LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

Correctly Implementing Synchronous Message Passing in the Pi-Calculus by Concurrent Haskell's MVars

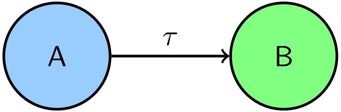
Manfred Schmidt-Schauß
Goethe-University Frankfurt

David Sabel
LMU Munich

EXPRESS/SOS 2020
August 31, 2020

General Motivation

- We are interested in the **correctness of translations** between programming languages



- Questions:
  - can language B **express** language A?
  - does τ **correctly implement** the primitives of A using the primitives of B?
- We focus correctness w.r.t. **contextual equivalence**.
  - equates programs if they **behave the same** (w.r.t. termination) in all contexts
  - it is a generic notion, **applicable for many** programming languages

Goals of the Current Work

π-calculus [Milner, Parrow, & Walker,1992]

- a standard model for (mobile) processes with message passing
- we use the synchronous π-calculus with replication and a constant `stop` (called $\Pi_{\texttt{Stop}}$)

Concurrent Haskell [Peyton-Jones, Gordon, & Finne, 1996]

- extends Haskell by concurrent threads and shared-memory (so-called MVars)
- we use the calculus CH (a variant of CHF, [S. & Schmidt-Schauß, 2011])

Questions:

- Can we encode/translate $\Pi_{\texttt{Stop}}$ into CH?
- Which (correctness) properties hold for the translation?

$$\Pi_{\texttt{Stop}} \xrightarrow{\tau} \mathrm{CH}$$

The Source Language $\Pi_{\texttt{Stop}}$: The π-calculus with Stop

- we consider the synchronous π-calculus, with replication, without sums
- extended with a constant $\texttt{Stop}$ to signal success [S. & Schmidt-Schauß 2015]

Syntax of Processes

$$P, Q \in \mathit{Proc}_\pi ::= P \mid Q \mid \underbrace{x(y).P}_{\text{input}} \mid \underbrace{\overline{x}y.P}_{\text{output}} \mid \nu x.P \mid !P \mid \mathbf{0} \mid \underbrace{\texttt{Stop}}_{\text{success}}$$

The Source Language $\Pi_{\texttt{Stop}}$: The π-calculus with Stop

- we consider the synchronous π-calculus, with replication, without sums
- extended with a constant $\texttt{Stop}$ to signal success [S. & Schmidt-Schauß 2015]

Syntax of Processes

$$P, Q \in \mathit{Proc}_\pi ::= P \mid Q \mid \underbrace{x(y).P}_{\text{input}} \mid \underbrace{\overline{x}y.P}_{\text{output}} \mid \nu x.P \mid !P \mid \mathbf{0} \mid \underbrace{\texttt{Stop}}_{\text{success}}$$

Reduction contexts: $\mathbb{D} \in \mathit{PCtxt}_\pi ::= [\cdot] \mid \mathbb{D} \mid P \mid P \mid \mathbb{D} \mid \nu x.\mathbb{D}$

Reduction rule for interaction

$$x(y).P \mid \overline{x}z.Q \xrightarrow{ia} P[z/y] \mid Q$$

Standard Reduction $\xrightarrow{sr}$:

$$P \xrightarrow{sr} Q \text{ if } P \equiv \mathbb{D}[P'], P' \xrightarrow{ia} Q', \mathbb{D}[Q'] \equiv Q$$

Process P is **successful** if $P \equiv \mathbb{D}[\texttt{Stop}]$

The Target Language CH: Functional Language + Threads & MVars

Syntax of Processes:

$$P_i \in \mathit{Proc}_{CH} ::= P_1 \mid P_2 \mid \nu x.P \mid \underbrace{\Leftarrow e}_{\text{thread}} \mid x = e \mid \underbrace{x\,\mathbf{m}\,e \mid x\,\mathbf{m}\,-}_{\text{MVars}}$$

Main-thread: a unique distinguished thread $\xLeftarrow{\text{main}} e$

Syntax of Expressions:

$$e_i ::= \overbrace{x \mid \lambda x.e \mid (e_1\ e_2) \mid c\,\overrightarrow{e_i} \mid \texttt{case}\ e\ \texttt{of}\ alts \mid \texttt{seq}\ e_1\ e_2 \mid \texttt{letrec}\ x_1{=}e_1,\ldots,x_n{=}e_n\ \texttt{in}\ e}^{\text{extended lambda-calculus}}$$

$$\mid \underbrace{\texttt{return}\ e \mid e_1 \gg\!= e_2 \mid \texttt{forkIO}\ e}_{\text{IO-monad \& concurrency}} \mid \underbrace{\texttt{newMVar}\ e \mid \texttt{takeMVar}\ e \mid \texttt{putMVar}\ e_1\ e_2}_{\text{monadic MVar-operations}}$$

CH: Operational Semantics (Excerpt)

Monadic Computations

$$
\begin{array}{llcl}
\text{(lunit)} & \Leftarrow \mathbb{M}[\mathtt{return}\ e_1 \gg= \ e_2] & \to & \Leftarrow \mathbb{M}[e_2\ e_1] \\
\text{(fork)} & \Leftarrow \mathbb{M}[\mathtt{forkIO}\ e] & \to & \Leftarrow \mathbb{M}[\mathtt{return}\ ()] \mid \ \Leftarrow e \\
\text{(tmvar)} & \Leftarrow \mathbb{M}[\mathtt{takeMVar}\ x] \mid x\,\mathbf{m}\,e & \to & \Leftarrow \mathbb{M}[\mathtt{return}\ e] \mid x\,\mathbf{m}\,- \\
\text{(pmvar)} & \Leftarrow \mathbb{M}[\mathtt{putMVar}\ x\ e] \mid x\,\mathbf{m}\,- & \to & \Leftarrow \mathbb{M}[\mathtt{return}\ ()] \mid x\,\mathbf{m}\,e
\end{array}
$$

...

