GOETHE UNIVERSITÄT FRANKFURT AM MAIN

www.uni-frankfurt.de

Aktuelle Themen zu Informatik der Systeme:

Nebenläufige Programmierung: Praxis und Semantik

Semantische Modelle nebenläufiger Programmiersprachen (2)

WS 2011/12

Stand der Folien: 31. Januar 2012

Übersicht

1. Der π-Kalkül: Einleitung
2. Synchroner π-Kalkül
3. Der asynchrone π-Kalkül
4. Erweiterungen und Varianten des π-Kalküls
   - Nichtdeterministische Auswahl
   - Polyadischer π-Kalkül
   - Rekursive Definitionen
5. Prozess-Gleichheit im π-Kalkül
   - Starke Bisimulation
   - Bisimulation
   - Barbed Kongruenz

Der π-Kalkül

Rückblick:

- Der Lambda-Kalkül ist ein Modell sequentieller Programmiersprachen
- wobei Operationen und Daten durch Funktionen (Abstraktionen) ausgedrückt wird.
- der Kontrollfluss wird durch Anwendungen von Funktionen auf Argumente ausgedrückt.

Der π-Kalkül (2)

Der π-Kalkül:

- Der π-Kalkül ist ein Modell nebenläufiger und verteilter Programmiersprachen
- Daten und Operationen werden durch Prozesse ausgedrückt
- der Kontrollfluss wird durch Prozesskommunikation ausgedrückt.
- D.h. ein Message-Passing-Modell: Kommunikation nur über Nachrichten

Der π-Kalkül (3)

- Entwickelt von Robin Milner, Joachim Parrow and David Walker in den 1990er Jahren

Anwendungen auch außerhalb der nebenläufigen Programmierung sogar außerhalb der Informatik, z.B.

- Spi-Kalkül: π-Kalkül zur Beschreibung und Verfizieren von kryptographischen Protokollen
- Microsofts XLANG: Beschreibungssprache für Geschäftsprozesse
- Biochemie: Stochastischer π-Kalkül zur formalen Darstellung biochemischer Prozesse

Der π-Kalkül (4)

Verschiedene Varianten

- Synchroner π-Kalkül
- Asynchroner π-Kalkül
- Mit oder ohne Summen: Nichtdeterministische Auswahl
- Mit Rekursion / mit Replikation
- monadisch / polyadisch
- etc.

Synchroner π-Kalkül ohne Summe mit Replikation

Syntax

- $\mathcal{N}$ abzählbar unendliche Menge von Namen (ähnlich zu Variablen)
- Syntax für π-Kalkül-Prozesse ($x \in \mathcal{N}$)

$$
\begin{array}{rcll}
P & ::= & \pi.P & \text{(Aktion)} \\
 & \mid & P_1 \mid P_2 & \text{(Parallele Komposition)} \\
 & \mid & !\,P & \text{(Replikation)} \\
 & \mid & \mathbf{0} & \text{(Inaktiver Prozess)} \\
 & \mid & \nu x.P & \text{(Restriktion)}
\end{array}
$$

- Syntax für Aktionspräfixe wobei $x, y \in \mathcal{N}$

$$
\begin{array}{rcll}
\pi & ::= & x(y) & \text{Input} \\
 & \mid & \overline{x}\langle y\rangle & \text{Output}
\end{array}
$$

Synchroner π-Kalkül (2)

- $x(y).P$ bedeutet:
  Empfange über den Kanal namens x einen Namen und binde ihn an y.
  Nachdem der Name empfangen wurde: Verhalte dich wie P, wobei für y der empfangene Name eingesetzt wird.
- $\overline{x}\langle y\rangle.P$ bedeutet:
  Sende über den Kanal namens x den Namen y. Danach verhalte dich wie P.
- $\mathbf{0}$ ist der inaktive Prozess, der nichts tut
- $P_1 \mid P_2$ ist die parallele Komposition von P_1 und P_2.
- $\nu x.P$ führt P als Geltungsbereich für x ein.
- Replikation $!\,P$ steht für unendlich viele parallele Ausführungen von P, d.h. $\underbrace{P \mid P \mid \ldots \mid P}_{\text{unendlich oft}}$.

Bindungsbereiche

Binder im π-Kalkül sind:

- Restriktion $\nu z.P$ bindet den Namen z mit Geltungsbereich P
- $x(y).P$ bindet y mit Geltungsbereich P
- Beachte: Der Output-Präfix $\overline{x}\langle y\rangle$ bindet weder x noch y.

Freie Namen fn(P)

$$
\begin{aligned}
\mathsf{fn}(x(y).P) &= \{x\} \cup (\mathsf{fn}(P) \setminus \{y\}) \\
\mathsf{fn}(\overline{x}\langle y\rangle.P) &= \{x,y\} \cup \mathsf{fn}(P) \\
\mathsf{fn}(P_1 \mid P_2) &= \mathsf{fn}(P_1) \cup \mathsf{fn}(P_2) \\
\mathsf{fn}(\mathbf{0}) &= \emptyset \\
\mathsf{fn}(\nu x.P) &= \mathsf{fn}(P) \setminus \{x\} \\
\mathsf{fn}(!\,P) &= \mathsf{fn} P
\end{aligned}
$$

Gebundene Namen bn(P)

$$
\begin{aligned}
\mathsf{bn}(x(y).P) &= \{y\} \cup \mathsf{bn}(P) \\
\mathsf{bn}(\overline{x}\langle y\rangle.P) &= \mathsf{bn}(P) \\
\mathsf{bn}(P_1 \mid P_2) &= \mathsf{bn}(P_1) \cup \mathsf{bn}(P_2) \\
\mathsf{bn}(\mathbf{0}) &= \emptyset \\
\mathsf{bn}(\nu x.P) &= \{x\} \cup \mathsf{bn}(P) \\
\mathsf{bn}(!\,P) &= \mathsf{bn} P
\end{aligned}
$$

Alle Namen eines Prozesses: $\mathsf{n}(P) := \mathsf{fn}(P) \cup \mathsf{bn}(P)$

Alpha-Äquivalenz

Definition (Alpha-Äquivalenz)

Eine Umbenennung gebundener Namen eines Prozesses P ist die Ersetzung eines Unterterms

- $x(y).P'$ von P durch $x(z).P'[z/y]$, oder
- $\nu y.P'$ von P durch $\nu z.P'[z/y]$.

