Einführung in den Pi-Kalkül - Teil II

Theorie kommunizierender Systeme: Der Pi-Kalkül

Sabine Fischer

Betreuer: Gert Smolka

5. Oktober 2004

Übersicht

Syntaktischer Zucker

- Polyadischer Pi-Kalkül
- Rekursive Definitionen

Abstraktionen

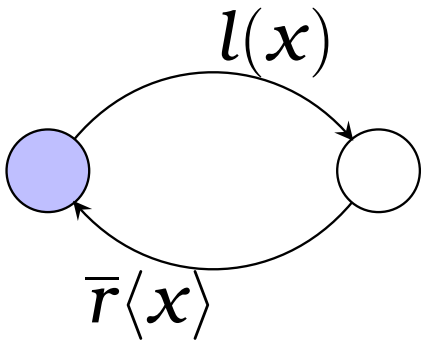
- Bindung von Namen im Pi-Kalkül

Sequentielles Senden

- Einfache Systeme
- Eindeutige Sender
- x-Vergesslichkeit
- Die Invariante

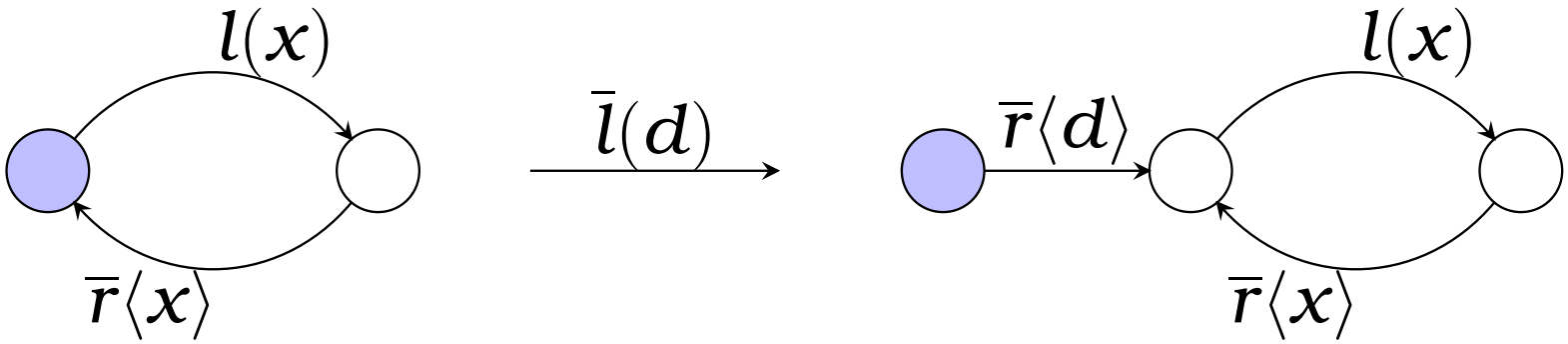
Der einfache Puffer

$$Buf(l,r) \stackrel{def}{=} l(x).\overline{r}\langle x\rangle.Buf(l,r)$$



Der einfache Puffer

$$Buf(l,r) \stackrel{def}{=} l(x).\overline{r}\langle x\rangle.Buf(l,r)$$



Der polyadische Pi-Kalkül

- Ziel: Senden von mehreren Namen über einen Kanal als eine einzige Nachricht

Der polyadische Pi-Kalkül

- Ziel: Senden von mehreren Namen über einen Kanal als eine einzige Nachricht
- Syntax ($\vec{y} = y_1 y_2 \ldots y_n$)
  - $\overline{x}\langle\vec{y}\rangle.P$: sende $\vec{y}$ über x
  - $x(\vec{y}).Q$: empfange $\vec{y}$ über x

Naive Kodierung(funktioniert fast)

- Idee: Sende $y_1, y_2, \ldots, y_n$ nacheinander über x

Naive Kodierung(funktioniert fast)

- Idee: Sende $y_1, y_2, \ldots, y_n$ nacheinander über x

$$\overline{x}\langle\vec{y}\rangle.P \mapsto \overline{x}\langle y_1\rangle.\overline{x}\langle y_2\rangle.\cdots .\overline{x}\langle y_n\rangle.P$$
$$x(\vec{y}).Q \mapsto x(y_1).x(y_2).\cdots .x(y_n).Q$$

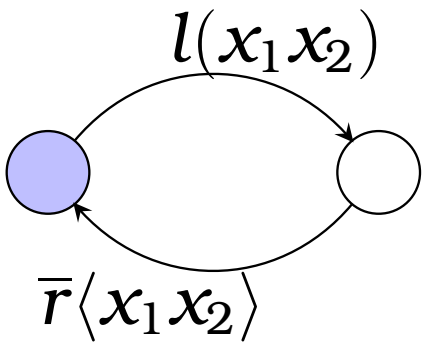
Naive Kodierung(funktioniert fast)

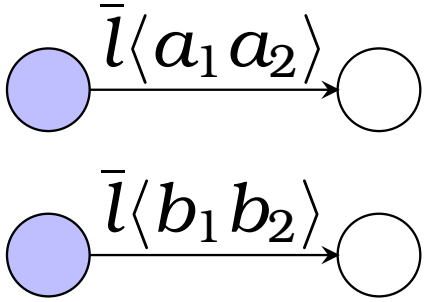
- Idee: Sende $y_1, y_2, \ldots, y_n$ nacheinander über x

$$\overline{x}\langle\vec{y}\rangle.P \mapsto \overline{x}\langle y_1\rangle.\overline{x}\langle y_2\rangle.\cdots .\overline{x}\langle y_n\rangle.P$$
$$x(\vec{y}).Q \mapsto x(y_1).x(y_2).\cdots .x(y_n).Q$$

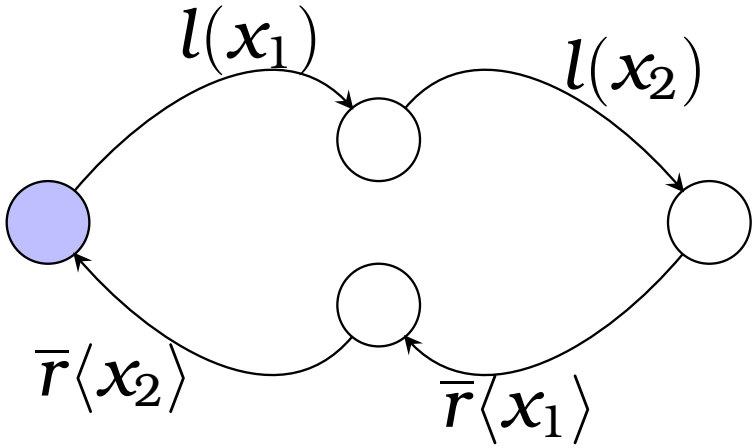
- funktioniert wenn es nur einen Sender und einen Empfänger auf x gibt