Functional Evaluation

$$
\text{(beta)} \quad \Leftarrow \mathbb{M}[\mathbb{F}[((\lambda x.e_1)\ e_2)]] \to \ \Leftarrow \mathbb{M}[\mathbb{F}[e_1[e_2/x]]]
$$

...

Standard Reduction $\xrightarrow{sr}$:

$P \xrightarrow{sr} Q$ if $P \equiv \mathbb{D}[P'], P' \to Q', \mathbb{D}[Q'] \equiv Q$

Process P is **successful** if

$$
P \equiv \nu x_1 \ldots x_n.(\stackrel{\text{main}}{\Longleftarrow} \mathtt{return}\ e \mid P')
$$

Contexts:

$$
\begin{array}{lcl}
\mathbb{M} & ::= & [\cdot] \mid \mathbb{M} \gg= e \\
\mathbb{E} & ::= & [\cdot] \mid (\mathbb{E}\ e) \mid (\mathtt{case}\ \mathbb{E}\ \mathtt{of}\ alts) \mid (\mathtt{seq}\ \mathbb{E}\ e) \\
\mathbb{F} & ::= & \mathbb{E} \mid (\mathtt{takeMVar}\ \mathbb{E}) \mid (\mathtt{putMVar}\ \mathbb{E}\ e) \\
\mathbb{D} & ::= & [\cdot] \mid \mathbb{D} \mid P \mid P \mid \mathbb{D} \mid \nu x.\mathbb{D}
\end{array}
$$

Semantics of Processes in Source and Target Language

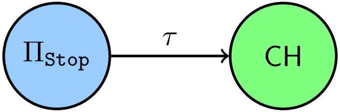
Observations:

- P **may-converges** $(P{\downarrow})$ iff $P \xrightarrow{sr,*} P'$ and P' is successful.
- P **should-converges** $(P{\Downarrow})$ iff $\forall P' : P \xrightarrow{sr,*} P' \implies P'{\downarrow}$

Contextual equivalence $\sim_c$

$$P_1 \sim_c P_2 \text{ iff } \forall C : C[P_1]{\downarrow} \iff C[P_2]{\downarrow} \text{ and } C[P_1]{\Downarrow} \iff C[P_2]{\Downarrow}$$

Task: Find a Translation ...



- that is correct w.r.t. $\sim_c$
- we present the main ideas of the translation step by step:
  - translation of the Stop-constant
  - translation of 0, parallel composition, replication
  - translation of channels (and interaction): with different variations

Translation

Translation of `Stop`:

$$\tau_0(P) = \stackrel{\textsf{main}}{\Longleftarrow} \textbf{do}\ \{stop \leftarrow \texttt{newMVar}\ ();\ \texttt{forkIO}\ \tau(P);\ \texttt{putMVar}\ stop\ ()\}$$

$$= C^{\tau}_{out}[\tau(P)]$$

$$\tau(\texttt{Stop}) = \texttt{takeMVar}\ stop$$

Translation

Translation of `Stop`:

$$\tau_0(P) = \overset{\text{main}}{\Longleftarrow} \textbf{do}\ \{stop \leftarrow \texttt{newMVar}\ ();\ \texttt{forkIO}\ \tau(P);\ \texttt{putMVar}\ stop\ ()\}$$

$$= C^{\tau}_{out}[\tau(P)]$$

$$\tau(\texttt{Stop}) = \texttt{takeMVar}\ stop$$

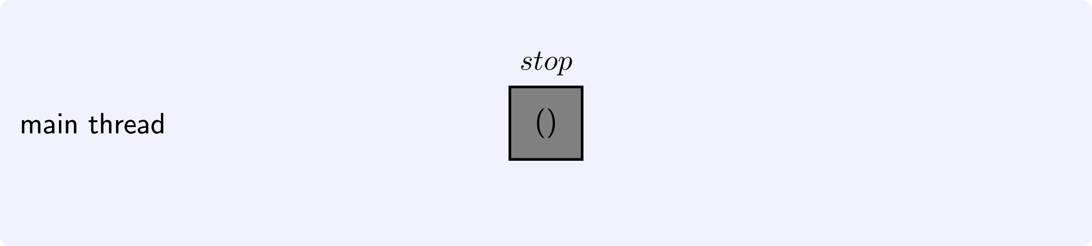
Translation

Translation of `Stop`:

$$\tau_0(P) = \overset{\text{main}}{\Longleftarrow} \textbf{do} \; \{ stop \leftarrow \texttt{newMVar} \; (); \; \texttt{forkIO} \; \tau(P); \; \texttt{putMVar} \; stop \; () \}$$

$$= C^{\tau}_{out}[\tau(P)]$$

$$\tau(\texttt{Stop}) = \texttt{takeMVar} \; stop$$



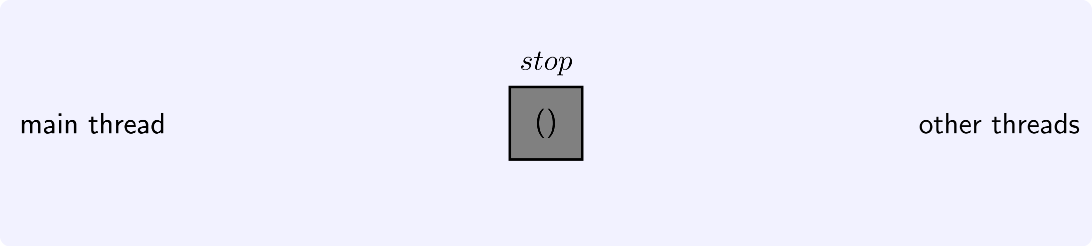
Translation

Translation of `Stop`:

$$\tau_0(P) = \overset{\text{main}}{\Longleftarrow} \textbf{do}\ \{ stop \leftarrow \texttt{newMVar}\ ();\ \texttt{forkIO}\ \tau(P);\ \texttt{putMVar}\ stop\ () \}$$