Prozesse P_1, P_2 heißen α-äquivalent, wenn sie durch Umbenennungen gebundener Namen gleich gemacht werden können.

Wie vorher:

- Alpha-äquivalente Prozesse werden als gleich angesehen.
- Annahme: Distinct Name Convention

Prozesskontexte

Prozesskontext D

- Prozess, der an einer Position anstelle eines Prozesses ein Loch $[\cdot]$ hat.
- Grammatik:

$$D ::= [\cdot] \mid \pi.D \mid D \mid P \mid P \mid D \mid !\,D \mid \nu x.D \qquad \text{für } x \in \mathcal{N}$$

Strukturelle Kongruenz

Strukturelle Kongruenz $\equiv$: Kleinste Kongruenz auf Prozessen, die die folgenden Axiome erfüllt.

$$
\begin{aligned}
P &\equiv Q, \text{ falls } P \text{ und } Q \ \alpha\text{-äquivalent sind} \\
P_1 \mid (P_2 \mid P_3) &\equiv (P_1 \mid P_2) \mid P_3 \\
P_1 \mid P_2 &\equiv P_2 \mid P_1 \\
P \mid \mathbf{0} &\equiv P \\
\nu z.\nu w.P &\equiv \nu w.\nu z.P \\
\nu z.\mathbf{0} &\equiv \mathbf{0} \\
\nu z.(P_1 \mid P_2) &\equiv P_1 \mid \nu z.P_2, \text{ falls } z \notin \mathsf{fn}(P_1) \\
!\,P &\equiv P \mid !\,P
\end{aligned}
$$

- Umgekehrt ausgedrückt: Kann man P_1 mithilfe der Axiome (angewendet auf Unterprozesse) und α-Umbenennungen in P_2 überführen, dann sind P_1 und P_2 strukturell kongruent (d.h. $P_1 \equiv P_2$)

Operationale Semantik

Reduktionsregeln

(Interact)	$x(y).P \mid \overline{x}\langle v\rangle.Q \rightarrow P[v/y] \mid Q$
(Par)	$P \mid Q \rightarrow P' \mid Q$, falls $P \rightarrow P'$
(New)	$\nu x.P \rightarrow \nu x.P'$, falls $P \rightarrow P'$
(StructCongr)	$P \rightarrow P'$, falls $Q \rightarrow Q'$, $P \equiv Q$ und $P' \equiv Q'$

Beispiele

Betrachte: $P \equiv \underbrace{x(y).\overline{y}\langle y\rangle.\mathbf{0}}_{P_1} \mid \underbrace{\overline{x}\langle z\rangle.\mathbf{0}}_{P_2} \mid \underbrace{z(w).\mathbf{0}}_{P_3}$

- P_1 und P_2 kommunizieren über den Kanal x, wobei P_2 an P_1 den Namen z verschickt:

$$P \rightarrow \underbrace{\overline{z}\langle z\rangle.\mathbf{0}}_{P_1'} \mid \underbrace{\mathbf{0}}_{P_2'} \mid \underbrace{z(w).\mathbf{0}}_{P_3} \equiv P'$$

- P_1' und P_3 kommunizieren über den Kanal z, wobei P_1' den Namen z verschickt:

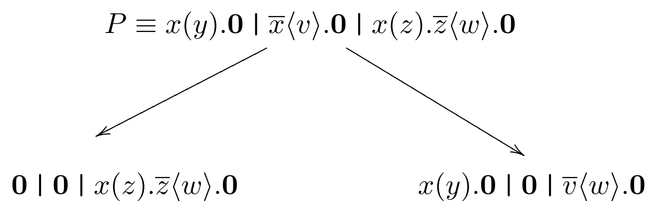
$$P' \rightarrow \mathbf{0} \mid \mathbf{0} \mid \mathbf{0} \equiv P''$$

- Ergebis ist strukturell kongruent zu $\mathbf{0}$ und kann nicht weiter reduziert werden.

Beispiele (2)

- Die Reduktion ist nicht-deterministisch
- Betrachte:

$$P \equiv x(y).\mathbf{0} \mid \overline{x}\langle v\rangle.\mathbf{0} \mid x(z).\overline{z}\langle w\rangle.\mathbf{0}$$



Beispiele (3)

- ν-Binder verhindern Kommunikationsmöglichkeiten und ermöglichen lokale Kommunikation, die durch den Eingriff von außen geschützt ist.
- Betrachte:

$$\begin{aligned} P \equiv\ & \nu x.(x(y).\mathbf{0} \mid \overline{x}\langle v\rangle.\mathbf{0}) \mid x(z).\overline{z}\langle w\rangle.\mathbf{0} \\ \rightarrow\ & \nu x.(\mathbf{0} \mid \mathbf{0}) \mid x(z).\overline{z}\langle w\rangle.\mathbf{0} \end{aligned}$$

- nur eine Reduktion möglich!
- Nach α-Umbenennung von P:

$$P \equiv \nu x'.(x'(y).\mathbf{0} \mid \overline{x'}\langle v\rangle.\mathbf{0}) \mid x(z).\overline{z}\langle w\rangle.\mathbf{0}$$

Extrusion

- Lokale Namen, können ihren Bindungsbereich durch Kommunikation verlassen.
- Betrachte
$$P \equiv x(y).P_1 \mid \nu z.\overline{x}\langle z\rangle.P_2$$
wobei z nicht in P_1 vorkommt.
- Bei Reduktion mit (INTERACT):
$$P \equiv \nu z.(x(y).P_1 \mid \overline{x}\langle z\rangle.P_2) \to \nu z.(P_1[z/x] \mid P_2)$$
- Der scheinbar nur für P_2 bekannte Namen z wird auch für P_1 bekannt
- Dieses Phänomen wird als Extrusion bezeichnet.

