für $p, q \in \mathcal{P}_0^\delta$ gilt: $\Sigma_{\text{BSP}}^{\text{re}} \vdash p = q \iff p = q$ in der Readiness-Semantik.

6.3 Failure-Semantik

Failure-Mengen und Verhaltensäquivalenz

Variante der Readiness-Semantik:

Andere Art der Zusatzinformation über Verklemmungsverhalten.

Definition. Eine *Failure-Menge* ist eine Menge von Paaren (σ, F) mit $\sigma \in (\mathcal{A} \setminus \{\delta\})^*$ und $F \subseteq (\mathcal{A} \setminus \{\delta\}) \cup \{\downarrow\}$.

Intention: σ : Spur;

F : Menge von Aktionen, deren Ausführung nach der Aktionenfolge σ (eventuell) nicht möglich ist.

Definition der Failure-Menge $fm(p)$ für $p \in \mathcal{P}_0^\delta$:

$$fm(p) = \{(\sigma, F) \mid \text{es gibt } (\sigma, M) \in rm(p) \text{ und } F \subseteq ((\mathcal{A} \setminus \{\delta\}) \cup \{\downarrow\}) \setminus M\}.$$

Beispiele:

$$\begin{aligned} fm(ab) = & \{(\varepsilon, \emptyset), (\varepsilon, \{b\}), (\varepsilon, \{\downarrow\}), (\varepsilon, \{b, \downarrow\}), \\ & (a, \emptyset), (a, \{a\}), (a, \{\downarrow\}), (a, \{a, \downarrow\}), \\ & (ab, \emptyset), (ab, \{a\}), (ab, \{b\}), (ab, \{a, b\})\}. \\ fm(ab + a\delta) = & fm(ab) \cup \{(a, \{b\}), (a, \{a, b\}), (a, \{b, \downarrow\}), (a, \{a, b, \downarrow\})\}. \end{aligned}$$

Definition. $p, q \in \mathcal{P}_0^\delta$ heißen *verhaltensäquivalent in der Failure-Semantik* ($p = q$), wenn gilt: $fm(p) = fm(q)$.

Axiomatisierung

Lemma 6.3.1 Für $p, q \in \mathcal{P}_0^\delta$ gilt: $rm(p) \subseteq rm(q) \Rightarrow fm(p) \subseteq fm(q)$.

Satz 6.3.2 Die Axiome (P1)-(P5),(P16),(P17),(RE1),(RE2) gelten auch in der Failure-Semantik.

Satz 6.3.3 In der Failure-Semantik gilt für alle $p, q, r \in \mathcal{P}_0^\delta$, $a \in \mathcal{A} \setminus \{\delta\}$:

$$(FA) \quad ap + a(q + r) = ap + a(p + q) + a(q + r).$$

Theorie $\Sigma_{\text{BSP}}^{\text{fa}}$: Axiome (P1)-(P5),(P16),(P17),(RE1),(RE2) und (FA).

Satz 6.3.4 (Korrektheit und Vollständigkeit von $\Sigma_{\text{BSP}}^{\text{fa}}$)

Für $p, q \in \mathcal{P}_0^\delta$ gilt: $\Sigma_{\text{BSP}}^{\text{fa}} \vdash p = q \iff p = q$ in der Failure-Semantik.

Zusammenfassung

Es gilt:

- $p = q$ in der Bisimulations-Semantik $\Rightarrow p = q$ in der Readiness-Semantik
- (RE1) gilt nicht allgemein in der Bisimulations-Semantik.
- $p = q$ in der Readiness-Semantik $\Rightarrow p = q$ in der Failure-Semantik
- (FA) gilt nicht allgemein in der Readiness-Semantik.
- $p = q$ in der Failure-Semantik $\Rightarrow p = q$ in der Spur-Semantik
- (TR) gilt nicht allgemein in der Failure-Semantik.

Damit ergibt sich folgender Zusammenhang der behandelten Semantiken:

Bisimulations-Semantik \triangleright Readiness-Semantik \triangleright Failure-Semantik \triangleright Spur-Semantik.

($S_1 \triangleright S_2 : p = q$ in $S_1 \Rightarrow p = q$ in S_2 , aber nicht umgekehrt,
“ S_1 ist stärker differenzierend als S_2 ”.)