$$\tau(\texttt{Stop}) = \texttt{takeMVar}\ stop$$

$$= C^{\tau}_{out}[\tau(P)]$$



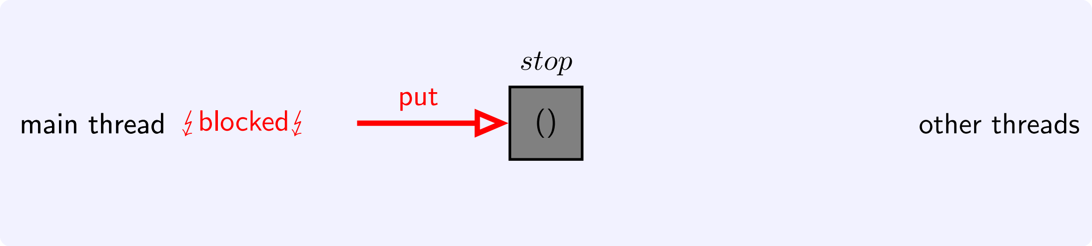
Translation

Translation of `Stop`:

$$\tau_0(P) = \overset{\text{main}}{\Longleftarrow} \textbf{do}\ \{stop \leftarrow \texttt{newMVar}\ ();\quad \texttt{forkIO}\ \tau(P);\quad \color{red}{\texttt{putMVar}\ stop\ ()}\}$$

$$= C^{\tau}_{out}[\tau(P)]$$

$$\tau(\texttt{Stop}) = \texttt{takeMVar}\ stop$$

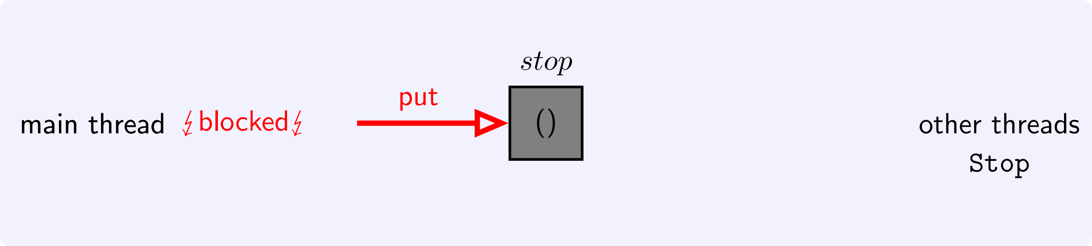


Translation of `Stop`:

$$\tau_0(P) = \xleftarrow{\text{main}} \mathbf{do}\ \{stop \leftarrow \texttt{newMVar}\ ();\ \texttt{forkIO}\ \tau(P);\ \texttt{putMVar}\ stop\ ()\}$$

$$\tau(\texttt{Stop}) = \texttt{takeMVar}\ stop$$

$$= C^{\tau}_{out}[\tau(P)]$$



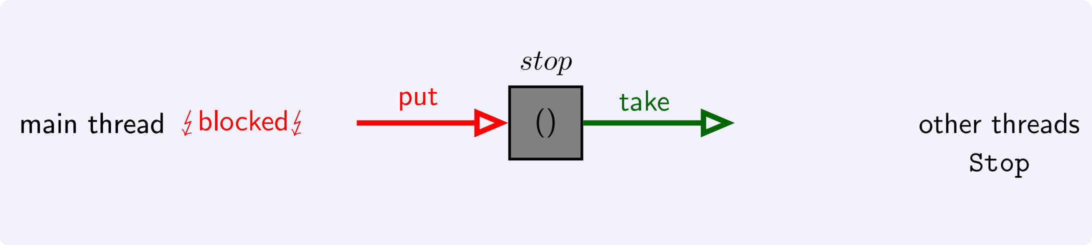
Translation

Translation of `Stop`:

$$\tau_0(P) = \overset{\text{main}}{\Longleftarrow} \textbf{do}\ \{ stop \leftarrow \texttt{newMVar}\ ();\ \texttt{forkIO}\ \tau(P);\ \texttt{putMVar}\ stop\ ()\}$$

$$= C^{\tau}_{out}[\tau(P)]$$

$$\tau(\texttt{Stop}) = \texttt{takeMVar}\ stop$$



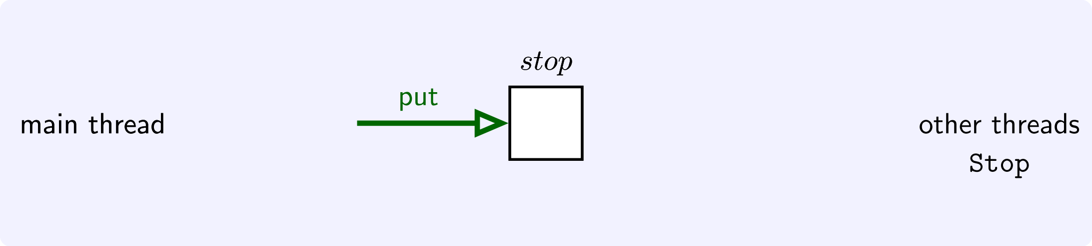
Translation

Translation of `Stop`:

$$\tau_0(P) = \overset{\text{main}}{\Longleftarrow} \textbf{do}\ \{stop \leftarrow \texttt{newMVar}\ ();\ \texttt{forkIO}\ \tau(P);\ \texttt{putMVar}\ stop\ ()\}$$

$$= C^{\tau}_{out}[\tau(P)]$$

$$\tau(\texttt{Stop}) = \texttt{takeMVar}\ stop$$



Translation

Translation of `Stop`**:**

$$\tau_0(P) = \overset{\text{main}}{\Longleftarrow} \textbf{do}\ \{ stop \leftarrow \texttt{newMVar}\ ();$$
$$\texttt{forkIO}\ \tau(P);$$
$$\texttt{putMVar}\ stop\ ()\}$$