Unendliche Reduktionsfolgen

- Unendlich lange Reduktionsfolgen sind möglich
- Beispiel:
$$P = !\,(\overline{x}\langle v\rangle.\mathbf{0}) \mid !\,(x(z).\mathbf{0})$$
- Anfang der Reduktion

$$\begin{aligned}
P &= !\,(\overline{x}\langle v\rangle.\mathbf{0}) \mid !\,(x(z).\mathbf{0}) \\
&\equiv !\,(\overline{x}\langle v\rangle.\mathbf{0}) \mid \overline{x}\langle v\rangle.\mathbf{0} \mid !\,(x(z).\mathbf{0}) \\
&\equiv !\,(\overline{x}\langle v\rangle.\mathbf{0}) \mid \overline{x}\langle v\rangle.\mathbf{0} \mid x(z).\mathbf{0} \mid !\,(x(z).\mathbf{0}) \\
&\to !\,(\overline{x}\langle v\rangle.\mathbf{0}) \mid \mathbf{0} \mid \mathbf{0} \mid !\,(x(z).\mathbf{0}) \\
&\equiv !\,(\overline{x}\langle v\rangle.\mathbf{0}) \mid !\,(x(z).\mathbf{0}) \\
&\equiv P
\end{aligned}$$

Turing-Mächtigkeit des π-Kalküls

- Bewiesen von Robin Milner, 1992
- Kodiere den call-by-name Lambda Kalkül in den π-Kalkül
- Übersetzung $[\![\cdot]\!]$
- Für Lambda-Ausdruck s: $[\![s]\!]u$ ist Prozess
- wobei u ein Name ist
- über u “erhält” s seine Argumente

Kodierung $[\![\cdot]\!]$

Übersetzung von Abstraktionen

$$[\![\lambda x.s]\!]u := u(x).u(v).[\![s]\!]v$$

- Über u empfängt die Abstraktion zwei Namen:
- Zum einen x,
- zum anderen den Namen über den s seine Argumente erhält

Übersetzung von Variablen

$$[\![x]\!]u := \overline{x}\langle u\rangle.\mathbf{0}$$

Beispiel: $[\![\lambda x.x]\!]u = u(x).u(v).\overline{x}\langle v\rangle.\mathbf{0}$

Kodierung $[\![\cdot]\!]$ (2)

Übersetzung von Anwendungen

- Bei $(s\ t)$ wird quasi eine Bindung $x = t$ für das Argument als eigener Prozess angelegt.
- Der Prozess für $x = t$ fordert jedes Mal einen Kanalnamen an, über welchen er seine Argumente erhält
- Abkürzung:
$$\langle x = t\rangle := \mathbf{!}\,(x(w).[\![t]\!]w)$$
- Replikation: Da t eventuell öfter gebraucht wird
- Übersetzung der Anwendung, wobei x ein neuer Name:

$$[\![s\ t]\!]u := \nu v.([\![s]\!]v \mid \nu x.\overline{v}\langle x\rangle.\overline{v}\langle u\rangle.\langle x = t\rangle)$$

Beispiel

$$\begin{array}{rcl}
[\![(\lambda x.x)\ t]\!]u & = & \nu v.([\![(\lambda x.x)]\!]v \mid \nu y.\overline{v}\langle y\rangle.\overline{v}\langle u\rangle.\langle y = t\rangle) \\
& = & \nu v.(v(x).v(w).[\![x]\!]w \mid \nu y.\overline{v}\langle y\rangle.\overline{v}\langle u\rangle.\langle y = t\rangle) \\
& = & \nu v.(v(x).v(w).\overline{x}\langle w\rangle.\mathbf{0} \mid \nu y.\overline{v}\langle y\rangle.\overline{v}\langle u\rangle.\langle y = t\rangle) \\
& \equiv & \nu y.\nu v.(v(x).v(w).\overline{x}\langle w\rangle.\mathbf{0} \mid \overline{v}\langle y\rangle.\overline{v}\langle u\rangle.\langle y = t\rangle) \\
& \to & \nu y.\nu v.(v(w).\overline{y}\langle w\rangle.\mathbf{0} \mid \overline{v}\langle u\rangle.\langle y = t\rangle) \\
& \to & \nu y.\nu v.(\overline{y}\langle u\rangle.\mathbf{0} \mid \langle y = t\rangle) \\
& = & \nu y.\nu v.(\overline{y}\langle u\rangle.\mathbf{0} \mid \mathbf{!}\,(y(w).[\![t]\!]w)) \\
& \equiv & \nu y.\nu v.(\overline{y}\langle u\rangle.\mathbf{0} \mid (y(w).[\![t]\!]w) \mid \mathbf{!}\,(y(w).[\![t]\!]w)) \\
& \to & \nu y.\nu v.(\mathbf{0} \mid ([\![t]\!]u) \mid \mathbf{!}\,(y(w).[\![t]\!]w)) \\
& \equiv & [\![t]\!]u \mid \nu y.(\mathbf{!}\,(y(w).[\![t]\!]w))
\end{array}$$

Beispiel (2)

Zusammengefasst:

$$[\![(\lambda x.x)\ t]\!]u \xrightarrow{*} [\![t]\!]u \mid \nu y.(\mathbf{!}\,(y(w).[\![t]\!]w))$$

- $\nu y.(\mathbf{!}\,(y(w).[\![t]\!]w))$ kann nicht kommunizieren, daher „Garbage“
- Im call-by-name Lambda Kalkül: $(\lambda x.x)\ t \to t$

Resultate aus Milner, 1992

Proposition

Sei s ein geschlossener Lambda-Ausdruck, dann gilt eine der folgenden Bedingungen:

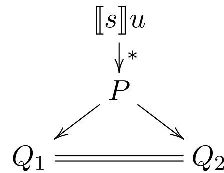
1. $s \downarrow_{name}$ und $[\![s]\!]u \xrightarrow{*} P$, sodass P irreduzibel ist.
2. $s \uparrow_{name}$ und es gibt kein P, so dass $[\![s]\!]u \xrightarrow{*} P$, wobei P irreduzibel ist.

Informal: Call-by-name Auswertung von Lambda-Ausdrücken kann im π-Kalkül simuliert werden.

Resultate aus Milner, 1992 (2)

Lemma

Sei s ein geschlossener Lambda-Ausdruck, dann ist $[\![s]\!]u$ deterministisch bzgl. $\equiv$, d.h. falls $[\![s]\!]u \xrightarrow{*} P$ und $P \to Q_1$ als auch $P \to Q_2$, dann gilt stets $Q_1 \equiv Q_2$.



Insgesamt folgt:

Theorem

Der synchrone π-Kalkül ist Turing-mächtig.