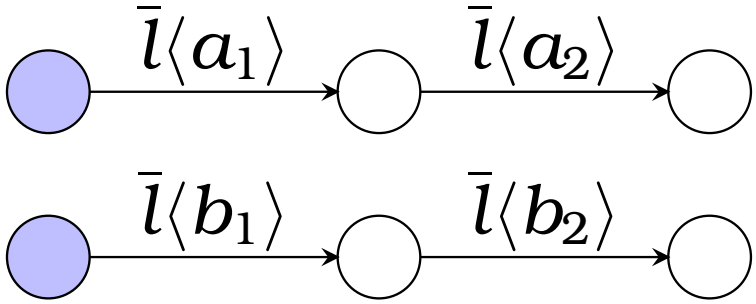
Beispiel: Doppelpuffer, zwei Sender



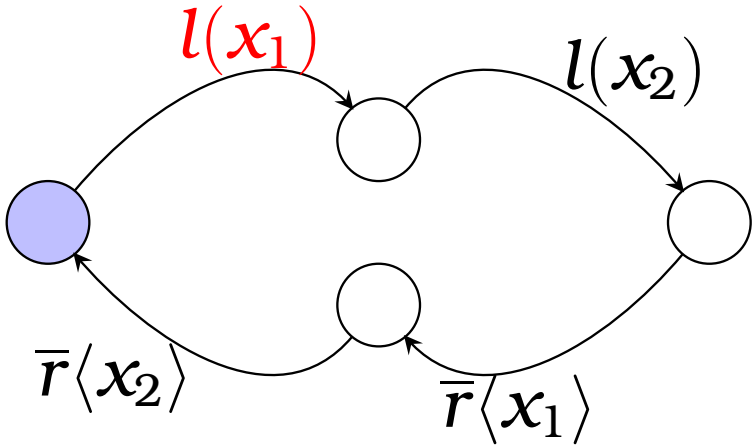


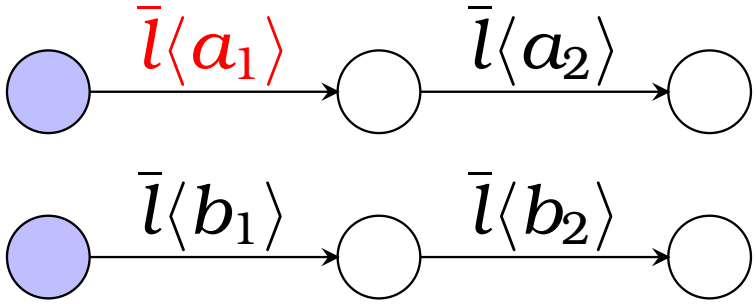
Beispiel: Doppelpuffer, zwei Sender



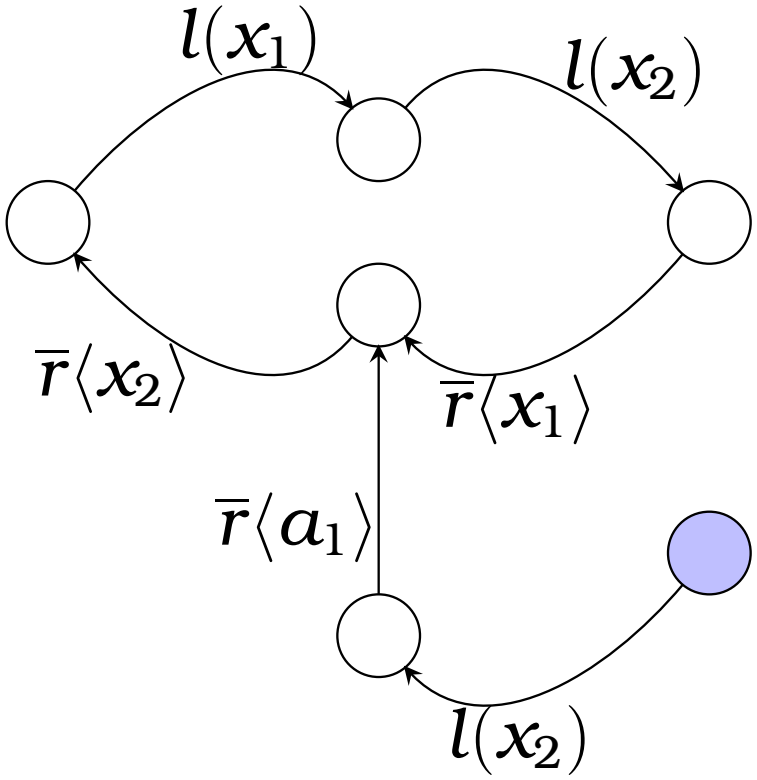


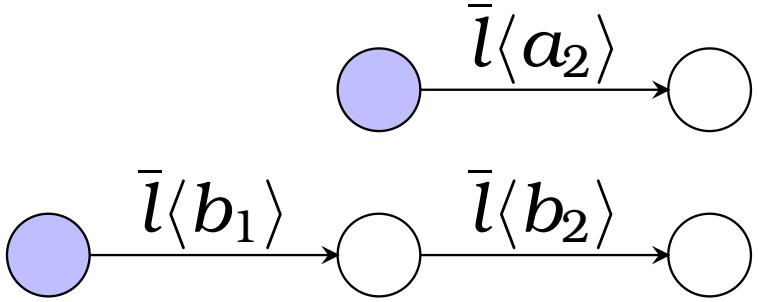
Beispiel: Doppelpuffer, zwei Sender



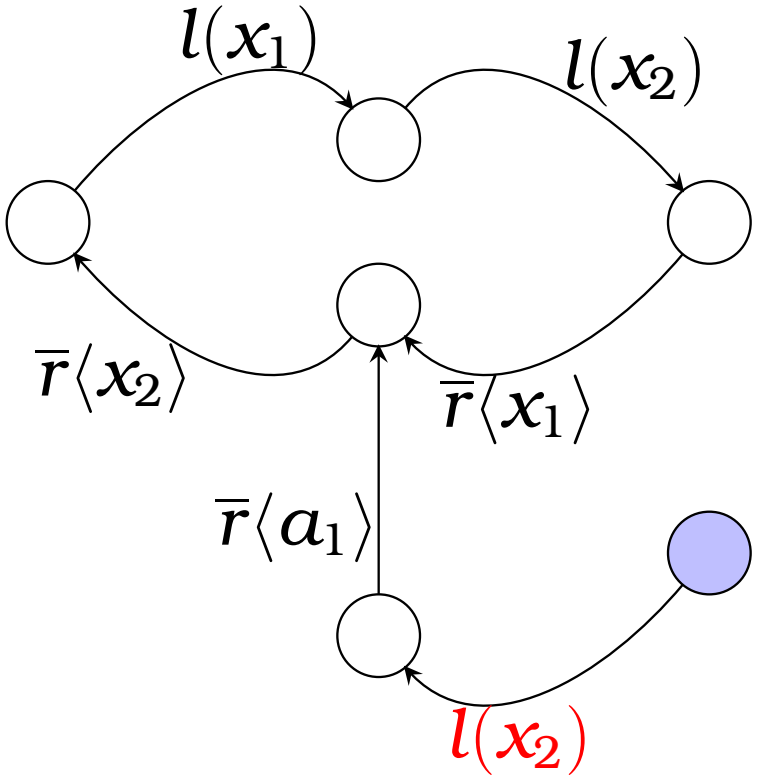


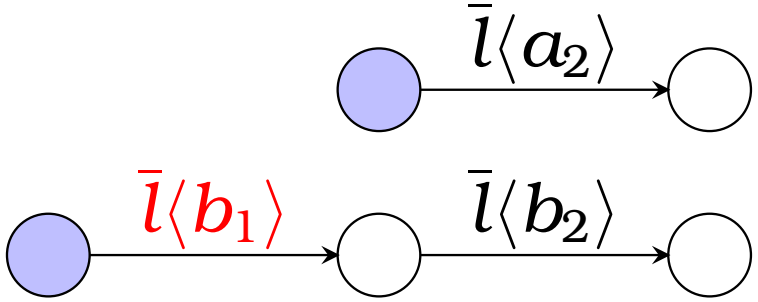
Beispiel: Doppelpuffer, zwei Sender



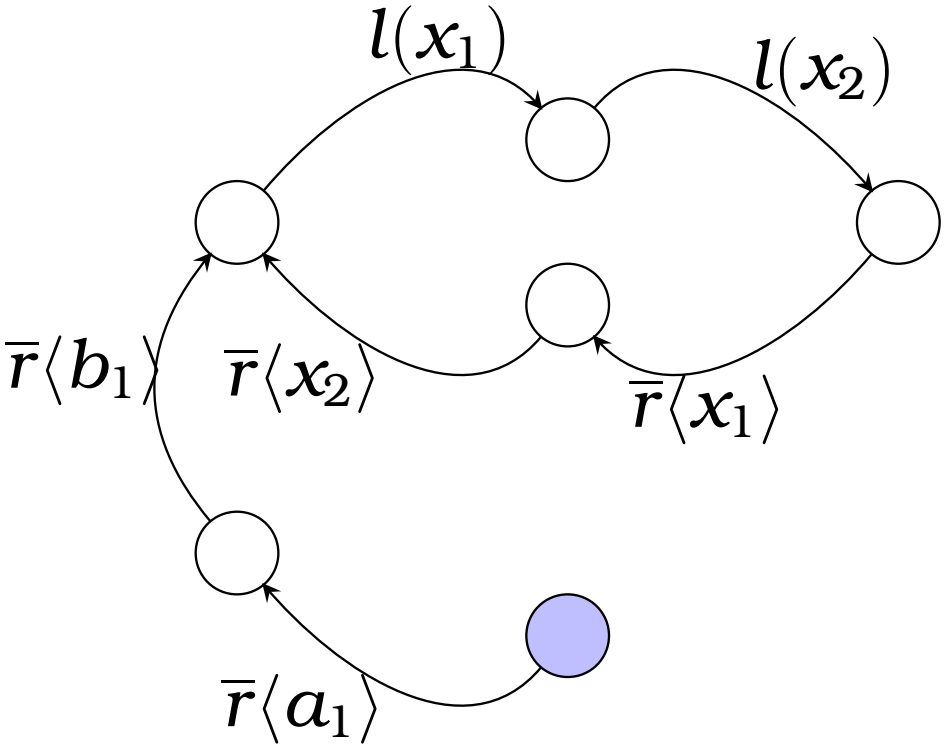


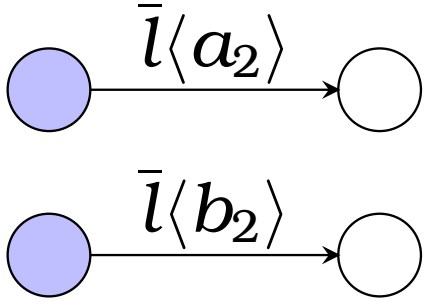
Beispiel: Doppelpuffer, zwei Sender



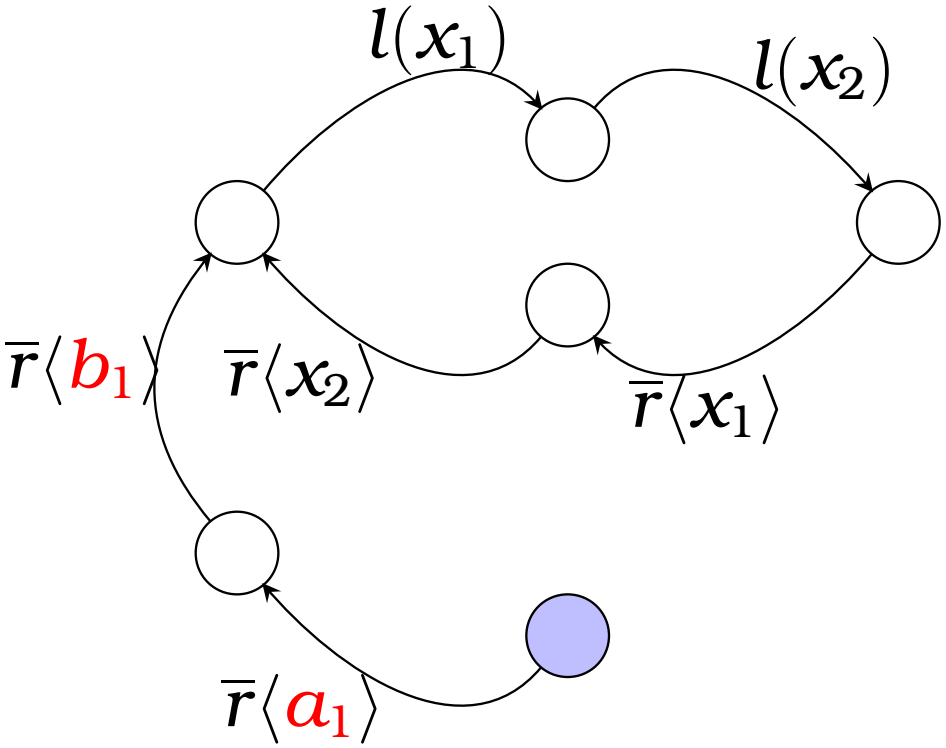


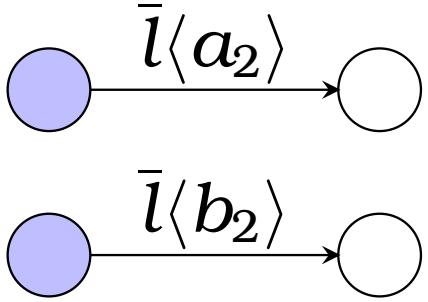
Beispiel: Doppelpuffer, zwei Sender





Beispiel: Doppelpuffer, zwei Sender





Korrekte Kodierung

- Problem: konkurrierende Sender oder Empfänger

Korrekte Kodierung

- Problem: konkurrierende Sender oder Empfänger
- Lösung: schicke erst einen neuen Kanal w über x, sende dann naiv kodiert die Nachricht über w

Korrekte Kodierung

- Problem: konkurrierende Sender oder Empfänger
- Lösung: schicke erst einen neuen Kanal w über x, sende dann naiv kodiert die Nachricht über w

$$
\begin{aligned}
x(\vec{y}).P &\mapsto x(w).w(y_1).w(y_2).\cdots.w(y_n).P \\
\overline{x}\langle\vec{y}\rangle.Q &\mapsto \text{new } w\ (\overline{x}\langle w\rangle.\overline{w}\langle y_1\rangle.\cdots.\overline{w}\langle y_n\rangle.Q)
\end{aligned}
$$

Rekursive Definitionen

- Gegeben: Rekursiv definierte Prozedur $A\langle\vec{x}\rangle = P$

Rekursive Definitionen

- Gegeben: Rekursiv definierte Prozedur $A\langle\vec{x}\rangle = P$
- Ziel: Prozesse, die A "aufrufen", explizit im Pi-Kalkül ausdrücken

Rekursive Definitionen

- Gegeben: Rekursiv definierte Prozedur $A\langle\vec{x}\rangle = P$
- Ziel: Prozesse, die A "aufrufen", explizit im Pi-Kalkül ausdrücken
- Idee: Erfinde einen neuen Namen a der für A stehen soll

Rekursive Definitionen

- Gegeben: Rekursiv definierte Prozedur $A\langle\vec{x}\rangle = P$
- Ziel: Prozesse, die A "aufrufen", explizit im Pi-Kalkül ausdrücken
- Idee: Erfinde einen neuen Namen a der für A stehen soll
- Senden von $\vec{x}$ entlang a repräsentiert den Aufruf $A\langle\vec{x}\rangle$

Übersetzung von definierenden Gleichungen

$$A\langle\vec{x}\rangle = P \quad \mapsto \quad a(\vec{x}).P[A := \overline{a}]$$

Übersetzung von definierenden Gleichungen

$$A\langle\vec{x}\rangle = P \quad \mapsto \quad !a(\vec{x}).P[A := \overline{a}]$$

Übersetzung von definierenden Gleichungen

$$A\langle\vec{x}\rangle = P \quad \mapsto \quad !a(\vec{x}).P[A := \overline{a}]$$

- selbst wenn A nicht rekursiv ist, ist die Replikation notwendig

Übersetzung von definierenden Gleichungen

$$A\langle\vec{x}\rangle = P \quad \mapsto \quad !a(\vec{x}).P[A := \overline{a}]$$

- selbst wenn A nicht rekursiv ist, ist die Replikation notwendig

Ist S ein Prozess der A aufruft, so ergibt sich

$$S \quad \mapsto \quad S[A := \overline{a}] \mid !a(\vec{x}).P[A := \overline{a}]$$

Übersetzung von definierenden Gleichungen

$$A\langle\vec{x}\rangle = P \quad \mapsto \quad !a(\vec{x}).P[A := \overline{a}]$$

- selbst wenn A nicht rekursiv ist, ist die Replikation notwendig

Ist S ein Prozess der A aufruft, so ergibt sich

$$S \quad \mapsto \quad S[A := \overline{a}] \mid !a(\vec{x}).P[A := \overline{a}]$$

Beispiel

$$Buf(l, r) = l(x).\overline{r}\langle x\rangle.Buf(l, r)$$

Übersetzung von definierenden Gleichungen

$$A\langle\vec{x}\rangle = P \quad \mapsto \quad !a(\vec{x}).P[A := \overline{a}]$$

- selbst wenn A nicht rekursiv ist, ist die Replikation notwendig

Ist S ein Prozess der A aufruft, so ergibt sich
$S \quad \mapsto \quad S[A := \overline{a}] \mid !a(\vec{x}).P[A := \overline{a}]$

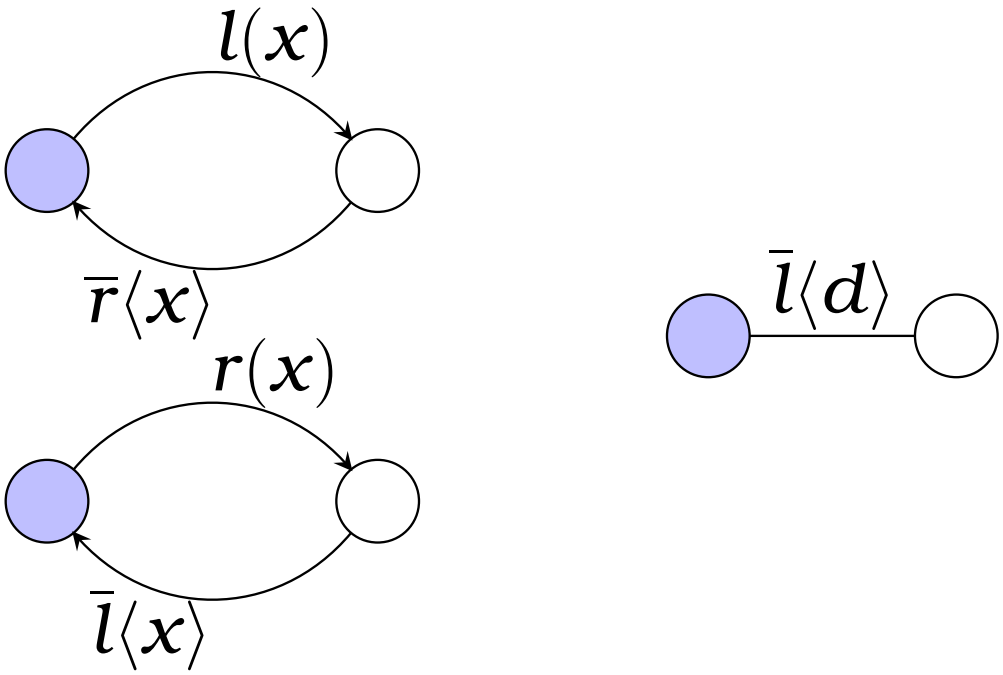
Beispiel

$Buf(l, r) = l(x).\overline{r}\langle x\rangle.Buf(l, r)$
$\mapsto \quad !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$