$$= C^{\tau}_{out}[\tau(P)]$$

$$\tau(\texttt{Stop}) = \texttt{takeMVar}\ stop$$

main thread **successful**

$stop$

$\boxed{()}$

other threads

`Stop`

Translation

Translation of `Stop`:

$$\tau_0(P) = \stackrel{\text{main}}{\Longleftarrow} \mathbf{do}\ \{stop \leftarrow \texttt{newMVar}\ ();\ \texttt{forkIO}\ \tau(P);\ \texttt{putMVar}\ stop\ ()\}$$

$$= C^{\tau}_{out}[\tau(P)]$$

$$\tau(\texttt{Stop}) = \texttt{takeMVar}\ stop$$

Translation of 0, Parallel Composition, and Replication:

$$\tau(0) = \texttt{return}\ ()$$

$$\tau(P \mid Q) = \mathbf{do}\ \{\texttt{forkIO}\ \tau(Q); \tau(P)\}$$

$$\tau(!P) = \texttt{letrec}\ f = \mathbf{do}\ \{\texttt{forkIO}\ \tau(P); f\}\ \texttt{in}\ f$$

Translation of Channels and Message Passing

Two approaches to encode synchronous communication by several accesses to MVars

- Using **a private MVar** per communication

  (similar to [Boudol 1992, Honda & Tokora, 1991] where private names guarantee correct communication while encoding the synchronous in the asynchronous π-calculus)

- Using a fixed number of **global MVars** per channel

  avoids to dynamically generate "garbage"

Translation with Private MVar

π-calculus-channels are translated into

$$\texttt{data Channel} = \texttt{Chan}\ (\texttt{MVar}\ (\texttt{Channel}, \texttt{MVar}\ ()))$$

$$\tau(\nu x.P) = \textbf{do}\,\{chanx \leftarrow newEmptyMVar; \texttt{letrec}\ x = \texttt{Chan}\,chanx\ \texttt{in}\ \tau(P)\}$$

$$\begin{aligned}\tau(\overline{x}z.Q) = \textbf{do}\,\{&checkx \leftarrow \texttt{newMVar}\ ();\\ &\texttt{putMVar}\ (unchan\,x)\ (z, checkx);\\ &\texttt{putMVar}\ checkx\ (); \tau(Q)\}\end{aligned}$$

$$\begin{aligned}\tau(x(y).P) = \textbf{do}\,\{&(y, checkx) \leftarrow \texttt{takeMVar}\ (unchan\,x);\\ &\texttt{takeMVar}\ checkx; \tau(P)\}\end{aligned}$$

Translation with Private MVar

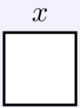
π-calculus-channels are translated into

$$\texttt{data Channel} = \texttt{Chan}\ (\texttt{MVar}\ (\texttt{Channel}, \texttt{MVar}\ ()))$$

$\tau(\nu x.P) = \textbf{do}\,\{\mathit{chanx} \leftarrow \mathit{newEmptyMVar}; \texttt{letrec}\ x = \texttt{Chan}\ \mathit{chanx}\ \texttt{in}\ \tau(P)\}$

$$\begin{aligned}\tau(\overline{x}z.Q) = \textbf{do}\,\{&\mathit{checkx} \leftarrow \texttt{newMVar}\ ();\\ &\texttt{putMVar}\ (\mathit{unchan}\, x)\ (z, \mathit{checkx});\\ &\texttt{putMVar}\ \mathit{checkx}\ (); \tau(Q)\}\end{aligned}$$

$$\begin{aligned}\tau(x(y).P) = \textbf{do}\,\{&(y, \mathit{checkx}) \leftarrow \texttt{takeMVar}\ (\mathit{unchan}\, x);\\ &\texttt{takeMVar}\ \mathit{checkx}; \tau(P)\}\end{aligned}$$



channel creation

νx

Translation with Private MVar

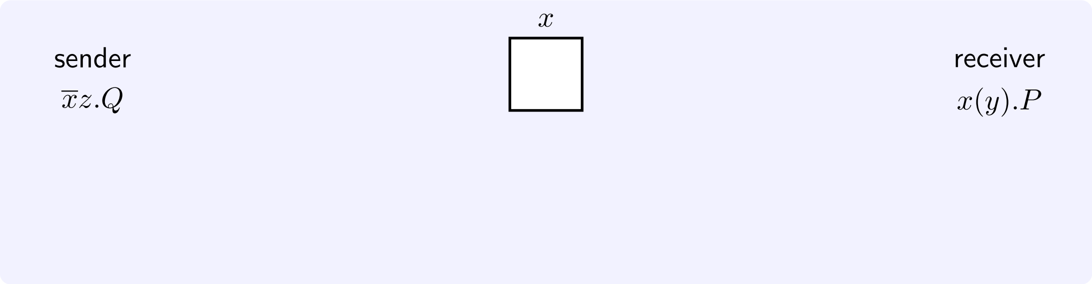
π-calculus-channels are translated into

$$\texttt{data Channel} = \texttt{Chan}\ (\texttt{MVar}\ (\texttt{Channel}, \texttt{MVar}\ ()))$$

$$\tau(\nu x.P) = \textbf{do}\,\{chanx \leftarrow newEmptyMVar; \texttt{letrec}\ x = \texttt{Chan}\, chanx\ \texttt{in}\ \tau(P)\}$$

$$\begin{aligned}\tau(\overline{x}z.Q) = \textbf{do}\,\{&checkx \leftarrow \texttt{newMVar}\ ();\\ &\texttt{putMVar}\ (unchan\, x)\ (z, checkx);\\ &\texttt{putMVar}\ checkx\ (); \tau(Q)\}\end{aligned}$$

$$\begin{aligned}\tau(x(y).P) = \textbf{do}\,\{&(y, checkx) \leftarrow \texttt{takeMVar}\ (unchan\, x);\\ &\texttt{takeMVar}\ checkx; \tau(P)\}\end{aligned}$$