Bemerkungen

- Resultat zeigt nur, dass Lambda-Ausdrücke ausgeführt werden können.
- Kein Zusammenhang zwischen Gleichheitstheorien
- Milner 1992 zeigt stärkere Aussagen über $[\![\cdot]\!]$
- Wir haben allerdings (noch) keinen Gleichheitsbegriff für den π-Kalkül definiert
- Es gibt verschiedene Begriffe!
- Milner 1992 gibt zusätzlich eine Kodierung des call-by-value Lambda-Kalküls in den π-Kalkül an.

Der asynchrone π-Kalkül

- Bisher: Die Kommunikation findet in einem Schritt statt, d.h. synchrone Kommunikation
- Im asynchronen π-Kalkül: Fast alles identisch, nur ein Unterschied:
- Nach dem Output-Präfix $\overline{x}\langle y\rangle$ darf nur der inaktive Prozess $\mathbf{0}$ folgen.

Syntax

$$P \quad ::= \quad \mathbf{0} \mid \overline{x}\langle y\rangle.\mathbf{0} \mid x(y).P \mid P_1 \mid P_2 \mid \nu z.P \mid !P$$

Der asynchrone π-Kalkül (2)

- Der asynchrone Charakter kommt durch die Sichtweise:
- $\overline{x}\langle y\rangle.\mathbf{0}$ sind gesendete Nachrichten, die im Raum „herumschwirren“

$$P \equiv \underbrace{\overline{x_1}\langle y_1\rangle.\mathbf{0} \mid \ldots \mid \overline{x_n}\langle y_n\rangle.\mathbf{0}}_{\text{Medium der gesendeten Nachrichten (Puffer)}} \mid \underbrace{Q_1 \mid \ldots \mid Q_n}_{\text{„echtes“ Programm}}$$

Kodierung: Synchroner π-Kalkül in den asynchronen

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

- Umkehrung klar: Da asynchroner π-Kalkül ein Subkalkül des synchronen ist.
- Sei $P' \mid Q' \equiv (\overline{x}\langle z\rangle.P \mid x(y).Q)$ ein Prozess des synchronen π-Kalküls.
- Wir nennen P' den Sender, Q' den Empfänger
- Ziel: P und Q sind blockiert bis die Kommunikation statt gefunden hat.
- Erste Idee: Übersetze $\overline{x}\langle z\rangle.P$ als $\overline{x}\langle z\rangle.\mathbf{0} \mid P$
- Funktioniert nicht: P kann weiterreduzieren ohne auf die Kommunikation zu warten!
- Idee: Verwende asynchrone Kommunikation, wobei der Sender auf eine Empfangsbestätigung wartet.

Kodierung mit Empfangsbestätigung

Sender $S = \overline{x}\langle z\rangle.\mathbf{0} \mid u(v).P$
Empfänger $E = x(y).(\overline{u}\langle v\rangle.\mathbf{0} \mid Q)$

Auswertung dazu:

$$\begin{array}{ll} & S \mid E \\ = & (\overline{x}\langle z\rangle.\mathbf{0} \mid u(v).P) \mid x(y).(\overline{u}\langle v\rangle.\mathbf{0} \mid Q) \\ & \rightarrow u(v).P \mid \overline{u}\langle v\rangle.\mathbf{0} \mid Q[z/y] \\ & \rightarrow P \mid Q[z/x] \end{array}$$

Kodierung mit Empfangsbestätigung (2)

- Schwachpunkt: Nicht unabhängig vom Kontext:
- Übersetzung von $D[P' \mid Q']$stellt nicht sicher, dass andere Prozesse aus D über Kanal u kommunizieren.
- Beispiel $D = u(w).\mathbf{0} \mid [\cdot]$
- Dann kann $D[S \mid E]$ wie folgt reduzieren:

$$\begin{array}{ll} & u(w).\mathbf{0} \mid (\overline{x}\langle z\rangle.\mathbf{0} \mid u(v).P) \mid x(y).(\overline{u}\langle v\rangle.\mathbf{0} \mid Q) \\ \rightarrow & u(w).\mathbf{0} \mid (u(v).P) \mid \overline{u}\langle v\rangle.\mathbf{0} \mid Q[z/y] \\ \rightarrow & (u(v).P) \mid Q[z/y] \end{array}$$

- Lösung: zuerst ein privater Kanalname zwischen Sender und Empfänger austauschen

Kodierung mit Empfangsbestätigung (3)

- Sender $\nu u.(\overline{x}\langle u\rangle.\mathbf{0} \mid u(v)P)$
- Dann kann $u(v)P$ erst dann weiter reduzieren, wenn u über x versendet wurde, da u vorher nach außen unsichtbar!
- $u(v)P$ blockiert bis die Kommunikation stattgefunden hat.
- Damit kann man synchronisieren

Kodierung mit Empfangsbestätigung (4)

Übersetzung $[\![\cdot]\!]_\pi$ übersetzt synchrone in asynchrone Prozesse

$$
\begin{array}{lcl}
[\![\mathbf{0}]\!]_\pi & = & \mathbf{0} \\
[\![\overline{x}\langle y\rangle.P]\!]_\pi & = & \nu u.(\overline{x}\langle u\rangle.\mathbf{0} \mid u(v).(\overline{v}\langle y\rangle.\mathbf{0} \mid [\![P]\!]_\pi)), \\
 & & \text{wobei } u,v \notin \mathsf{fn}(P) \\
[\![x(y).P]\!]_\pi & = & x(u).\nu v.(\overline{u}\langle v\rangle.\mathbf{0} \mid v(y).[\![P]\!]_\pi),\ \text{wobei } u,v \notin \mathsf{fn}(P) \\
[\![P \mid Q]\!]_\pi & = & [\![P]\!]_\pi \mid [\![Q]\!]_\pi \\
[\![!\,P]\!]_\pi & = & !\,[\![P]\!]_\pi \\
[\![\nu x.P]\!]_\pi & = & \nu x.[\![P]\!]_\pi
\end{array}
$$