Beispiel: Ping-Pong-Puffer - Rekursiv

$$Buf(l,r) \stackrel{def}{=} l(x).\overline{r}\langle x\rangle.Buf(l,r)$$

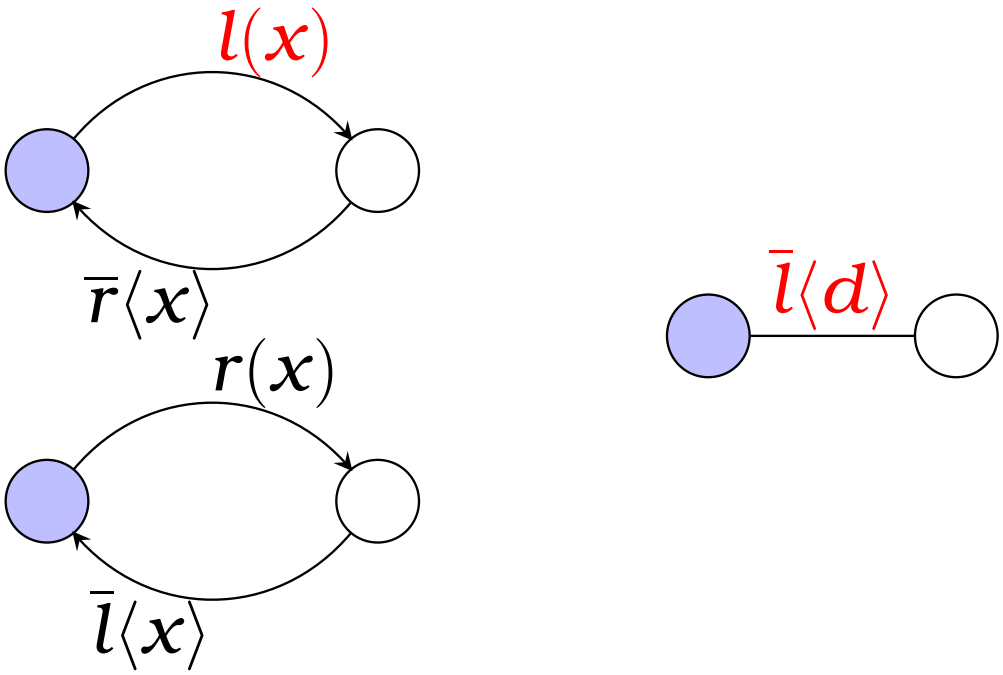
$$Buf(l,r) \mid Buf(r,l) \mid \overline{l}\langle d\rangle$$



Beispiel: Ping-Pong-Puffer - Rekursiv

$$Buf(l,r) \stackrel{def}{=} l(x).\overline{r}\langle x\rangle.Buf(l,r)$$

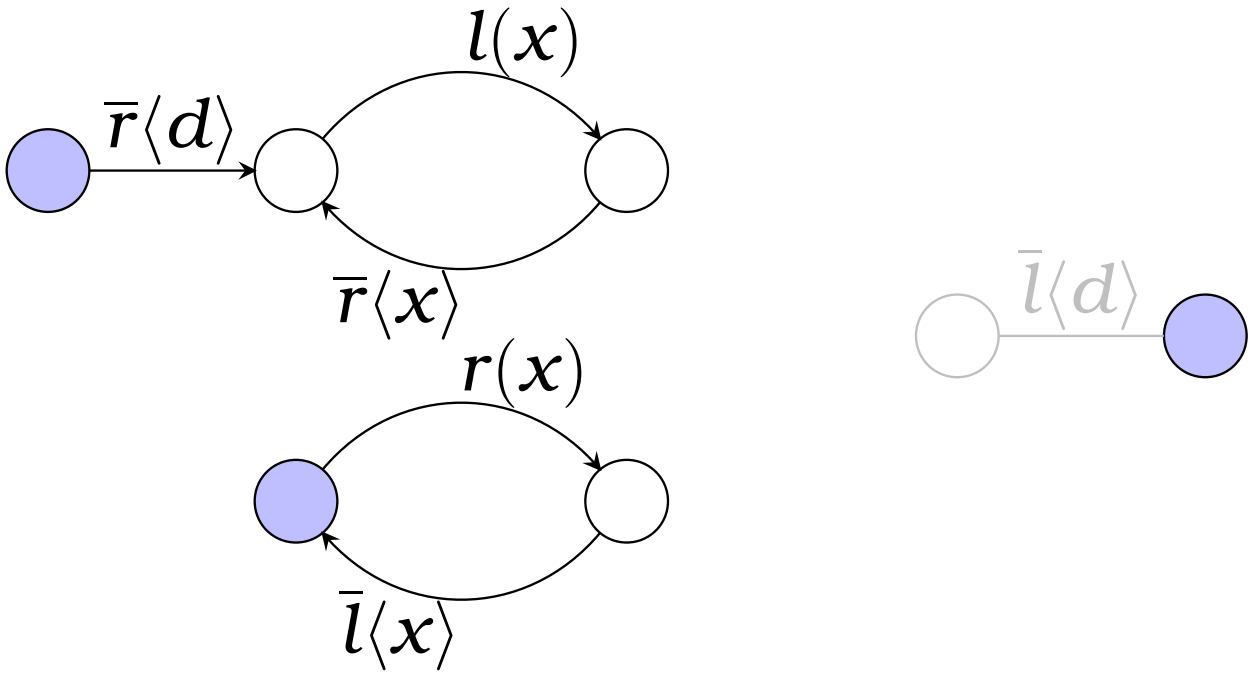
$$Buf(l,r) \mid Buf(r,l) \mid \overline{l}\langle d\rangle$$



Beispiel: Ping-Pong-Puffer - Rekursiv

$$Buf(l,r) \stackrel{def}{=} l(x).\overline{r}\langle x\rangle.Buf(l,r)$$

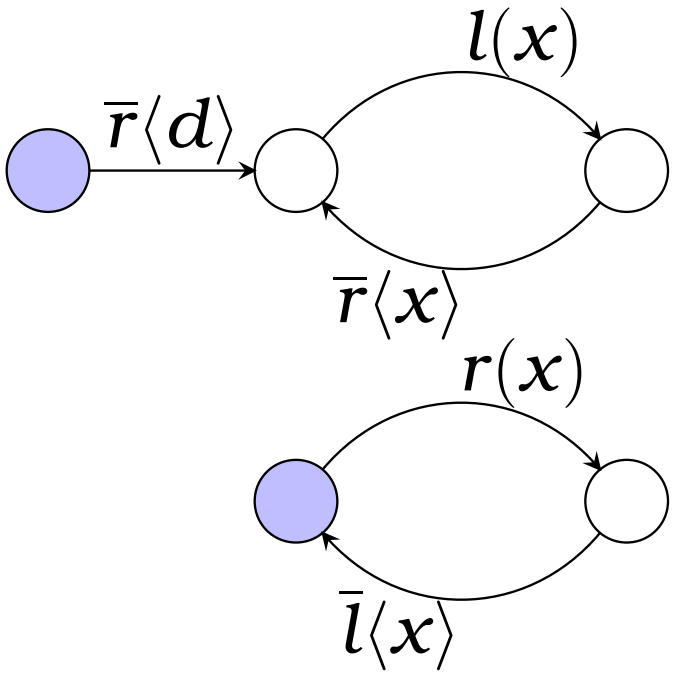
$$Buf(l,r) \mid Buf(r,l) \mid \overline{l}\langle d\rangle$$



Beispiel: Ping-Pong-Puffer - Rekursiv

$$Buf(l,r) \stackrel{def}{=} l(x).\overline{r}\langle x\rangle.Buf(l,r)$$

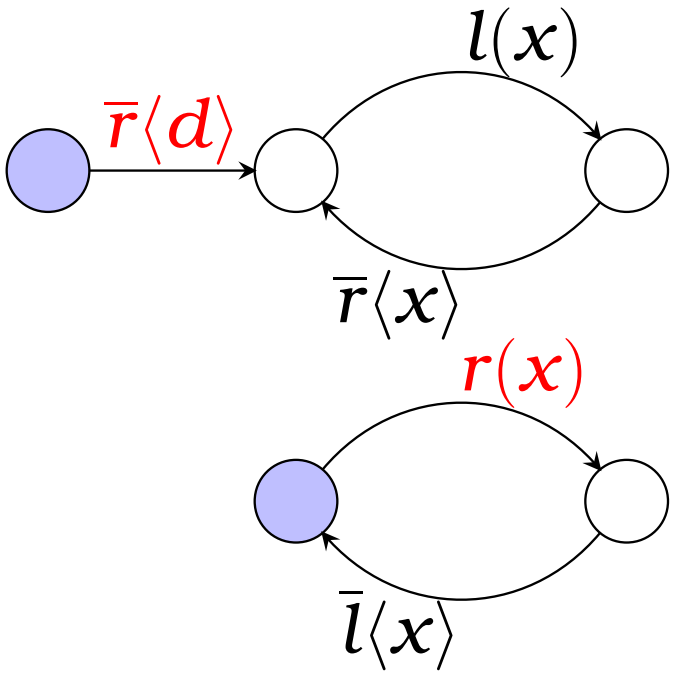
$$Buf(l,r) \mid Buf(r,l) \mid \overline{l}\langle d\rangle$$



Beispiel: Ping-Pong-Puffer - Rekursiv

$$Buf(l,r) \stackrel{def}{=} l(x).\overline{r}\langle x\rangle.Buf(l,r)$$

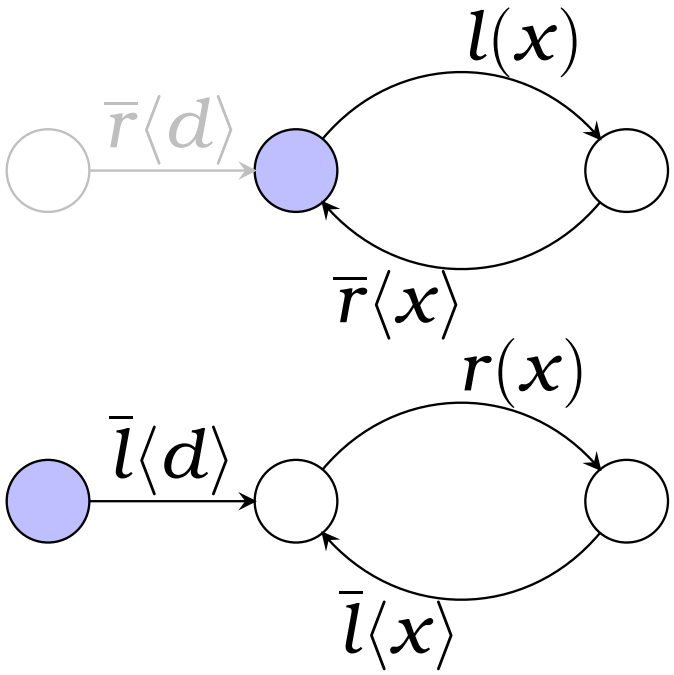
$$Buf(l,r) \mid Buf(r,l) \mid \overline{l}\langle d\rangle$$



Beispiel: Ping-Pong-Puffer - Rekursiv

$$Buf(l,r) \stackrel{def}{=} l(x).\overline{r}\langle x\rangle.Buf(l,r)$$

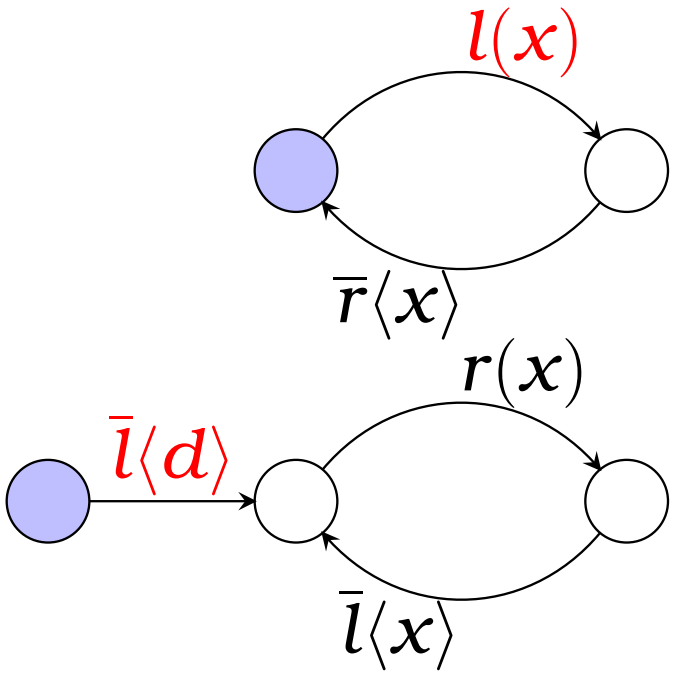
$$Buf(l,r) \mid Buf(r,l) \mid \overline{l}\langle d\rangle$$



Beispiel: Ping-Pong-Puffer - Rekursiv

$$Buf(l,r) \stackrel{def}{=} l(x).\overline{r}\langle x\rangle.Buf(l,r)$$

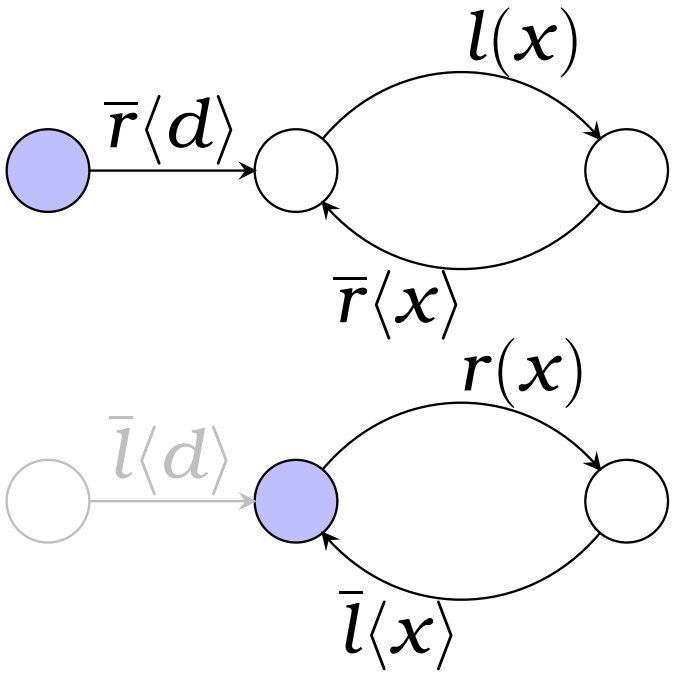
$$Buf(l,r) \mid Buf(r,l) \mid \overline{l}\langle d\rangle$$