Translation with Private MVar

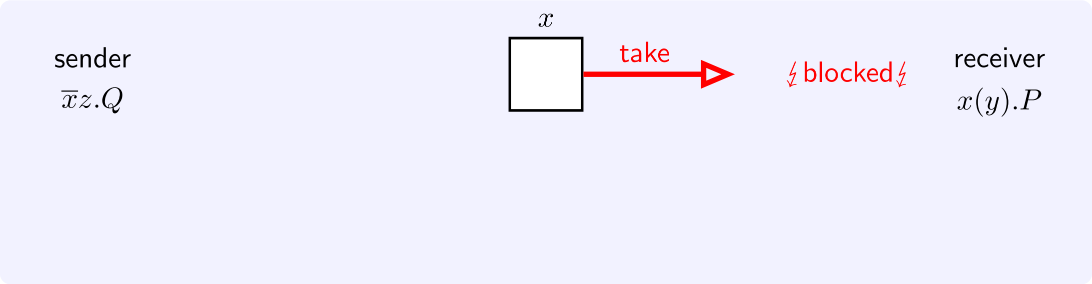
π-calculus-channels are translated into

$$\texttt{data Channel} = \texttt{Chan}\ (\texttt{MVar}\ (\texttt{Channel}, \texttt{MVar}\ ()))$$

$\tau(\nu x.P) = \textbf{do}\,\{chanx \leftarrow newEmptyMVar; \texttt{letrec}\ x = \texttt{Chan}\, chanx\ \texttt{in}\ \tau(P)\}$

$\tau(\overline{x}z.Q) = \textbf{do}\,\{checkx \leftarrow \texttt{newMVar}\ ();$
$\texttt{putMVar}\ (unchan\, x)\ (z, checkx);$
$\texttt{putMVar}\ checkx\ (); \tau(Q)\}$

$\tau(x(y).P) = \textbf{do}\,\{(y, checkx) \leftarrow \texttt{takeMVar}\ (unchan\, x);$
$\texttt{takeMVar}\ checkx; \tau(P)\}$



Translation with Private MVar

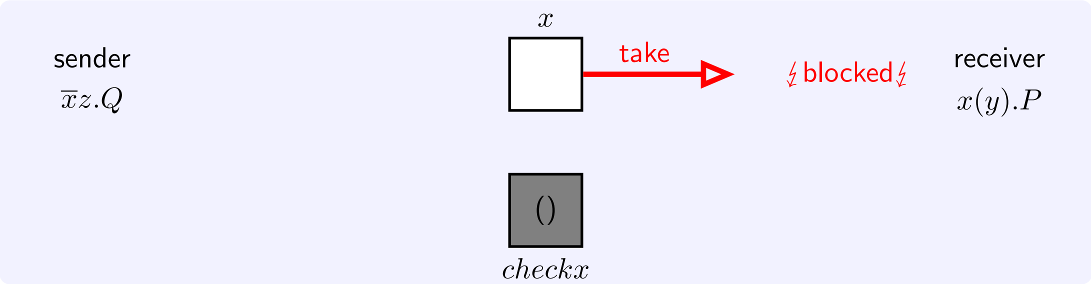
π-calculus-channels are translated into

$$\texttt{data Channel} = \texttt{Chan}\ (\texttt{MVar}\ (\texttt{Channel}, \texttt{MVar}\ ()))$$

$$\tau(\nu x.P) = \textbf{do}\ \{chanx \leftarrow newEmptyMVar; \texttt{letrec}\ x = \texttt{Chan}\ chanx\ \texttt{in}\ \tau(P)\}$$

$$\begin{aligned}\tau(\overline{x}z.Q) = \textbf{do}\ \{&checkx \leftarrow \texttt{newMVar}\ ();\\ &\texttt{putMVar}\ (unchan\ x)\ (z, checkx);\\ &\texttt{putMVar}\ checkx\ (); \tau(Q)\}\end{aligned}$$

$$\begin{aligned}\tau(x(y).P) = \textbf{do}\ \{&(y, checkx) \leftarrow \texttt{takeMVar}\ (unchan\ x);\\ &\texttt{takeMVar}\ checkx; \tau(P)\}\end{aligned}$$



Translation with Private MVar

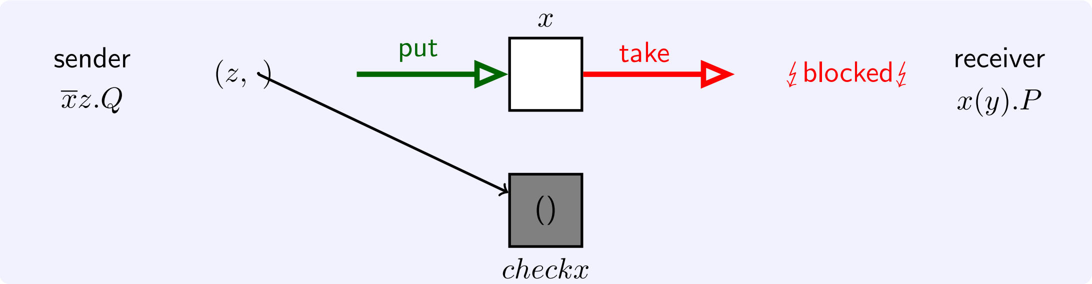
π-calculus-channels are translated into

$$\texttt{data Channel} = \texttt{Chan}\ (\texttt{MVar}\ (\texttt{Channel}, \texttt{MVar}\ ()))$$

$$\tau(\nu x.P) = \textbf{do}\ \{chanx \leftarrow newEmptyMVar; \texttt{letrec}\ x = \texttt{Chan}\ chanx\ \texttt{in}\ \tau(P)\}$$

$$\begin{aligned}\tau(\overline{x}z.Q) = \textbf{do}\ \{&checkx \leftarrow \texttt{newMVar}\ ();\\ &\texttt{putMVar}\ (unchan\ x)\ (z, checkx);\\ &\texttt{putMVar}\ checkx\ (); \tau(Q)\}\end{aligned}$$

$$\begin{aligned}\tau(x(y).P) = \textbf{do}\ \{&(y, checkx) \leftarrow \texttt{takeMVar}\ (unchan\ x);\\ &\texttt{takeMVar}\ checkx; \tau(P)\}\end{aligned}$$