Beispiel

Auswertung von $[\![\overline{x}\langle z\rangle.P \mid x(y).Q]\!]_\pi$:

$$
\begin{array}{l}
[\![\overline{x}\langle z\rangle.P \mid x(y).Q]\!]_\pi \\
= \nu u.(\overline{x}\langle u\rangle.\mathbf{0} \mid u(v).(\overline{v}\langle z\rangle.\mathbf{0} \mid [\![P]\!]_\pi)) \mid x(u').\nu v'.(\overline{u'}\langle v'\rangle.\mathbf{0} \mid v'(y).[\![Q]\!]_\pi) \\
\to \nu u.(\mathbf{0} \mid u(v).(\overline{v}\langle z\rangle.\mathbf{0} \mid [\![P]\!]_\pi) \mid \nu v'.(\overline{u}\langle v'\rangle.\mathbf{0} \mid v'(y).[\![Q]\!]_\pi)) \\
\to \nu v'.(\nu u.(\mathbf{0} \mid (\overline{v'}\langle z\rangle.\mathbf{0}) \mid [\![P]\!]_\pi \mid \mathbf{0} \mid v'(y).[\![Q]\!]_\pi)) \\
\to \nu v'.(\nu u.(\mathbf{0} \mid \mathbf{0} \mid [\![P]\!]_\pi \mid \mathbf{0} \mid [\![Q]\!]_\pi[z/y])) \\
\equiv [\![P]\!]_\pi \mid [\![Q]\!]_\pi[z/y]
\end{array}
$$

- P kann weiter rechnen bevor z gesendet wurde
- Aber Kommunikation ist sichergestellt, nur Q kennt v'

Resultate aus Boudol 1992

Notation für geschlossene Prozesse

Definition

Ein Ausgabe-geschlossener Prozess des synchronen bzw. asynchronen π-Kalküls ist ein Prozess, so dass für jeden Output-Präfix $\overline{x}\langle y\rangle$ gilt: Die Namen x und y sind durch Binder gebunden.

Notation für Antworten

Definition

Ein Prozess P ist eine Eingabe-Antwort, wenn gilt $P \equiv \nu x_1.\ldots.\nu x_n.x(y).P' \mid Q$

Resultate aus Boudol 1992 (2)

Resultat für die Konvergenz:

Theorem

Sei P ein Ausgabe-geschlossener Prozess des synchronen π-Kalküls. Dann gilt:

$$\exists \text{ Eingabe-Antwort } P'\text{: } P \xrightarrow{*} P'$$
genau dann, wenn
$$\exists \text{ Eingabe-Antwort } P''\text{: } [\![P]\!]_\pi \xrightarrow{*} P''$$

- Boudol zeigt noch mehr: Adäquatheit bezüglich der kontextuellen Präordnungen

Resultat von Palamidessi

- Synchroner π-Kalkül mit Summen ist nicht (sinnvoll) kodierbar in den asychronen π-Kalkül ohne Summen

Voraussetzungen:

- Übersetzung ist kompositional und $[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$
- Übersetzung ist wohl-verhaltend bezüglich Umbennenungen: $[\![\sigma(P)]\!] = \sigma([\![P]\!])$.

Nichtdeterministische Auswahl

- $P + Q$, verhält sich wie P oder Q
- Oft beschränkt: "guarded choice": Beim Auswählen muss Kommunikation stattfinden, $\pi_1.P + \pi_2.Q$
- Wir: Synchroner π-Kalkül mit Summen
- Zusätzlicher Präfix: τ = nicht-beobachtbare Aktion.
- Wir verwenden das Summenzeichen Σ und $+$

$$\pi ::= x(y) \mid \overline{x}\langle y \rangle \mid \tau \quad \text{wobei } x, y \in \mathcal{N}$$

$$P ::= \pi.P \mid P_1 \mid P_2 \mid !\,P \mid \mathbf{0} \mid \nu x.P \mid \textstyle\sum_i \pi.P_i \quad \text{für } x \in \mathcal{N}$$

- Strukturelle Kongruenz, neu: $P + Q \equiv Q + P$, $((P_1 + P_2) + P_3) \equiv (P_1 + (P_2 + P_3))$, $P + P \equiv P$.

Reduktion

(SILENT)	$\tau.P \to P$
(SILENTSUM)	$\tau.P + M \to P$
(INTERACTSUM)	$x(y).P + M \mid \overline{x}\langle v \rangle.Q + N \to P[v/y] \mid Q$
(INTERACT)	$x(y).P \mid \overline{x}\langle v \rangle.Q \to P[v/y] \mid Q$
(PAR)	$P \mid Q \to P' \mid Q$, falls $P \to P'$
(NEW)	$\nu x.P \to \nu x.P'$, falls $P \to P'$
(STRUCTCONGR)	$P \to P'$, falls $Q \to Q'$ und $P \equiv Q$ und $P' \equiv Q'$

Polyadischer π-Kalkül

- bisher: monadischer π-Kalkül: Nur ein Name wird kommuniziert
- polyadischer π-Kalkül: Tupel von Namen wird ausgetauscht
- Die Action-Präfixe π im polyadischen π-Kalkül:

$$\pi ::= x(\overrightarrow{y}) \mid \overline{x}\langle \overrightarrow{z} \rangle \text{ wobei } x, y_i, z_i \in \mathcal{N}$$

- INTERACT-Regel:

  (INTERACT) $x(\overrightarrow{y}).P \mid \overline{x}\langle \overrightarrow{z} \rangle.Q \to P[\overrightarrow{z}/\overrightarrow{y}] \mid Q$ falls $|\overrightarrow{y}| = |\overrightarrow{z}|$

- $P[\overrightarrow{z}/\overrightarrow{y}]$ meint hierbei $P[z_1/y_1, \ldots, z_n, y_n]$ wenn $|\overrightarrow{z}| = n$
- Verboten: Gleicher Kanal mit verschiedenen Tupellängen benutzen, z.B.$x(a, b).P \mid x(a, b, c).Q$
- Formal über Sorten (wie ein Typsystem)

Übersetzung: Polyadischer π-Kalkül in monadischen

- Idee: Schicke Elemente aus Tupel $(z_1,\ldots,z_n)$ nacheinander
- Übersetze $x(y_1,\ldots,y_n).P$ als $x(y_1).\ldots.x(y_n).P'$
- und übersetze $\overline{x}\langle(z_1,\ldots,z_n)\rangle.Q$ als $\overline{x}\langle z_1\rangle.\ldots.\overline{x}\langle z_n\rangle.Q'$

Funktioniert nicht, $P := \overline{x}\langle z_1,z_2\rangle.\mathbf{0} \mid \overline{x}\langle z_3,z_4\rangle.\mathbf{0} \mid x(y_1,y_2).\overline{y_1}\langle y_2\rangle.\mathbf{0}$
Möglichkeiten P zu reduzieren:

- $P \to \mathbf{0} \mid \overline{x}\langle z_3,z_4\rangle.\mathbf{0} \mid \overline{z_1}\langle z_2\rangle.\mathbf{0}$
- $P \to \overline{x}\langle z_1,z_2\rangle.\mathbf{0} \mid \overline{z_3}\langle z_4\rangle.\mathbf{0}$

Übersetzung $Q := \overline{x}\langle z_1\rangle.\overline{x}\langle z_2\rangle.\mathbf{0} \mid \overline{x}\langle z_3\rangle.\overline{x}\langle z_4\rangle.\mathbf{0} \mid x(y_1).x(y_2).\overline{y_1}\langle y_2\rangle.\mathbf{0}$
Reduktionen für Q:

- $Q \xrightarrow{*} \mathbf{0} \mid \overline{x}\langle z_3\rangle.\overline{x}\langle z_4\rangle.\mathbf{0} \mid \overline{z_1}\langle z_2\rangle.\mathbf{0}$
- $Q \xrightarrow{*} \overline{x}\langle z_1\rangle.\overline{x}\langle z_2\rangle.\mathbf{0} \mid \mathbf{0} \mid \overline{z_3}\langle z_4\rangle.\mathbf{0}$
- $Q \xrightarrow{*} \overline{x}\langle z_2\rangle.\mathbf{0} \mid \overline{x}\langle z_4\rangle.\mathbf{0} \mid \overline{z_1}\langle z_3\rangle.\mathbf{0}$
- $Q \xrightarrow{*} \overline{x}\langle z_2\rangle.\mathbf{0} \mid \overline{x}\langle z_4\rangle.\mathbf{0} \mid \overline{z_3}\langle z_1\rangle.\mathbf{0}$

Übersetzung: Polyadischer π-Kalkül in monadischen (2)

- Korrektur: Sender und Empfänger tauschen privaten Namen aus

Korrekte Übersetzung

$$
\begin{array}{lcl}
[\![x(z_1,\ldots,z_n).P]\!]_{poly} & = & x(w).w(z_1).\ldots.w(z_n).[\![P]\!]_{poly} \\
[\![\overline{x}\langle z_1,\ldots,z_n\rangle.P]\!]_{poly} & = & \nu v.\overline{x}\langle v\rangle.\overline{v}\langle z_1\rangle.\ldots.\overline{v}\langle z_n\rangle.[\![P]\!]_{poly} \\
[\![\nu x.P]\!]_{poly} & = & \nu x.[\![P]\!]_{poly} \\
[\![P \mid Q]\!]_{poly} & = & [\![P]\!]_{poly} \mid [\![Q]\!]_{poly} \\
[\![!\,P]\!]_{poly} & = & !\,[\![P]\!]_{poly} \\
[\![\mathbf{0}]\!]_{poly} & = & \mathbf{0}
\end{array}
$$

Rekursive Definitionen

- Anstelle der Replikation: rekursive Prozessdefinitionen
- Konstantenanwendungen $A(x_1,\ldots,x_n)$ hinzu

$$P := \ldots \mid \not{!}\not{P} \mid \ldots \mid A(x_1,\ldots,x_n) \text{ wobei } x_i \in \mathcal{N}\}$$

- Definition für A außerhalb des Programms

$$A(x_1,\ldots,x_n) = P \quad \text{wobei } \mathrm{fn}(P) \subseteq \{x_1,\ldots,x_n\}$$

- Neue Reduktionsregel:

$$(\textsc{Const}) \quad A(y_1,\ldots,y_n) \to P[y_1/x_1,\ldots,y_n/x_n]$$
falls $A(x_1,\ldots,x_n) = P$ die Definition von A ist

Übersetzung Rekursion in Replikation, polyadisch

- Seien $\{A_1,\ldots A_n\}$ alle definierten Konstanten, mit Definition $A_i(\overrightarrow{x_i}) = P_i$.
- Neue Kanalnamen $a_1,\ldots,a_n$
- Jede Konstantenanwendung $A_i(\overrightarrow{y_i})$ wird durch $\langle A_i(\overrightarrow{y_i})\rangle := \overline{a_i}\langle\overrightarrow{y_i}\rangle.0$ ersetzt
- Definitionen der A_i durch Prozesse darstellen: $A_i' := !\,a_i(\overrightarrow{x_i}).\langle P_i\rangle$.
- Übersetzung:

$$[\![Q]\!]_{rek} := \nu a_1.\ldots.\nu a_n.(\langle Q\rangle \mid A_1' \mid \ldots \mid A_n')$$

- Achtung: Nicht kompositional,z.B. $[\![Q_1 \mid Q_2]\!]_{rek} \neq [\![Q_1]\!]_{rek} \mid [\![Q_2]\!]_{rek}$.

Übersetzung: Replikation in Rekursion

- Übersetzung $[\![\cdot]\!]_{\mathit{repl}}$
- Betrachte $!\,P$ mit $\{x_1,\ldots,x_n\} = \mathsf{fn}(P)$.
- Wähle neue Konstante A_P und übersetze:

$$[\![!\,P]\!]_{\mathit{repl}} := A_P(x_1,\ldots,x_n)$$

- Alle anderen Konstrukte werden homomorph übersetzt.
- Definition für die Konstante A_P:

$$A_P(x_1,\ldots,x_n) = \nu x_1.\ldots\nu x_n.([\![P]\!]_{\mathit{repl}} \mid A_P(x_1,\ldots,x_n))$$

Beispiel

$$Q = !\,\overline{x}\langle z\rangle.\mathbf{0} \mid x(y).\mathbf{0} \mid x(w).\mathbf{0}$$

Auswertung von Q:

$$Q \equiv !\,\overline{x}\langle z\rangle.\mathbf{0} \mid \overline{x}\langle z\rangle.\mathbf{0} \mid x(y).\mathbf{0} \mid x(w).\mathbf{0}$$
$$\rightarrow !\,\overline{x}\langle z\rangle.\mathbf{0} \mid \mathbf{0} \mid \mathbf{0} \mid x(w).\mathbf{0} \equiv !\,\overline{x}\langle z\rangle.\mathbf{0} \mid \overline{x}\langle z\rangle.\mathbf{0} \mid x(w).\mathbf{0}$$
$$\rightarrow !\,\overline{x}\langle z\rangle.\mathbf{0} \mid \mathbf{0} \mid \mathbf{0} \equiv !\,\overline{x}\langle z\rangle.\mathbf{0}$$