Beispiel: Ping-Pong-Puffer - Rekursiv

$$Buf(l,r) \stackrel{def}{=} l(x).\overline{r}\langle x\rangle.Buf(l,r)$$

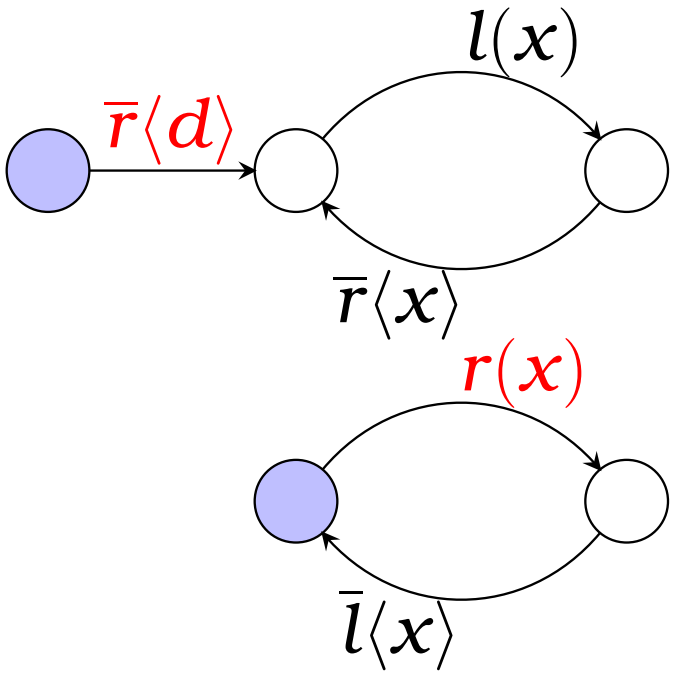
$$Buf(l,r) \mid Buf(r,l) \mid \overline{l}\langle d\rangle$$



Beispiel: Ping-Pong-Puffer - Rekursiv

$$Buf(l,r) \stackrel{def}{=} l(x).\overline{r}\langle x\rangle.Buf(l,r)$$

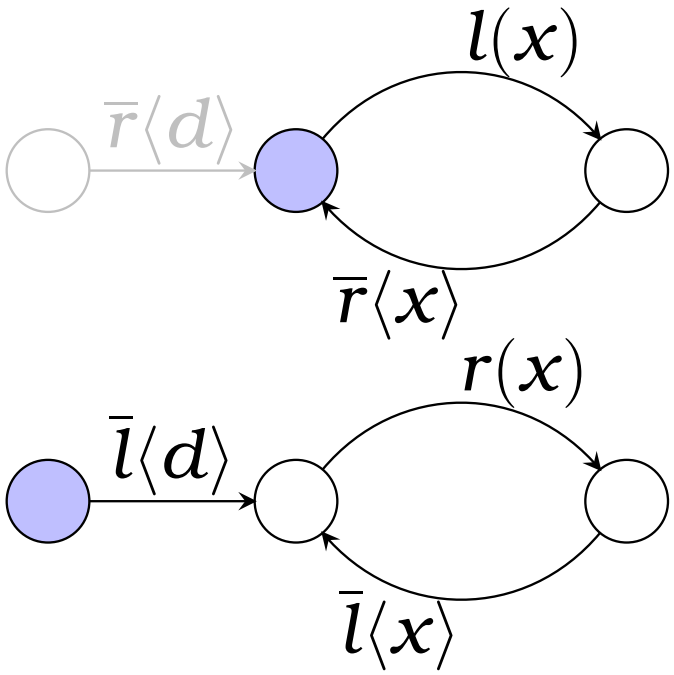
$$Buf(l,r) \mid Buf(r,l) \mid \overline{l}\langle d\rangle$$



Beispiel: Ping-Pong-Puffer - Rekursiv

$$Buf(l,r) \stackrel{def}{=} l(x).\overline{r}\langle x\rangle.Buf(l,r)$$

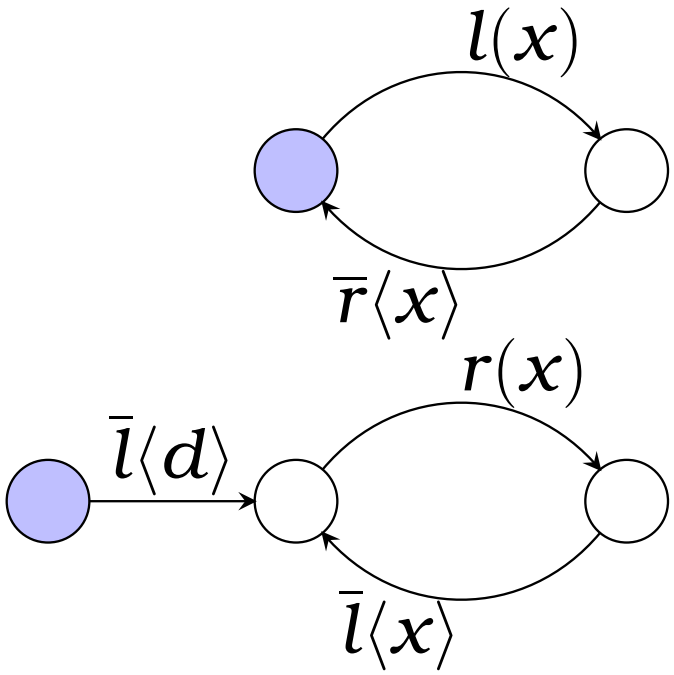
$$Buf(l,r) \mid Buf(r,l) \mid \overline{l}\langle d\rangle$$



Beispiel: Ping-Pong-Puffer - Rekursiv

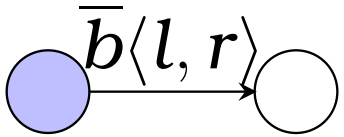
$$Buf(l,r) \stackrel{def}{=} l(x).\overline{r}\langle x\rangle.Buf(l,r)$$

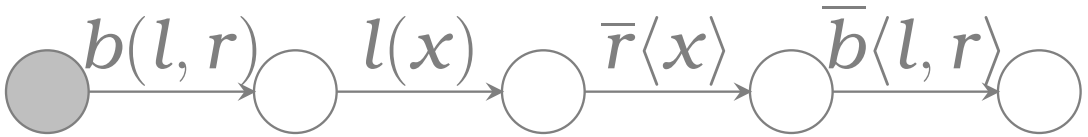
$$Buf(l,r) \mid Buf(r,l) \mid \overline{l}\langle d\rangle$$

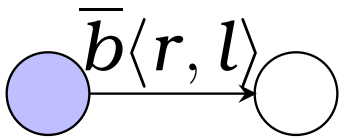


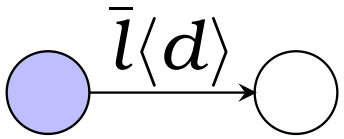
Beispiel: Ping-Pong-Puffer - Prozeduraufrufe

$$\overline{b}\langle l, r\rangle \mid \overline{b}\langle r, l\rangle \mid \overline{l}\langle d\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$$



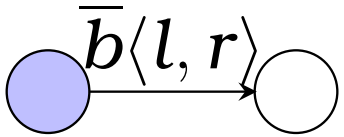


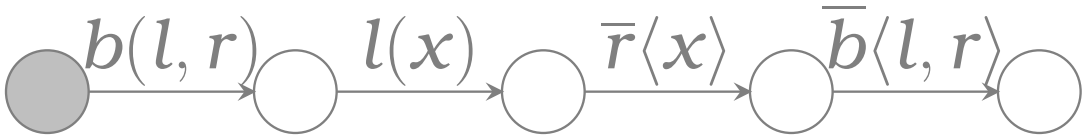


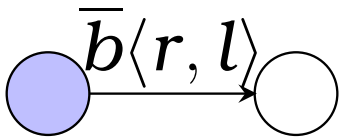


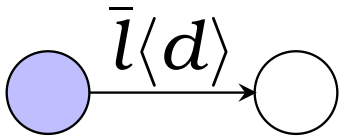
Beispiel: Ping-Pong-Puffer - Prozeduraufrufe

$$\overline{b}\langle l, r\rangle \mid \overline{b}\langle r, l\rangle \mid \overline{l}\langle d\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$$



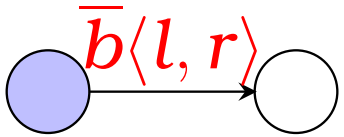


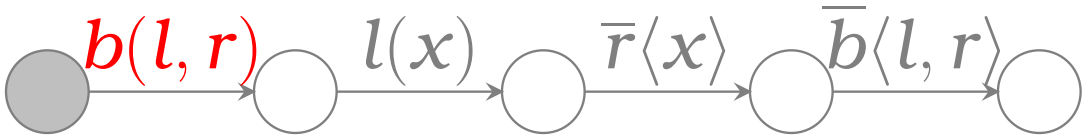


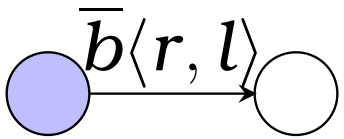


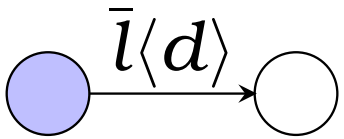
Beispiel: Ping-Pong-Puffer - Prozeduraufrufe

$$\overline{b}\langle l, r\rangle \mid \overline{b}\langle r, l\rangle \mid \overline{l}\langle d\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$$



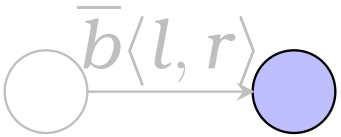


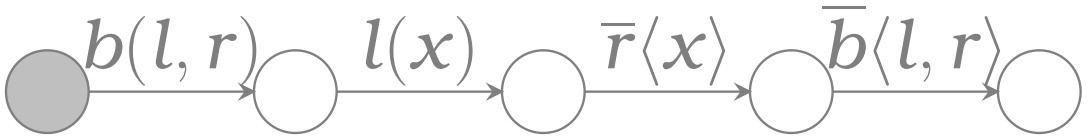


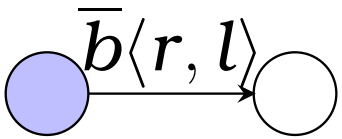


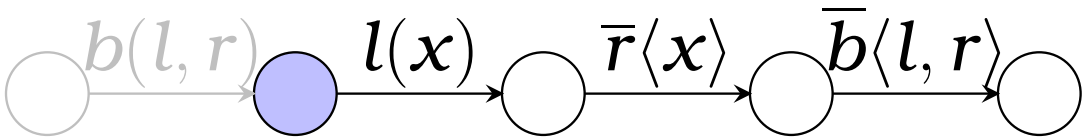
Beispiel: Ping-Pong-Puffer - Prozeduraufrufe

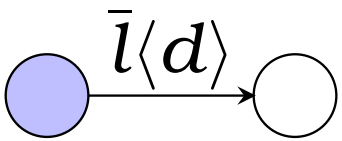
$$\overline{b}\langle l, r\rangle \mid \overline{b}\langle r, l\rangle \mid \overline{l}\langle d\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$$