Translation with Private MVar

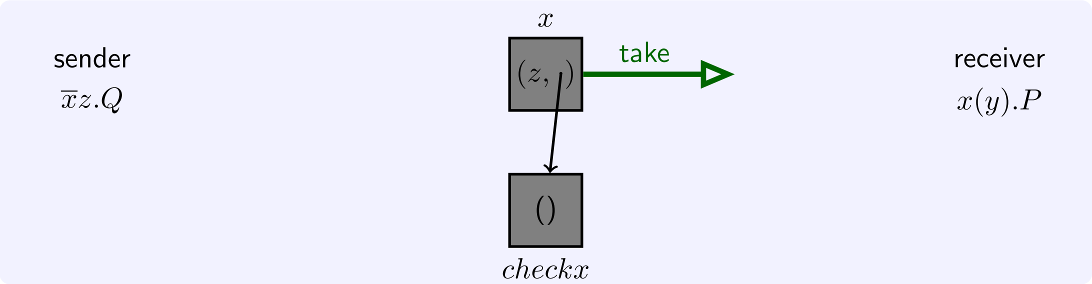
π-calculus-channels are translated into

$$\texttt{data Channel} = \texttt{Chan}\ (\texttt{MVar}\ (\texttt{Channel}, \texttt{MVar}\ ()))$$

$$\tau(\nu x.P) = \textbf{do}\,\{chanx \leftarrow newEmptyMVar; \texttt{letrec}\ x = \texttt{Chan}\, chanx\ \texttt{in}\ \tau(P)\}$$

$$\begin{aligned}\tau(\overline{x}z.Q) = \textbf{do}\,\{&checkx \leftarrow \texttt{newMVar}\ ();\\ &\texttt{putMVar}\ (unchan\,x)\ (z, checkx);\\ &\texttt{putMVar}\ checkx\ (); \tau(Q)\}\end{aligned}$$

$$\begin{aligned}\tau(x(y).P) = \textbf{do}\,\{&(y, checkx) \leftarrow \texttt{takeMVar}\ (unchan\,x);\\ &\texttt{takeMVar}\ checkx; \tau(P)\}\end{aligned}$$



Translation with Private MVar

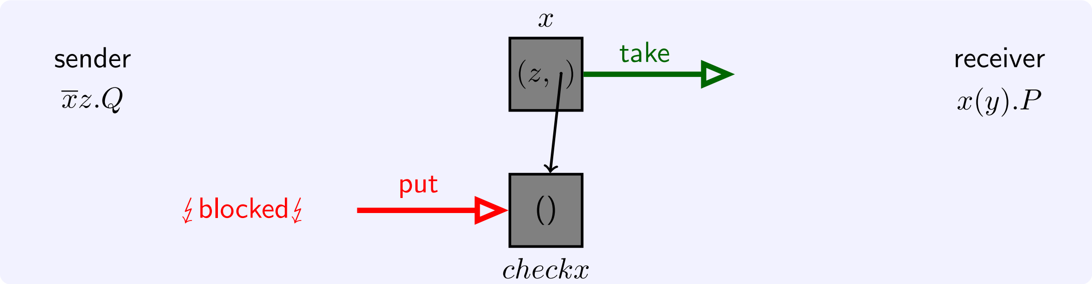
π-calculus-channels are translated into

$$\texttt{data Channel} = \texttt{Chan}\ (\texttt{MVar}\ (\texttt{Channel}, \texttt{MVar}\ ()))$$

$$\tau(\nu x.P) = \textbf{do}\ \{chanx \leftarrow newEmptyMVar; \texttt{letrec}\ x = \texttt{Chan}\ chanx\ \texttt{in}\ \tau(P)\}$$

$$\begin{aligned}\tau(\overline{x}z.Q) = \textbf{do}\ \{&checkx \leftarrow \texttt{newMVar}\ ();\\ &\texttt{putMVar}\ (unchan\ x)\ (z, checkx);\\ &\texttt{putMVar}\ checkx\ (); \tau(Q)\}\end{aligned}$$

$$\begin{aligned}\tau(x(y).P) = \textbf{do}\ \{&(y, checkx) \leftarrow \texttt{takeMVar}\ (unchan\ x);\\ &\texttt{takeMVar}\ checkx; \tau(P)\}\end{aligned}$$



Translation with Private MVar

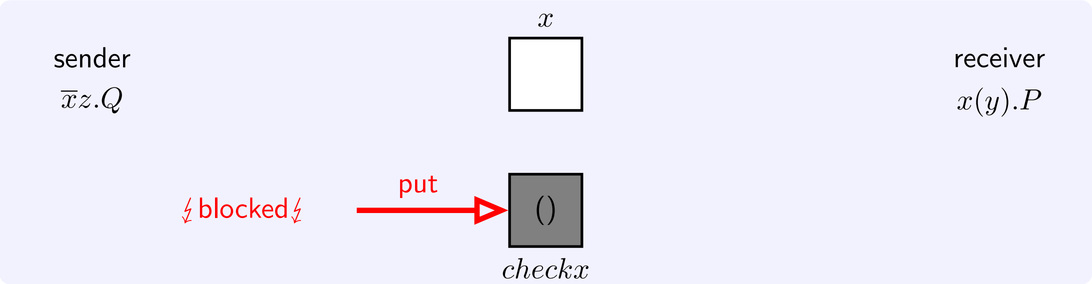
π-calculus-channels are translated into

$$\texttt{data Channel} = \texttt{Chan}\ (\texttt{MVar}\ (\texttt{Channel}, \texttt{MVar}\ ()))$$