Übersetzung:

$$[\![Q]\!]_{\mathit{repl}} = A(x,z) \mid x(y).\mathbf{0} \mid x(w).\mathbf{0}$$
wobei $A(x,z) = \nu x.\nu z.(\overline{x}\langle z\rangle \mid A(x,z))$

Eine Auswertung von $[\![Q]\!]_{\mathit{repl}}$:

$$\begin{array}{rcl} [\![Q]\!]_{\mathit{repl}} & = & A(x,z) \mid x(y).\mathbf{0} \mid x(w).\mathbf{0} \\ & \rightarrow & (\overline{x}\langle z\rangle \mid A(x,z)) \mid x(y).\mathbf{0} \mid x(w).\mathbf{0} \\ & \rightarrow & A(x,z) \mid \mathbf{0} \mid x(w).\mathbf{0} \\ & \rightarrow & (\overline{x}\langle z\rangle \mid A(x,z)) \mid \mathbf{0} \mid x(w).\mathbf{0} \\ & \rightarrow & A(x,z) \mid \mathbf{0} \mid \mathbf{0} \end{array}$$

Prozess-Gleichheit im π-Kalkül

- Im π-Kalkül verschiedene Begriffe, kein einheitlicher Begriff
- Kein Terminierungsbegriff
- Oft: Terminierung von Prozessen nicht zentral, da verteilte Systeme oft in Endlosschleifen laufen und somit nicht terminieren.
- Gleichheitsbegriff oft: Prozesse haben gleiche Ein- / Ausgabemöglichkeiten
- Statt Reduktion, labeled transition system
- Markierungen stellen gerade die Ein- und Ausgaben des Prozesses dar
- Unterscheidung interne Kommunikation und externe Kommunikation

Labeled Transition System

- Sichtweise der ext. Kommunikation: Wie kann ein Prozess kommunizieren, wenn man einen Kommunikationspartner hinzufügen würde?
- Prozess der Form $P_0 \equiv \nu z_1.\ldots\nu z_n.x(y).P$ mit $x,y \neq z_i$ kann eine Eingabe empfangen
- Prozess der Form $P_0 \equiv \nu z_1.\ldots\nu z_n.\overline{x}\langle y\rangle.P$ mit $x,y \neq z_i$ kann eine Ausgabe durchführen
- Andere Variante: Ausgabe eines gebundenen Namens, d.h. für $P_0 = \nu z_1.\ldots\nu z_n.\nu y..\overline{x}\langle y\rangle.P$ mit $x,y \neq z_i$: Für die Kommunikation muss $\nu y.$ verschoben weren
- Markierungen: $\alpha := \tau \mid x(y) \mid \overline{x}\langle y\rangle \mid \overline{x}(y)$
- $\mathsf{fn}(x(y)) = \{x,y\}$ $\quad$ $\mathsf{bn}(x(y)) = \emptyset$
  $\mathsf{fn}(\overline{x}\langle y\rangle) = \{x,y\}$ $\quad$ $\mathsf{bn}(\overline{x}\langle y\rangle) = \emptyset$
  $\mathsf{fn}(\overline{x}(y)) = \{x\}$ $\quad$ $\mathsf{bn}(\overline{x}(y)) = \{y\}$
  $\mathsf{fn}(\tau) = \emptyset$ $\quad$ bn

Labeled Transition System (2)

Für synchronen π-Kalkül:

(In) $x(y).P \xrightarrow{x(z)} P[z/y]$

(Out) $\overline{x}\langle y\rangle.P \xrightarrow{\overline{x}\langle y\rangle} P$

(Open) $\nu y.P \xrightarrow{\overline{x}(y)} Q$, falls $P \xrightarrow{\overline{x}\langle y\rangle} Q$ und $x \neq y$.

(Interact) $x(y).P \mid \overline{x}\langle v\rangle.Q \xrightarrow{\tau} P[x/v] \mid Q$

(Par) $P \mid Q \xrightarrow{\alpha} P' \mid Q$, falls $P \xrightarrow{\alpha} P'$ und $\mathsf{n}(\alpha) \cap \mathsf{fn}(Q) = \emptyset$

(New) $\nu x.P \xrightarrow{\alpha} \nu x.P'$, falls $P \xrightarrow{\alpha} P'$ und $x \notin \mathsf{n}(\alpha)$

(StructCongr) $P \xrightarrow{\alpha} P'$, falls $Q \xrightarrow{\alpha} Q'$ und $P \equiv Q$ und $P' \equiv Q'$

- Summen, τ usw. können hinzugefügt werden

Harmony-Lemma

$P \to Q$ gdw. $P \xrightarrow{\tau} Q'$ wobei $Q' \equiv Q$

Starke Bisimulation

Definition

Eine binäre Relation $\mathcal{R}$ auf Prozessen ist eine starke Bisimulation wenn für alle $(P, Q) \in \mathcal{R}$ gilt

- Falls $P \xrightarrow{\alpha} P'$, dann gibt es einen Prozess Q' so dass gilt $Q \xrightarrow{\alpha} Q'$ und $(P', Q') \in \mathcal{R}$
- Falls $Q \xrightarrow{\alpha} Q'$, dann gibt es einen Prozess P' so dass gilt $P \xrightarrow{\alpha} P'$ und $(P', Q') \in \mathcal{R}$.

Zwei Prozesse P, Q sind stark bisimilar (geschrieben $P \sim_{b,strong} Q$), falls es eine starke Bisimulation R mit $(P, Q) \in R$ gibt. Die Relation $\sim_{b,strong}$ heißt starke Bisimilarity.

Lemma

$\sim_{b,strong}$ ist die größte starke Bisimulation.