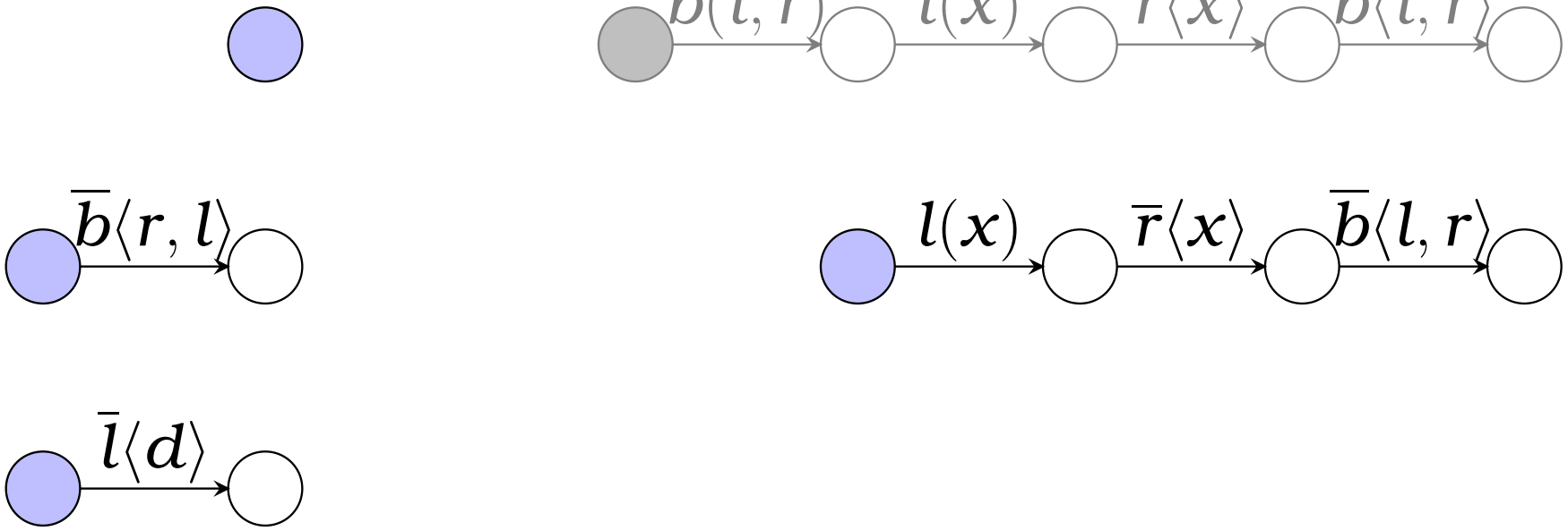






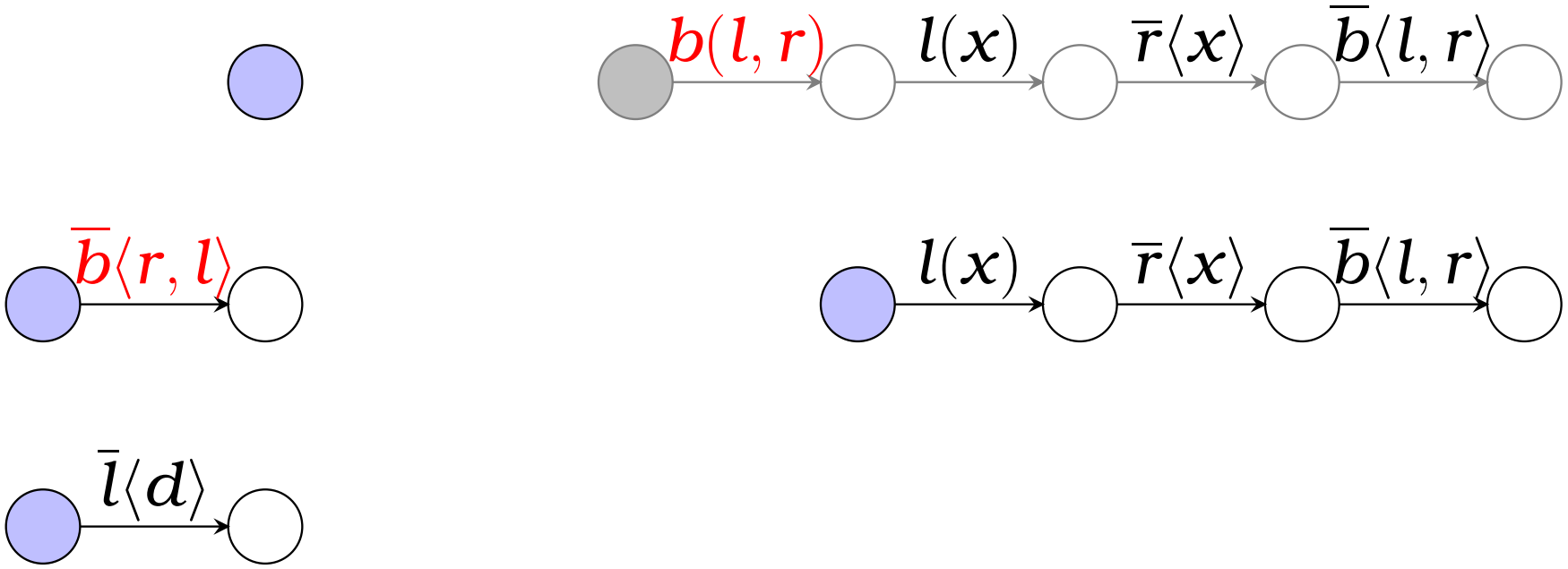
Beispiel: Ping-Pong-Puffer - Prozeduraufrufe

$$\overline{b}\langle l, r\rangle \mid \overline{b}\langle r, l\rangle \mid \overline{l}\langle d\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$$



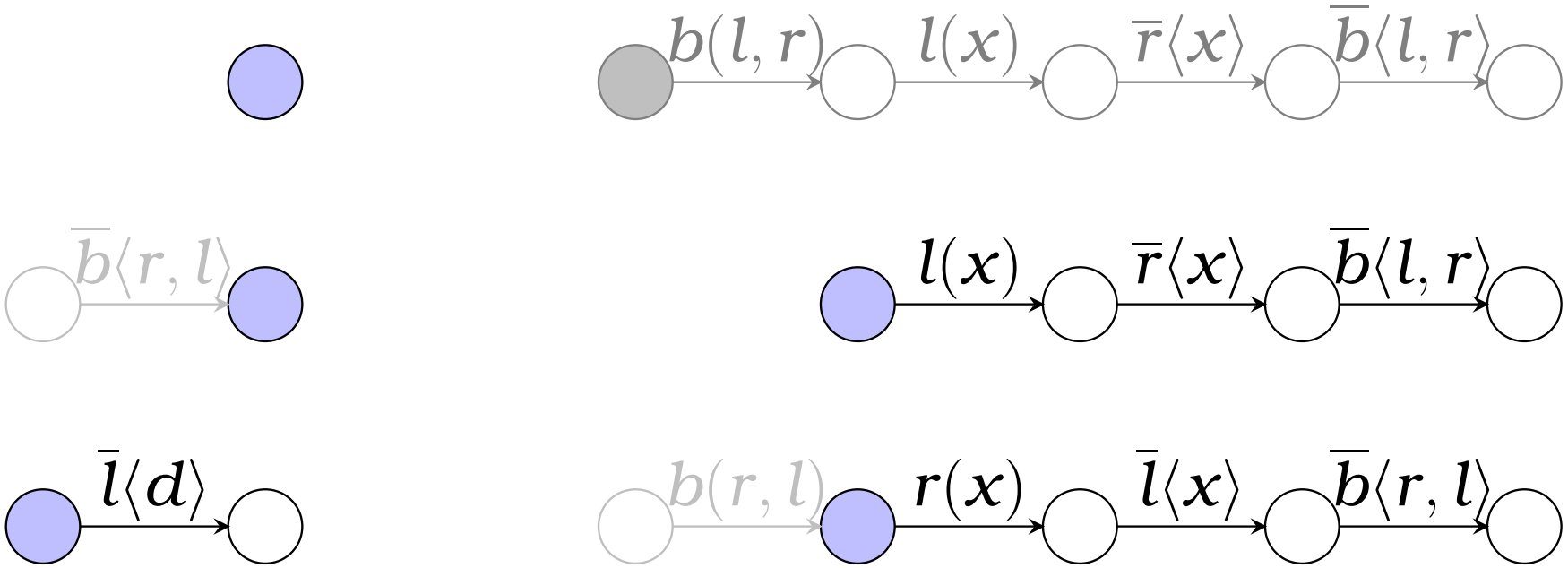
Beispiel: Ping-Pong-Puffer - Prozeduraufrufe

$$\overline{b}\langle l, r\rangle \mid \overline{b}\langle r, l\rangle \mid \overline{l}\langle d\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$$



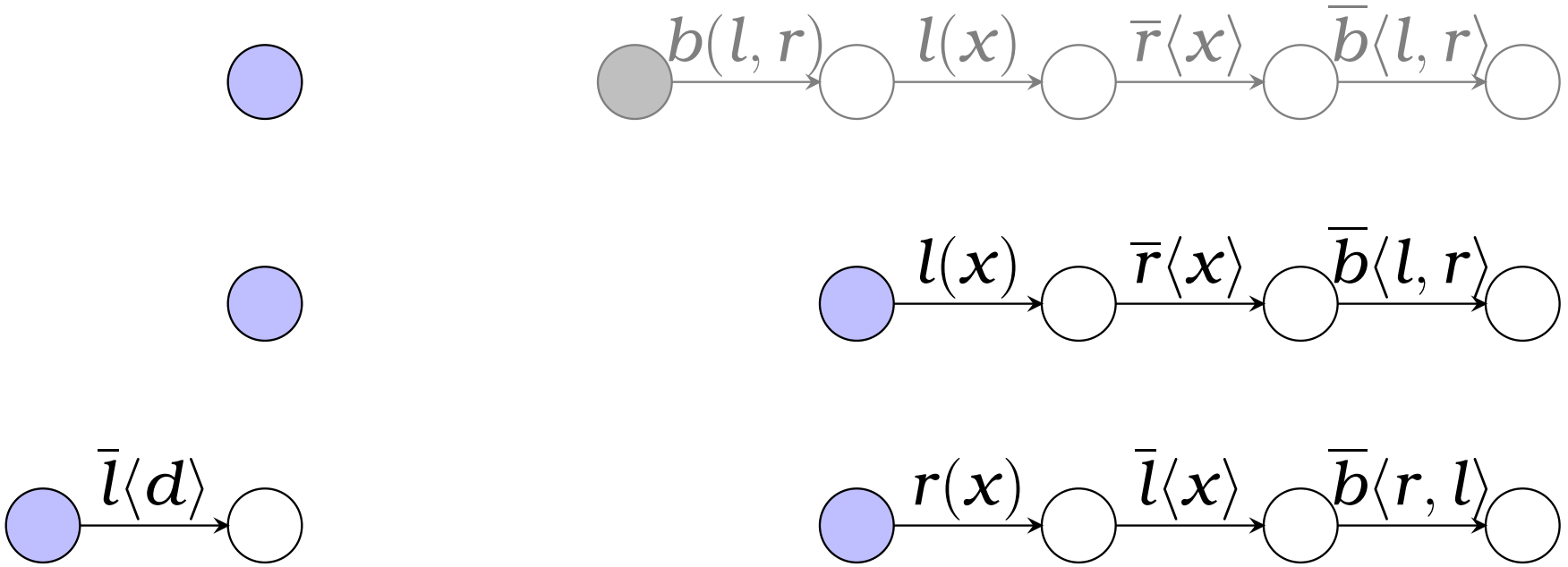
Beispiel: Ping-Pong-Puffer - Prozeduraufrufe

$$\overline{b}\langle l, r\rangle \mid \overline{b}\langle r, l\rangle \mid \overline{l}\langle d\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$$



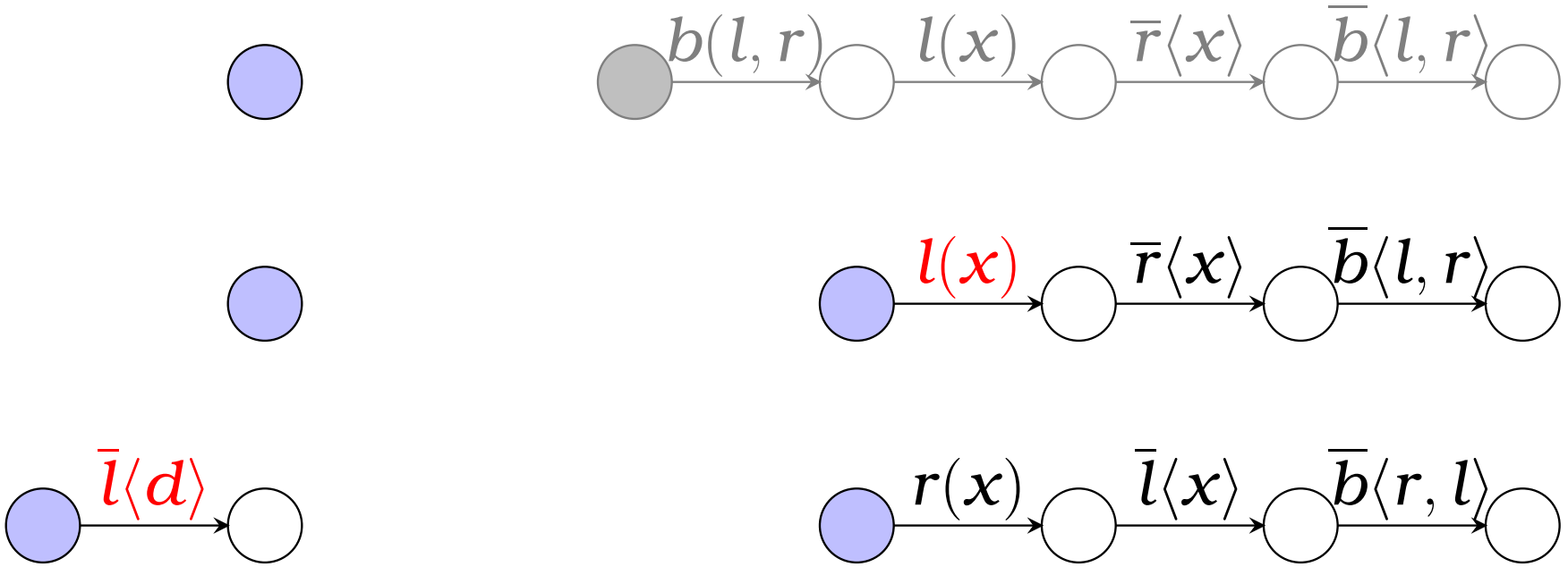
Beispiel: Ping-Pong-Puffer - Prozeduraufrufe

$$\overline{b}\langle l, r\rangle \mid \overline{b}\langle r, l\rangle \mid \overline{l}\langle d\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$$