$\tau(\nu x.P) = \textbf{do}\,\{chanx \leftarrow newEmptyMVar; \texttt{letrec}\ x = \texttt{Chan}\, chanx\ \texttt{in}\ \tau(P)\}$

$$\begin{aligned}\tau(\overline{x}z.Q) = \textbf{do}\,\{&checkx \leftarrow \texttt{newMVar}\ ();\\ &\texttt{putMVar}\ (unchan\, x)\ (z, checkx);\\ &\texttt{putMVar}\ checkx\ (); \tau(Q)\}\end{aligned}$$

$$\begin{aligned}\tau(x(y).P) = \textbf{do}\,\{&(y, checkx) \leftarrow \texttt{takeMVar}\ (unchan\, x);\\ &\texttt{takeMVar}\ checkx; \tau(P)\}\end{aligned}$$



Translation with Private MVar

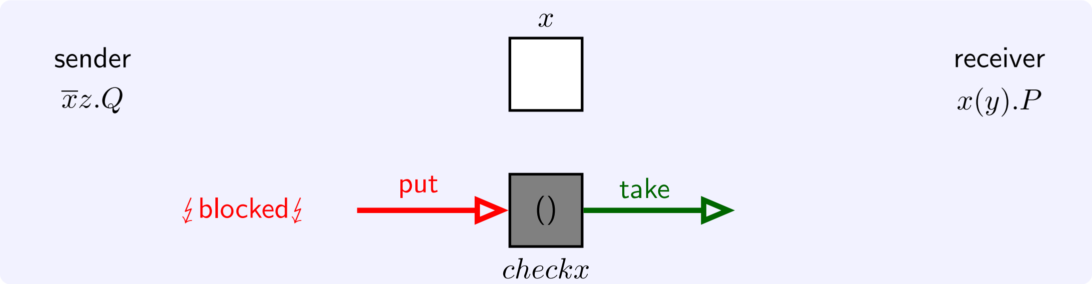
π-calculus-channels are translated into

$$\texttt{data Channel} = \texttt{Chan}\ (\texttt{MVar}\ (\texttt{Channel}, \texttt{MVar}\ ()))$$

$$\tau(\nu x.P) = \textbf{do}\ \{chanx \leftarrow newEmptyMVar; \texttt{letrec}\ x = \texttt{Chan}\ chanx\ \texttt{in}\ \tau(P)\}$$

$$\begin{aligned}\tau(\overline{x}z.Q) = \textbf{do}\ \{&checkx \leftarrow \texttt{newMVar}\ ();\\ &\texttt{putMVar}\ (unchan\, x)\ (z, checkx);\\ &\texttt{putMVar}\ checkx\ (); \tau(Q)\}\end{aligned}$$

$$\begin{aligned}\tau(x(y).P) = \textbf{do}\ \{&(y, checkx) \leftarrow \texttt{takeMVar}\ (unchan\, x);\\ &\texttt{takeMVar}\ checkx; \tau(P)\}\end{aligned}$$



Translation with Private MVar

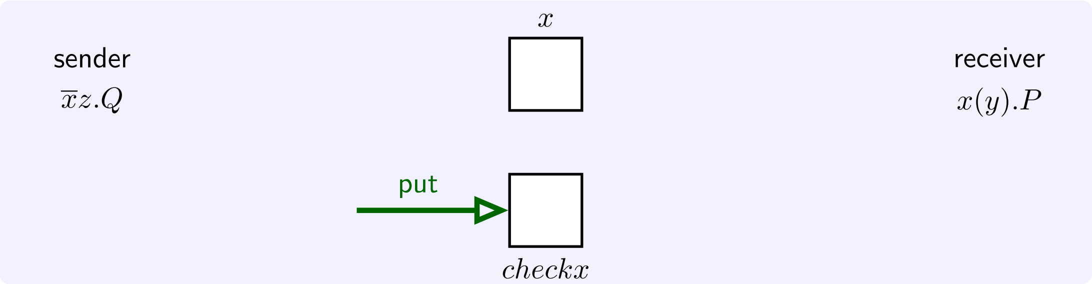
π-calculus-channels are translated into

$$\texttt{data Channel} = \texttt{Chan}\ (\texttt{MVar}\ (\texttt{Channel}, \texttt{MVar}\ ()))$$

$$\tau(\nu x.P) = \textbf{do}\ \{chanx \leftarrow newEmptyMVar; \texttt{letrec}\ x = \texttt{Chan}\ chanx\ \texttt{in}\ \tau(P)\}$$

$$\begin{aligned}\tau(\overline{x}z.Q) = \textbf{do}\ \{&checkx \leftarrow \texttt{newMVar}\ ();\\ &\texttt{putMVar}\ (unchan\, x)\ (z, checkx);\\ &\texttt{putMVar}\ checkx\ (); \tau(Q)\}\end{aligned}$$

$$\begin{aligned}\tau(x(y).P) = \textbf{do}\ \{&(y, checkx) \leftarrow \texttt{takeMVar}\ (unchan\, x);\\ &\texttt{takeMVar}\ checkx; \tau(P)\}\end{aligned}$$



Translation with Private MVar

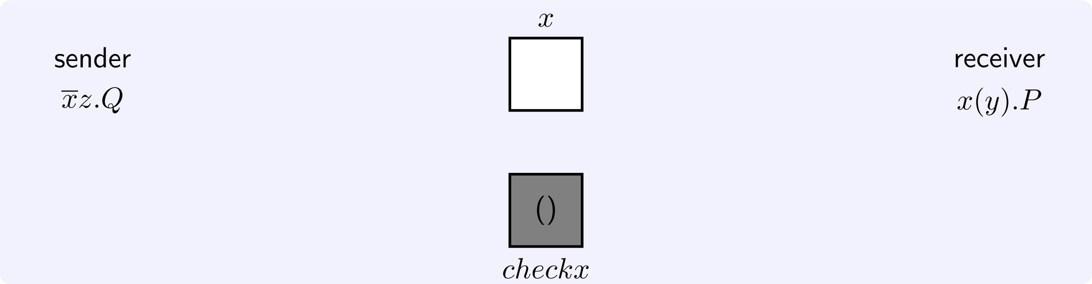
π-calculus-channels are translated into

$$\texttt{data Channel} = \texttt{Chan}\ (\texttt{MVar}\ (\texttt{Channel}, \texttt{MVar}\ ()))$$