Starke Bisimulation (2)

Starke Bisimilarity ist eine Äquivalenzrelation, aber keine Kongruenz:

- $P = \overline{z}\langle b\rangle.\mathbf{0} \mid a(c).\mathbf{0}$
- $Q = \overline{z}\langle b\rangle.a(c).\mathbf{0} + a(c).\overline{z}\langle b\rangle.\mathbf{0}$
- Es gilt $P \sim_{b,strong} Q$: $\mathcal{R} = \{(P, Q), (a(c).\mathbf{0}, a(c).\mathbf{0}), (\overline{z}\langle b\rangle.\mathbf{0}, \overline{z}\langle b\rangle.\mathbf{0}), (\mathbf{0}, \mathbf{0})\}$ ist eine starke Bisimulation
- Allerdings gilt $x(z).P \not\sim_{b,strong} x(z).Q$
- Damit gilt nicht $P \sim_{b,strong} Q \implies C[P] \sim_{b,strong} C[Q]$ für jeden Kontext C, d.h. $\sim_{b,strong}$ ist keine Kongruenz.

Starke volle Bisimulation

Definition

Zwei Prozesse P, Q sind stark voll bisimilar (Notation $P \sim_{b,strong,full} Q$), wenn für alle Substitutionen σ, die Namen für Namen ersetzen gilt: $\sigma(P) \sim_{b,strong} \sigma(Q)$.

- Man kann nachweisen, dass $\sim_{b,strong,full}$ eine Kongruenz ist.

Bisimulation

- Vernachlässigung von τ-Transitionen
- $\xrightarrow{\tau,*}$:= reflexiv-transitive Hülle von $\xrightarrow{\tau}$

Eine binäre Relation $\mathcal{R}$ ist eine Bisimulation wenn für alle $(P, Q) \in \mathcal{R}$ gilt

- Falls $P \xrightarrow{\alpha} P'$, $\alpha \neq \tau$ dann gibt es Q', Q'' so dass gilt $Q \xrightarrow{\alpha} Q'$, $Q' \xrightarrow{\tau,*} Q''$ und $(P', Q'') \in \mathcal{R}$
- Falls $Q \xrightarrow{\alpha} Q'$, $\alpha \neq \tau$ dann gibt es P', P'' so dass gilt $P \xrightarrow{\alpha} P'$, $P' \xrightarrow{\tau,*} P''$ und $(P'', Q') \in \mathcal{R}$.
- Falls $P \xrightarrow{\tau} P'$, dann gibt es Q' so dass gilt $Q \xrightarrow{\tau,*} Q'$ und $(P', Q') \in \mathcal{R}$
- Falls $Q \xrightarrow{\tau} Q'$, dann gibt es P' so dass gilt $P \xrightarrow{\tau,*} P'$ und $(P', Q') \in \mathcal{R}$

Zwei Prozesse P, Q sind bisimilar (geschrieben $P \sim_b Q$), falls es eine Bisimulation $\mathcal{R}$ mit $(P, Q) \in \mathcal{R}$ gibt. Die Relation $\sim_b$ heißt Bisimilarity.
Zwei Prozesse P, Q sind voll bisimilar ($P \sim_{b,full} Q$) falls $\sigma(P) \sim_b \sigma(Q)$ für alle Substitutionen σ gilt.

Bisimulation (2)

- Bisimilarity ist keine Kongruenz
- Volle Bismilarity ist eine Kongruenz.

Satz

Für alle Prozesse P, Q gilt:

- $P \sim_{b,strong} Q \implies P \sim_b Q$
- $P \sim_{b,strong,full} Q \implies P \sim_{b,full} Q$.

Weiterhin gilt $\sim_{b,strong} \neq \sim_b$ und $\sim_{b,strong,full} \neq \sim_{b,full}$.

Barbs

Für Prozess P schreiben wir

- $P \downarrow_x$, falls $P \xrightarrow{x(y)} P'$
- $P \downarrow_{\overline{x}}$, falls $P \xrightarrow{\overline{x}\langle y\rangle} P'$ oder $P \xrightarrow{\overline{x}(y)} P'$
- für $\beta \in \{x, \overline{x}\}$: $P \Downarrow_\beta$ falls es einen Prozess Q gibt so dass $P \xrightarrow{\tau,*} Q$ und $Q \downarrow_\beta$

Barbs

Lemma

Für alle Prozesse des synchronen π-Kalküls (mit Summen) gilt:

- $P \downarrow_x$ *genau dann, wenn* $P \equiv \nu v_1 \dots \nu v_n.(x(y).P' + M \mid Q)$ *wobei* $x \neq v_i$.
- $P \downarrow_{\overline{x}}$ *genau dann, wenn* $P \equiv \nu v_1 \dots \nu v_n.(\overline{x}\langle y \rangle.P' + M \mid Q)$ *wobei* $x \neq v_i$.

Deshalb: Man kann auch Reduktion statt labeled Transitionen verwenden!

Barbed Bisimulation

Definition

Eine binäre Relation $\mathcal{R}$ auf Prozessen ist eine barbed Bisimulation falls für alle $(P, Q) \in \mathcal{R}$ gilt:

- Falls $P \downarrow_x$ (bzw. $P \downarrow_{\overline{x}}$), dann $Q \Downarrow_x$ (bzw. $(Q \Downarrow_{\overline{x}})$
- Falls $Q \downarrow_x$ (bzw. $Q \downarrow_{\overline{x}}$), dann $P \Downarrow_x$ (bzw. $(P \Downarrow_{\overline{x}})$
- Falls $P \xrightarrow{\tau} P'$, dann exisitiert Q' mit $Q \xrightarrow{\tau,*} Q'$ und $(P', Q') \in \mathcal{R}$
- Falls $Q \xrightarrow{\tau} Q'$, dann exisitiert P' mit $P \xrightarrow{\tau,*} P'$ und $(P', Q') \in \mathcal{R}$

Zwei Prozesse P, Q sind barbed bisimilar (geschrieben $P \sim_{b,barbed} Q$) falls es eine barbed Bisimulation gibt, die (P, Q) enthält.

Barbed Kongruenz

Definition

Zwei Prozesse P, Q sind barbed kongruent (geschrieben $P \sim_c Q$) falls für alle Kontexte C gilt: $C[P] \sim_{b,barbed} C[Q]$.

Theorem

Für alle Prozesse P, Q *gilt:*
$P \sim_{b,full} Q \implies P \sim_c Q$

Es ist unbekannt, ob die Umkehrung dieser Implikation gilt.

Zusammenfassung

Zusammenfassend lässt sich feststellen, dass es viele verschiedene Gleichheitsbegriffe für den π-Kalkül gibt, wobei die barbed Kongruenz die gröbste der vorgestellten Gleichheiten ist.