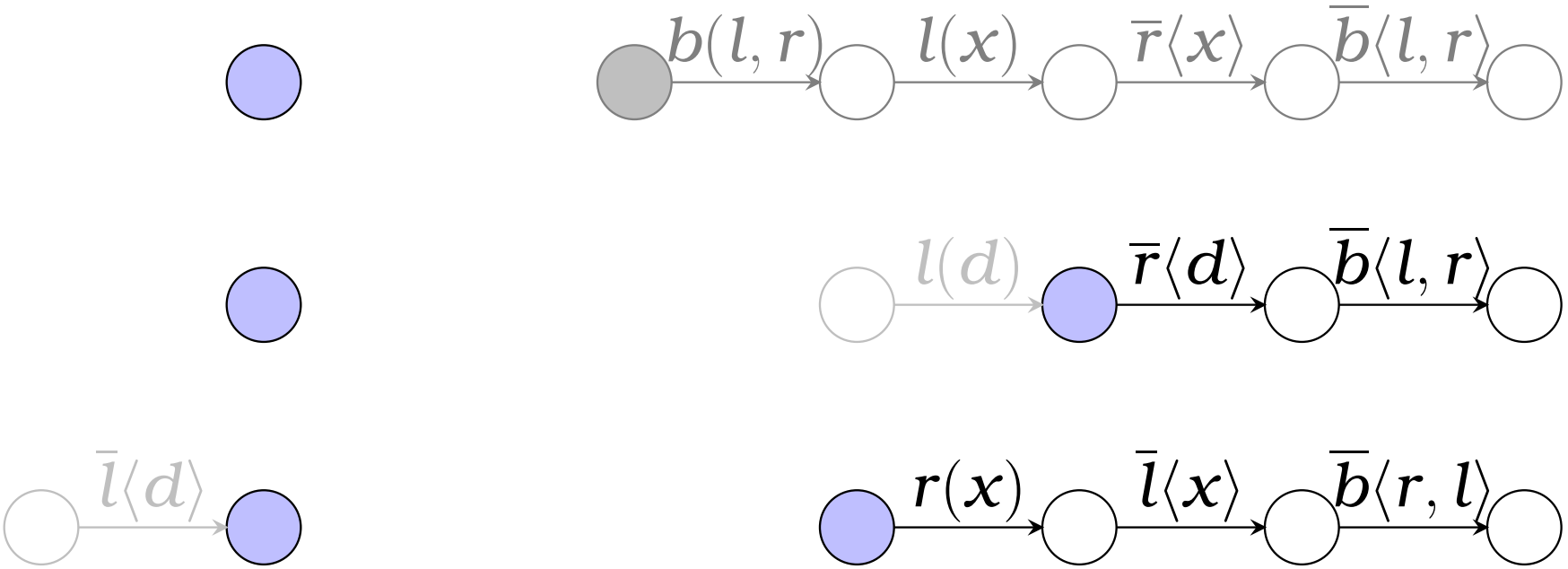
Beispiel: Ping-Pong-Puffer - Prozeduraufrufe

$$\overline{b}\langle l, r\rangle \mid \overline{b}\langle r, l\rangle \mid \overline{l}\langle d\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$$



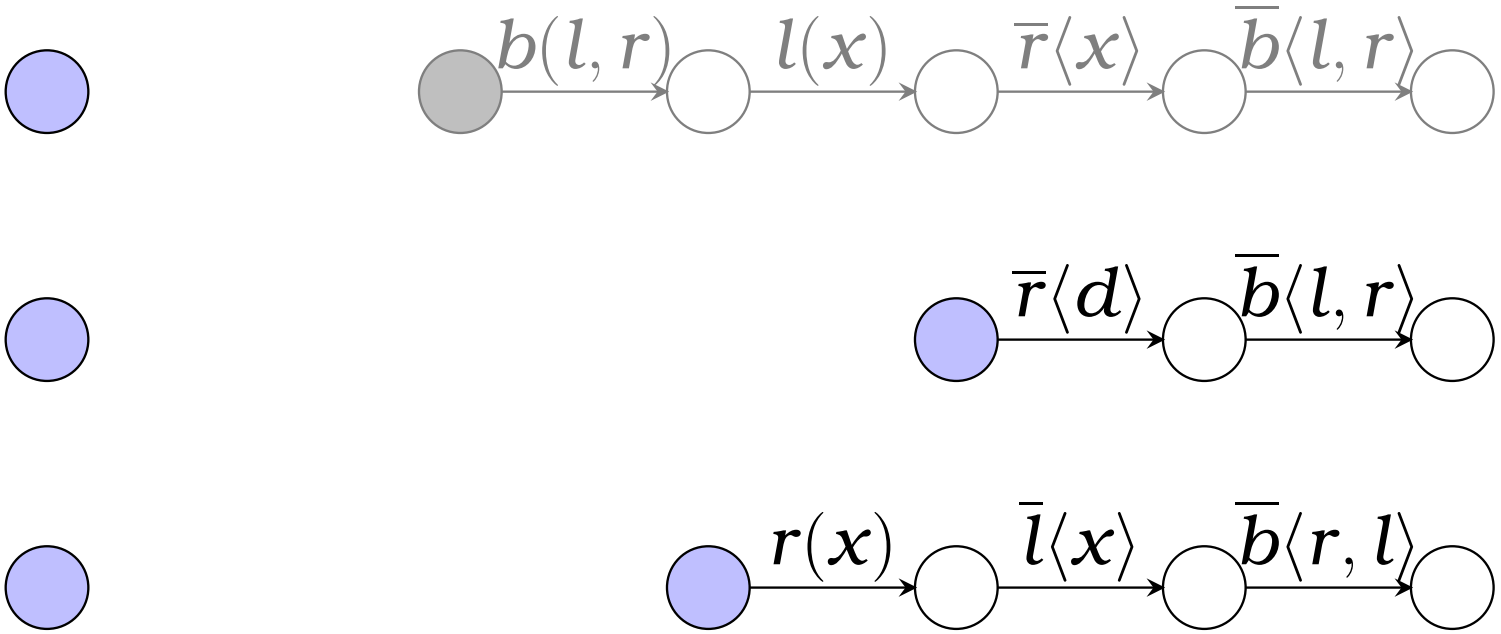
Beispiel: Ping-Pong-Puffer - Prozeduraufrufe

$$\overline{b}\langle l, r\rangle \mid \overline{b}\langle r, l\rangle \mid \overline{l}\langle d\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$$



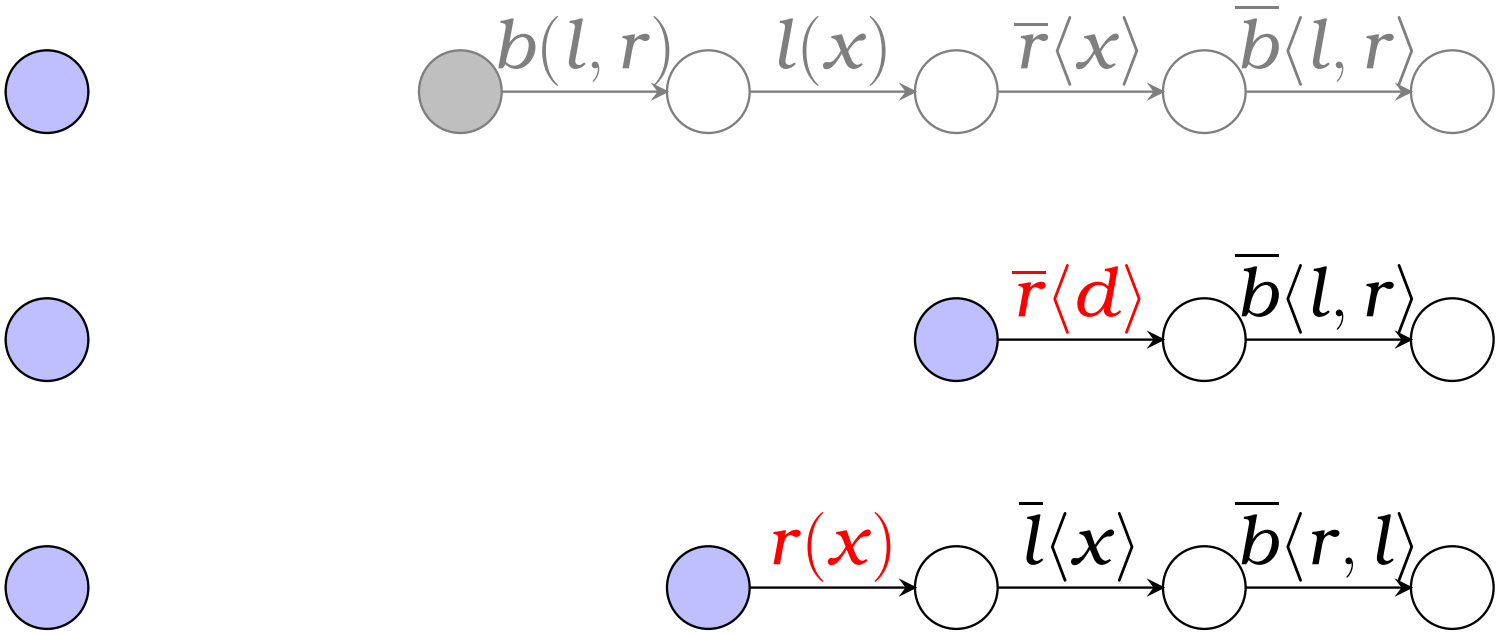
Beispiel: Ping-Pong-Puffer - Prozeduraufrufe

$$\overline{b}\langle l, r\rangle \mid \overline{b}\langle r, l\rangle \mid \overline{l}\langle d\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$$



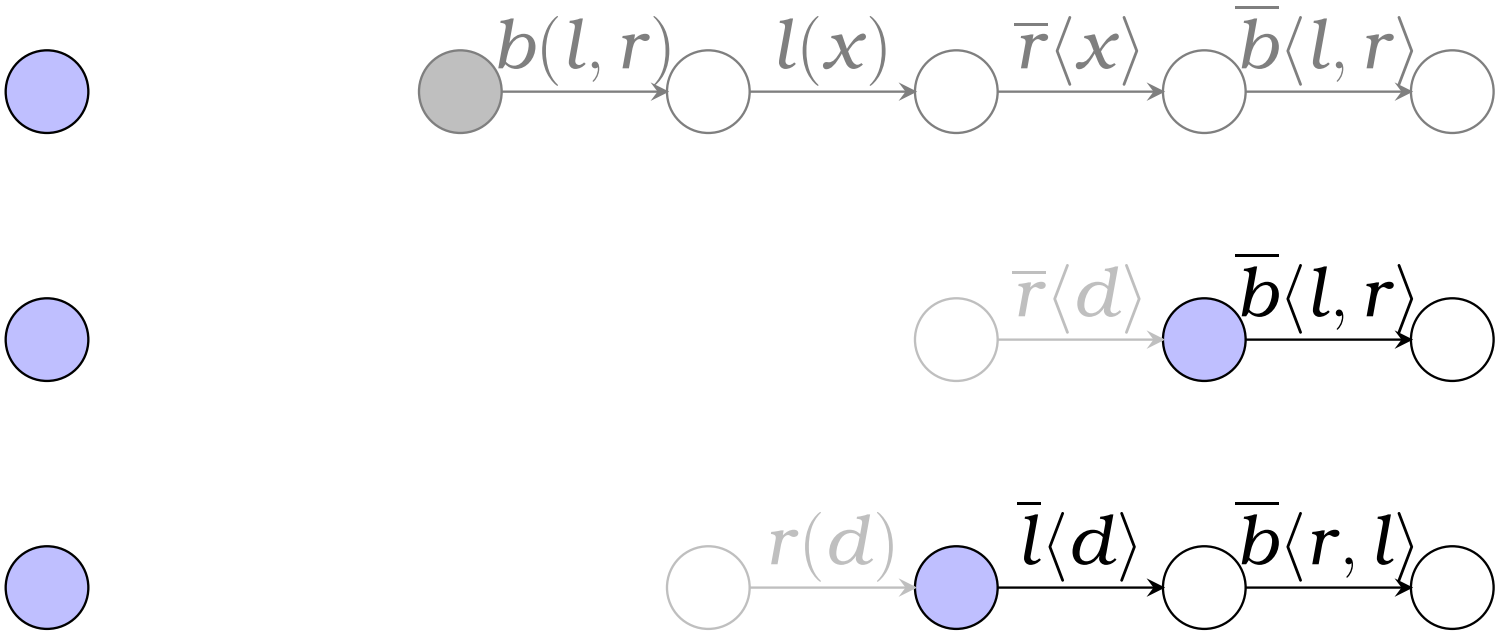
Beispiel: Ping-Pong-Puffer - Prozeduraufrufe

$$\overline{b}\langle l, r\rangle \mid \overline{b}\langle r, l\rangle \mid \overline{l}\langle d\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$$



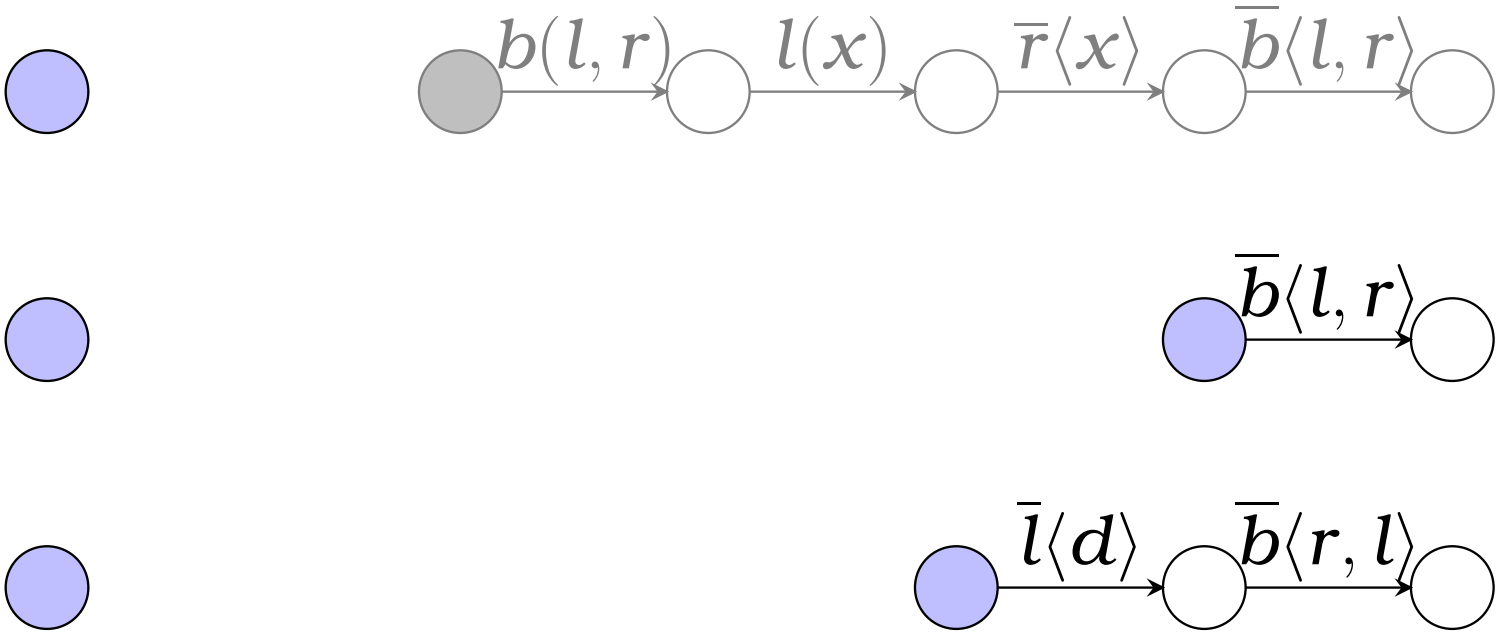
Beispiel: Ping-Pong-Puffer - Prozeduraufrufe

$$\overline{b}\langle l, r\rangle \mid \overline{b}\langle r, l\rangle \mid \overline{l}\langle d\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$$



Beispiel: Ping-Pong-Puffer - Prozeduraufrufe

$$\overline{b}\langle l, r\rangle \mid \overline{b}\langle r, l\rangle \mid \overline{l}\langle d\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$$



Warum Abstraktionen?

- Bisher: Zwei Möglichkeiten zur Variablenbindung

Warum Abstraktionen?

- Bisher: Zwei Möglichkeiten zur Variablenbindung

$$\pi ::= \tau \mid x(y) \mid \overline{x}\langle y \rangle$$
$$P ::= \sum \pi_i.P_i \mid P|P' \mid \text{new } x\ P \mid !P$$

Warum Abstraktionen?

- Bisher: Zwei Möglichkeiten zur Variablenbindung

$$\pi ::= \tau \mid x(y) \mid \overline{x}\langle y \rangle$$
$$P ::= \sum \pi_i.P_i \mid P|P' \mid \text{new } x\ P \mid !P$$

Warum Abstraktionen?

- Bisher: Zwei Möglichkeiten zur Variablenbindung

$$\pi ::= \tau \mid x(y) \mid \overline{x}\langle y \rangle$$
$$P ::= \sum \pi_i.P_i \mid P|P' \mid \text{new } x\ P \mid !P$$

- $x(y).P \xrightarrow{\overline{x}\langle a \rangle} \{a/y\}P$

Warum Abstraktionen?

- Bisher: Zwei Möglichkeiten zur Variablenbindung

$$\pi ::= \tau \mid x(y) \mid \overline{x}\langle y \rangle$$
$$P ::= \sum \pi_i.P_i \mid P|P' \mid \text{new } x\ P \mid !P$$

- $x(y).P \xrightarrow{\overline{x}\langle a \rangle} \{a/y\}P$
- $\text{new } x\ P$ bindet x in P durch einen neuen, in P nicht vorkommenden Namen

Warum Abstraktionen?

- Bisher: Zwei Möglichkeiten zur Variablenbindung

$$\pi ::= \tau \mid x(y) \mid \overline{x}\langle y \rangle$$
$$P ::= \sum \pi_i.P_i \mid P|P' \mid \text{new } x\ P \mid !P$$

- $x(y).P \xrightarrow{\overline{x}\langle a \rangle} \{a/y\}P$
- $\text{new } x\ P$ bindet x in P durch einen neuen, in P nicht vorkommenden Namen
- Idee: Variablenbindung vereinheitlichen

Was ist eine Abstraktion?

$$A ::= (x).P \mid (x).A$$

Was ist eine Abstraktion?

$$A ::= (x).P \mid (x).A$$

- $((x).A)\langle y \rangle \stackrel{def}{=} \{y/x\}A$ bezeichnet die Bindung aller freien Vorkommen von x in A durch y (Applikation)

Was ist eine Abstraktion?

$$A ::= (x).P \mid (x).A$$

- $((x).A)\langle y\rangle \stackrel{def}{=} \{y/x\}A$ bezeichnet die Bindung aller freien Vorkommen von x in A durch y (Applikation)

Abkürzende Notation:

- $\text{new } A \stackrel{def}{=} \text{new } x\ (A\langle x\rangle)$

Was ist eine Abstraktion?

$A ::= (x).P \mid (x).A$

- $((x).A)\langle y \rangle \stackrel{def}{=} \{y/x\}A$ bezeichnet die Bindung aller freien Vorkommen von x in A durch y (Applikation)

Abkürzende Notation:

- $\text{new } A \stackrel{def}{=} \text{new } x\ (A\langle x \rangle)$
- $x\ A \stackrel{def}{=} x(y).(A\langle y \rangle)$

Pi-Kalkül auf Abstraktionen

Bisher:

$$P ::= \sum \mathbf{S}_i \mid P|P' \mid \mathrm{new}\ x\ P \mid !P$$
$$\mathbf{S} ::= \tau.P \mid x(y).P \mid \overline{x}\langle y\rangle.P$$

Pi-Kalkül auf Abstraktionen

Bisher:

$$P ::= \sum S_i \mid P|P' \mid \text{new } x\ P \mid !P$$
$$S ::= \tau.P \mid x(y).P \mid \overline{x}\langle y\rangle.P$$

Mit Abstraktionen:

$$P ::= \sum S_i \mid P|P' \mid \text{new } A \mid !P$$
$$S ::= \tau.P \mid x\ A \mid \overline{x}\langle y\rangle.P$$
$$A ::= (x).P \mid (x).A$$

Pi-Kalkül auf Abstraktionen

Bisher:

$$P ::= \sum S_i \mid P|P' \mid \text{new } x\ P \mid !P$$
$$S ::= \tau.P \mid x(y).P \mid \overline{x}\langle y\rangle.P$$

Mit Abstraktionen:

$$P ::= \sum S_i \mid P|P' \mid \text{new } A \mid !P$$
$$S ::= \tau.P \mid x\ A \mid \overline{x}\langle y\rangle.P$$
$$A ::= (x).P \mid (x).A$$

- Bindung nur noch über Abstraktionen

Sequentielles Senden

Verhält sich ein Prozess P in Bezug auf das Senden auf einem Kanal x wie ein sequentieller Prozess?

Sequentielles Senden

Verhält sich ein Prozess P in Bezug auf das Senden auf einem Kanal x wie ein sequentieller Prozess?

Definition

P sendet sequentiell auf x

$\overset{def}{\Leftrightarrow} P \equiv \ldots \mid \overline{x}\langle a \rangle.Q \mid \overline{x}\langle b \rangle.R \mid \ldots$

Sequentielles Senden

Verhält sich ein Prozess P in Bezug auf das Senden auf einem Kanal x wie ein sequentieller Prozess?

Definition

P sendet sequentiell auf x

$\overset{def}{\Leftrightarrow} P \equiv \ldots \mid \overline{x}\langle a \rangle.Q \mid \overline{x}\langle b \rangle.R \mid \ldots$

- syntaktisch prüfbare Eigenschaft

Sequentielles Senden

Verhält sich ein Prozess P in Bezug auf das Senden auf einem Kanal x wie ein sequentieller Prozess?

Definition

P sendet sequentiell auf x

$\overset{def}{\Leftrightarrow} P \equiv \ldots \mid \overline{x}\langle a \rangle.Q \mid \overline{x}\langle b \rangle.R \mid \ldots$

- syntaktisch prüfbare Eigenschaft
- bleibt bei Reduktion nicht zwingend erhalten

Sequentielles Senden

Verhält sich ein Prozess P in Bezug auf das Senden auf einem Kanal x wie ein sequentieller Prozess?

Definition

P sendet sequentiell auf x

$\overset{def}{\Leftrightarrow} P \equiv \ldots \mid \overline{x}\langle a\rangle.Q \mid \overline{x}\langle b\rangle.R \mid \ldots$

- syntaktisch prüfbare Eigenschaft
- bleibt bei Reduktion nicht zwingend erhalten

$$P = \overline{x}\langle a\rangle \mid \overline{z}\langle b\rangle.\overline{x}\langle c\rangle \mid z(d)$$

Sequentielles Senden

Verhält sich ein Prozess P in Bezug auf das Senden auf einem Kanal x wie ein sequentieller Prozess?

Definition

P sendet sequentiell auf x

$\overset{def}{\Leftrightarrow} P \equiv \ldots \mid \overline{x}\langle a\rangle.Q \mid \overline{x}\langle b\rangle.R \mid \ldots$

- syntaktisch prüfbare Eigenschaft
- bleibt bei Reduktion nicht zwingend erhalten

$$P = \overline{x}\langle a\rangle \mid \overline{z}\langle b\rangle.\overline{x}\langle c\rangle \mid z(d)$$

Sequentielles Senden

Verhält sich ein Prozess P in Bezug auf das Senden auf einem Kanal x wie ein sequentieller Prozess?

Definition

P sendet sequentiell auf x

$\overset{def}{\Leftrightarrow} P \equiv \ldots \mid \overline{x}\langle a\rangle.Q \mid \overline{x}\langle b\rangle.R \mid \ldots$

- syntaktisch prüfbare Eigenschaft
- bleibt bei Reduktion nicht zwingend erhalten

$$P = \overline{x}\langle a\rangle \mid \overline{z}\langle b\rangle.\overline{x}\langle c\rangle \mid z(d)$$
$$\longrightarrow \overline{x}\langle a\rangle \mid \overline{x}\langle c\rangle \mid 0$$

Sequentielles Senden

Verhält sich ein Prozess P in Bezug auf das Senden auf einem Kanal x wie ein sequentieller Prozess?

Definition

P sendet sequentiell auf x

$\overset{def}{\Leftrightarrow} P \equiv \ldots \mid \overline{x}\langle a\rangle.Q \mid \overline{x}\langle b\rangle.R \mid \ldots$

- syntaktisch prüfbare Eigenschaft
- bleibt bei Reduktion nicht zwingend erhalten

$$P = \overline{x}\langle a\rangle \mid \overline{z}\langle b\rangle.\overline{x}\langle c\rangle \mid z(d)$$
$$\longrightarrow \overline{x}\langle a\rangle \mid \overline{x}\langle c\rangle \mid 0$$

Sequentielles Senden

Verhält sich ein Prozess P in Bezug auf das Senden auf einem Kanal x wie ein sequentieller Prozess?

Definition

P sendet sequentiell auf x

$\overset{def}{\Leftrightarrow} P \equiv \ldots \mid \overline{x}\langle a\rangle.Q \mid \overline{x}\langle b\rangle.R \mid \ldots$

- syntaktisch prüfbare Eigenschaft
- bleibt bei Reduktion nicht zwingend erhalten

$$P = \overline{x}\langle a\rangle \mid \overline{z}\langle b\rangle.\overline{x}\langle c\rangle \mid z(d)$$
$$\longrightarrow \overline{x}\langle a\rangle \mid \overline{x}\langle c\rangle \mid 0$$

$\Rightarrow$ Zusatzbedingungen finden

Einfache Systeme

- schwierig, geeignete Zusatzbedingungen für allgemeine Prozesse zu finden

Einfache Systeme

- schwierig, geeignete Zusatzbedingungen für allgemeine Prozesse zu finden

Kompromiss: Betrachte Einfache Systeme.

Einfache Systeme

- schwierig, geeignete Zusatzbedingungen für allgemeine Prozesse zu finden

Kompromiss: Betrachte Einfache Systeme. Warum? Invarianz unter Reduktion.

Einfache Systeme

- schwierig, geeignete Zusatzbedingungen für allgemeine Prozesse zu finden

Kompromiss: Betrachte Einfache Systeme. Warum? Invarianz unter Reduktion.

Definition

Ein Prozess P heisst *Einfaches System*, wenn er zu einer Normalform

$(*)\ P \equiv \ \mathbf{new}\ \vec{z}\ (M_1 | \ldots | M_m | !N_1 | \ldots | !N_n)$

strukturell kongruent ist,

Einfache Systeme

- schwierig, geeignete Zusatzbedingungen für allgemeine Prozesse zu finden

Kompromiss: Betrachte Einfache Systeme. Warum? Invarianz unter Reduktion.

Definition

Ein Prozess P heisst *Einfaches System*, wenn er zu einer Normalform

$(*)\ P \equiv \ \mathbf{new}\ \vec{z}\ (M_1 | \ldots | M_m | !N_1 | \ldots | !N_n)$

strukturell kongruent ist, und die folgende Bedingung erfüllt:

- Die M_i und N_j sind Summen die weder eine Replikation noch eine parallele Komposition enthalten

Beispiel: Einfache Systeme

Einfache Systeme sind eine parallele Komposition aus

- sequentiellen Prozessen und

Beispiel: Einfache Systeme

Einfache Systeme sind eine parallele Komposition aus

- sequentiellen Prozessen und
- replizierten sequentiellen Prozessen

Beispiel: Einfache Systeme

Einfache Systeme sind eine parallele Komposition aus

- sequentiellen Prozessen und
- replizierten sequentiellen Prozessen

Beispiel

$\overline{b}\langle l, r\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$

Puffer sind einfach.

Beispiel: Einfache Systeme

Einfache Systeme sind eine parallele Komposition aus

- sequentiellen Prozessen und
- replizierten sequentiellen Prozessen

Beispiel

$\overline{b}\langle l, r\rangle \mid !b(l, r).l(x).\overline{r}\langle x\rangle.\overline{b}\langle l, r\rangle$
Puffer sind einfach.

Das Mobile Phone Beispiel aus Kapitel 8 ist ein Einfaches System.

CCS-Prozesse als Einfache Systeme

- jeder CCS-Prozess P ist strukturell kongruent zu einer Normalform $\mathbf{new}\ \vec{z}\ (M_1 | \ldots | M_m)$

CCS-Prozesse als Einfache Systeme

- jeder CCS-Prozess P ist strukturell kongruent zu einer Normalform $\mathbf{new}\ \vec{z}\ (M_1 | \ldots | M_m)$
- CCS kennt keine Replikation, als einzige Bedingung bleibt: M_i enthält keine parallele Komposition

CCS-Prozesse als Einfache Systeme

- jeder CCS-Prozess P ist strukturell kongruent zu einer Normalform $\mathbf{new}\ \vec{z}\ (M_1 | \ldots | M_m)$
- CCS kennt keine Replikation, als einzige Bedingung bleibt: M_i enthält keine parallele Komposition

Beispiel

Alle Beispiele aus Kapitel 7 (Lottery, Job Shop, Scheduler, Buffer, Counter) sind Einfache Systeme.

CCS-Prozesse als Einfache Systeme

- jeder CCS-Prozess P ist strukturell kongruent zu einer Normalform $\mathbf{new}\ \vec{z}\ (M_1 | \ldots | M_m)$
- CCS kennt keine Replikation, als einzige Bedingung bleibt: M_i enthält keine parallele Komposition

Beispiel

Alle Beispiele aus Kapitel 7 ($\mathbf{Lottery}$, $\mathbf{Job\ Shop}$, $\mathbf{Scheduler}$, $\mathbf{Buffer}$, $\mathbf{Counter}$) sind Einfache Systeme.

$\mathit{Zelle} = \mathit{teile}.(\mathit{Zelle} \mid \mathit{Zelle}) + \mathit{stirb}.0$ ist kein Einfaches System.

Sequentielles Senden - Eindeutige Sender

Es darf höchstens eine der sequentiellen Komponenten M_i auf x senden können (d.h. einen Aktionspräfix der Form $\overline{x}\langle y \rangle.Q$ enthalten).

Sequentielles Senden - Eindeutige Sender

Es darf höchstens eine der sequentiellen Komponenten M_i auf x senden können (d.h. einen Aktionspräfix der Form $\overline{x}\langle y \rangle.Q$ enthalten).

- noch nicht Invariant

Sequentielles Senden - Eindeutige Sender

Es darf höchstens eine der sequentiellen Komponenten M_i auf x senden können (d.h. einen Aktionspräfix der Form $\overline{x}\langle y\rangle.Q$ enthalten).

- noch nicht Invariant

Beispiel

$P = \overline{x}\langle b\rangle \mid \overline{y}\langle x\rangle \mid y(b).\overline{b}\langle c\rangle$

Sequentielles Senden - Eindeutige Sender

Es darf höchstens eine der sequentiellen Komponenten M_i auf x senden können (d.h. einen Aktionspräfix der Form $\overline{x}\langle y\rangle.Q$ enthalten).

- noch nicht Invariant

Beispiel

$P = \overline{x}\langle b\rangle \mid \overline{y}\langle x\rangle \mid y(b).\overline{b}\langle c\rangle$

Sequentielles Senden - Eindeutige Sender

Es darf höchstens eine der sequentiellen Komponenten M_i auf x senden können (d.h. einen Aktionspräfix der Form $\overline{x}\langle y\rangle.Q$ enthalten).

- noch nicht Invariant

Beispiel

$P = \overline{x}\langle b\rangle \mid \overline{y}\langle x\rangle \mid y(b).\overline{b}\langle c\rangle \ \rightarrow \overline{x}\langle b\rangle \mid \overline{x}\langle c\rangle$

Sequentielles Senden - Eindeutige Sender

Es darf höchstens eine der sequentiellen Komponenten M_i auf x senden können (d.h. einen Aktionspräfix der Form $\overline{x}\langle y\rangle.Q$ enthalten).

- noch nicht Invariant

Beispiel

$P = \overline{x}\langle b\rangle \mid \overline{y}\langle x\rangle \mid y(b).\overline{b}\langle c\rangle \;\rightarrow \overline{x}\langle b\rangle \mid \overline{x}\langle c\rangle$

Sequentielles Senden - Eindeutige Sender

Es darf höchstens eine der sequentiellen Komponenten M_i auf x senden können (d.h. einen Aktionspräfix der Form $\overline{x}\langle y\rangle.Q$ enthalten).

- noch nicht Invariant

Beispiel

$P = \overline{x}\langle b\rangle \mid \overline{y}\langle x\rangle \mid y(b).\overline{b}\langle c\rangle \ \rightarrow \overline{x}\langle b\rangle \mid \overline{x}\langle c\rangle$

Definition

Ein Prozess P *verwaltet* einen Kanal x, wenn er einen Aktionspräfix der Form $\overline{x}\langle a\rangle$ oder $\overline{y}\langle x\rangle$ enthält.

Eindeutige Verwaltung eines Kanals x

Definition

Ein Einfaches System $\mathbf{new}\ \vec{z}\ (M_1|\ldots|M_m|N_1|\ldots|N_n)$ *verwaltet* einen Kanal x *eindeutig*, wenn höchstens eines der M_i und keines der N_j den Kanal x verwaltet.

Eindeutige Verwaltung eines Kanals x

Definition

Ein Einfaches System $\mathbf{new}\ \vec{z}\ (M_1|\ldots|M_m|N_1|\ldots|N_n)$ *verwaltet* einen Kanal x *eindeutig*, wenn höchstens eines der M_i und keines der N_j den Kanal x verwaltet.

- eindeutige Verwaltung $\Rightarrow$ sequentielles Senden

Eindeutige Verwaltung eines Kanals x

Definition

Ein Einfaches System $\mathbf{new}\ \vec{z}\ (M_1|\ldots|M_m|N_1|\ldots|N_n)$ *verwaltet* einen Kanal x *eindeutig*, wenn höchstens eines der M_i und keines der N_j den Kanal x verwaltet.

- eindeutige Verwaltung $\Rightarrow$ sequentielles Senden
- leider immer noch nicht invariant unter Reduktion

Eindeutige Verwaltung eines Kanals x

Definition

Ein Einfaches System $\mathbf{new}\ \vec{z}\ (M_1|\ldots|M_m|N_1|\ldots|N_n)$ *verwaltet* einen Kanal x *eindeutig*, wenn höchstens eines der M_i und keines der N_j den Kanal x verwaltet.

- eindeutige Verwaltung $\Rightarrow$ sequentielles Senden
- leider immer noch nicht invariant unter Reduktion

Beispiel

$P_1 = \overline{y}\langle x\rangle.\overline{x}\langle b\rangle \mid y(a).\overline{a}\langle c\rangle$

Eindeutige Verwaltung eines Kanals x

Definition

Ein Einfaches System $\mathbf{new}\ \vec{z}\ (M_1|\ldots|M_m|N_1|\ldots|N_n)$ *verwaltet* einen Kanal x *eindeutig*, wenn höchstens eines der M_i und keines der N_j den Kanal x verwaltet.

- eindeutige Verwaltung $\Rightarrow$ sequentielles Senden
- leider immer noch nicht invariant unter Reduktion

Beispiel

$P_1 = \overline{y}\langle x\rangle.\overline{x}\langle b\rangle \mid y(a).\overline{a}\langle c\rangle$

Eindeutige Verwaltung eines Kanals x

Definition

Ein Einfaches System $\mathbf{new}\ \vec{z}\ (M_1|\ldots|M_m|N_1|\ldots|N_n)$ *verwaltet* einen Kanal x *eindeutig*, wenn höchstens eines der M_i und keines der N_j den Kanal x verwaltet.

- eindeutige Verwaltung $\Rightarrow$ sequentielles Senden
- leider immer noch nicht invariant unter Reduktion

Beispiel

$P_1 = \overline{y}\langle x\rangle.\overline{x}\langle b\rangle \mid y(a).\overline{a}\langle c\rangle \ \rightarrow \overline{x}\langle b\rangle \mid \overline{x}\langle c\rangle$

Eindeutige Verwaltung eines Kanals x

Definition

Ein Einfaches System $\mathbf{new}\ \vec{z}\ (M_1|\ldots|M_m|N_1|\ldots|N_n)$ *verwaltet* einen Kanal x *eindeutig*, wenn höchstens eines der M_i und keines der N_j den Kanal x verwaltet.

- eindeutige Verwaltung $\Rightarrow$ sequentielles Senden
- leider immer noch nicht invariant unter Reduktion

Beispiel

$P_1 = \overline{y}\langle x\rangle.\overline{x}\langle b\rangle \mid y(a).\overline{a}\langle c\rangle \ \rightarrow \overline{x}\langle b\rangle \mid \overline{x}\langle c\rangle$

Problem: Sequentieller Prozess M_i versendet x, verwaltet x aber weiterhin

x-Vergesslichkeit

Definition

Ein Prozess P heisst *x-vergesslich*, wenn in jedem Subterm der Form $\overline{a}\langle x \rangle.Q$ der Prozess Q den Kanal x nicht verwaltet.

x-Vergesslichkeit

Definition

Ein Prozess P heisst *x-vergesslich*, wenn in jedem Subterm der Form $\overline{a}\langle x\rangle.Q$ der Prozess Q den Kanal x nicht verwaltet.

Beispiel

$P_1 = \overline{y}\langle x\rangle.\overline{x}\langle b\rangle \mid y(a).\overline{a}\langle c\rangle$ ist nicht x-vergesslich.

x-Vergesslichkeit

Definition

Ein Prozess P heisst *x-vergesslich*, wenn in jedem Subterm der Form $\overline{a}\langle x\rangle.Q$ der Prozess Q den Kanal x nicht verwaltet.

Beispiel

$P_1 = \overline{y}\langle x\rangle.\overline{x}\langle b\rangle \mid y(a).\overline{a}\langle c\rangle$ ist nicht x-vergesslich.

x-Vergesslichkeit

Definition

Ein Prozess P heisst *x-vergesslich*, wenn in jedem Subterm der Form $\overline{a}\langle x\rangle.Q$ der Prozess Q den Kanal x nicht verwaltet.

Beispiel

$P_1 = \overline{y}\langle x\rangle.\overline{x}\langle b\rangle \mid y(a).\overline{a}\langle c\rangle$ ist nicht x-vergesslich.
$P_2 = \overline{u}\langle x\rangle \mid u(a).\overline{y}\langle a\rangle.\overline{a}\langle b\rangle \mid y(a).\overline{a}\langle c\rangle$ ist x-vergesslich, aber...

x-Vergesslichkeit

Definition

Ein Prozess P heisst *x-vergesslich*, wenn in jedem Subterm der Form $\overline{a}\langle x\rangle.Q$ der Prozess Q den Kanal x nicht verwaltet.

Beispiel

$P_1 = \overline{y}\langle x\rangle.\overline{x}\langle b\rangle \mid y(a).\overline{a}\langle c\rangle$ ist nicht x-vergesslich.

$P_2 = \overline{u}\langle x\rangle \mid u(a).\overline{y}\langle a\rangle.\overline{a}\langle b\rangle \mid y(a).\overline{a}\langle c\rangle$ ist x-vergesslich, aber...

$P_2 \rightarrow P_1$

x-Vergesslichkeit

Definition

Ein Prozess P heisst *x-vergesslich*, wenn in jedem Subterm der Form $\overline{a}\langle x\rangle.Q$ der Prozess Q den Kanal x nicht verwaltet.

Beispiel

$P_1 = \overline{y}\langle x\rangle.\overline{x}\langle b\rangle \mid y(a).\overline{a}\langle c\rangle$ ist nicht x-vergesslich.
$P_2 = \overline{u}\langle x\rangle \mid u(a).\overline{y}\langle a\rangle.\overline{a}\langle b\rangle \mid y(a).\overline{a}\langle c\rangle$ ist x-vergesslich, aber...
$P_2 \rightarrow P_1$

- Problem: x wird von einem Prozess $u(a).Q$ empfangen, der nicht a-vergesslich ist

Sequentielles Senden - die Invariante

Satz

Sei P ein Einfaches System das die folgenden Bedingungen erfüllt:

Sequentielles Senden - die Invariante

Satz

Sei P ein Einfaches System das die folgenden Bedingungen erfüllt:

- P verwaltet x eindeutig

Sequentielles Senden - die Invariante

Satz

Sei P ein Einfaches System das die folgenden Bedingungen erfüllt:

- P verwaltet x eindeutig
- P ist x-vergesslich

Sequentielles Senden - die Invariante

Satz

Sei P ein Einfaches System das die folgenden Bedingungen erfüllt:

- P verwaltet x eindeutig
- P ist x-vergesslich
- für jeden Subterm der Form $z(y).Q$ von P ist Q y-vergesslich

Sequentielles Senden - die Invariante

Satz

Sei P ein Einfaches System das die folgenden Bedingungen erfüllt:

- P verwaltet x eindeutig
- P ist x-vergesslich
- für jeden Subterm der Form $z(y).Q$ von P ist Q y-vergesslich

Es gelte $P \rightarrow P'$.
Dann gilt: P' ist einfach und erfüllt ebenfalls die drei Bedingungen.

Übersicht

Syntaktischer Zucker

- Polyadischer Pi-Kalkül
- Rekursive Definitionen

Abstraktionen

- Bindung von Namen im Pi-Kalkül

Sequentielles Senden

- Einfache Systeme
- Eindeutige Sender
- x-Vergesslichkeit
- Die Invariante