$$\tau(\nu x.P) = \textbf{do}\ \{chanx \leftarrow newEmptyMVar; \texttt{letrec}\ x = \texttt{Chan}\ chanx\ \texttt{in}\ \tau(P)\}$$

$$\begin{aligned}\tau(\overline{x}z.Q) = \textbf{do}\ \{&checkx \leftarrow \texttt{newMVar}\ ();\\ &\texttt{putMVar}\ (unchan\ x)\ (z, checkx);\\ &\texttt{putMVar}\ checkx\ (); \tau(Q)\}\end{aligned}$$

$$\begin{aligned}\tau(x(y).P) = \textbf{do}\ \{&(y, checkx) \leftarrow \texttt{takeMVar}\ (unchan\ x);\\ &\texttt{takeMVar}\ checkx; \tau(P)\}\end{aligned}$$



Full Translation

$$
\begin{array}{ll}
\tau_0(P) & = C_{out}^{\tau}[\tau(p)] \\
\tau(\texttt{Stop}) & = \texttt{takeMVar}\ stop \\
\tau(0) & = \texttt{return}\ () \\
\tau(P \mid Q) & = \textbf{do}\ \{\texttt{forkIO}\ \tau(Q); \tau(P)\} \\
\tau(!P) & = \texttt{letrec}\ f = \textbf{do}\ \{\texttt{forkIO}\ \tau(P); f\}\ \texttt{in}\ f \\
\tau(\nu x.P) & = \textbf{do}\,\{chanx \leftarrow newEmptyMVar; \texttt{letrec}\ x = \texttt{Chan}\, chanx\ \texttt{in}\ \tau(P)\} \\
\tau(\overline{x}z.Q) & = \textbf{do}\,\{checkx \leftarrow \texttt{newMVar}\ (); \\
 & \qquad\quad \texttt{putMVar}\ (unchan\, x)\ (z, checkx); \\
 & \qquad\quad \{\texttt{putMVar}\ checkx\ (); \tau(Q)\} \\
\tau(x(y).P) & = \textbf{do}\,\{(y, checkx) \leftarrow \texttt{takeMVar}\ (unchan\, x); \\
 & \qquad\quad \texttt{takeMVar}\ checkx; \tau(P)\}
\end{array}
$$

Correctness of Translation τ

Theorem (Convergence Equivalence)

For closed $P \in \Pi_{\mathtt{Stop}}$: $P{\downarrow} \iff C^{\tau}_{out}[\tau(P)]{\downarrow}$ and $P{\Downarrow} \iff C^{\tau}_{out}[\tau(P)]{\Downarrow}$

Proof consists of four parts:

- ("$\downarrow \Rightarrow \downarrow$") $P \xrightarrow{sr,*} P', P'$ successful $\implies \exists Q : C^{\tau}_{out}[\tau(P)] \xrightarrow{sr,*} Q, Q$ successful.
- ("$\downarrow \Leftarrow \downarrow$") $C^{\tau}_{out}[\tau(P)] \xrightarrow{sr,*} Q, Q$ successful $\implies \exists P' : P \xrightarrow{sr,*} P', P'$ successful
- ("$\Downarrow \Leftarrow \Downarrow$") $P \xrightarrow{sr,*} P', P'{\Uparrow} \implies \exists Q : C^{\tau}_{out}[\tau(P)] \xrightarrow{sr,*} Q, Q{\Uparrow}$.
- ("$\Downarrow \Rightarrow \Downarrow$") $C^{\tau}_{out}[\tau(P)] \xrightarrow{sr,*} Q, Q{\Uparrow} \implies \exists P' : P \xrightarrow{sr,*} P', P'{\Uparrow}$

All parts require to inductively construct reduction sequences from given ones.

For parts ("$\downarrow \Leftarrow \downarrow$") and ("$\Downarrow \Rightarrow \Downarrow$"), the given sequences $C^{\tau}_{out}[\tau(P)] \xrightarrow{sr,*} Q$ have to be reordered, cut and/or extended to "back-translate" them.

Correctness of Translation τ (Cont'd)

Theorem (Adequacy)

Translation τ **is adequate**, i.e. for all $P, P' \in \Pi_{\texttt{Stop}}$: $\tau(P) \sim_{c,\tau_0} \tau(P') \implies P \sim_c P'$

Theorem

The translation τ is **not** fully abstract ($P \sim_c P' \iff \tau(P) \sim_{c,\tau_0} \tau(P')$).
On **closed** processes P, P': $P \sim_c P' \iff \tau(P) \sim_{c,\tau_0} \tau(P')$

where $e_1 \sim_{c,\tau_0} e_2$ iff for all $C : FV(C[e_1]) \cup FV(C[e_2]) \subseteq \{stop\}$:
$C^{\tau}_{out}[C[e_1]]{\downarrow} \iff C^{\tau}_{out}[C[e_2]]{\downarrow}$ and $C^{\tau}_{out}[C[e_1]]{\Downarrow} \iff C^{\tau}_{out}[C[e_2]]{\Downarrow}$

Translations with Global MVars

Ideas:

- Translation of $\texttt{stop}$, 0, $\mid$, $!$ as before
- π-calculus-channels are translated into data of type

$$\texttt{data Channel} = \texttt{Chan} \underbrace{\texttt{(MVar Channel)}}_{\text{content}} \underbrace{\texttt{(MVar ())} \ldots \texttt{(MVar ())}}_{\text{check-MVars}}$$

  i.e. channel x becomes a binding $x = \texttt{Chan}\ \mathit{content}\ \mathit{check}_1\ \ldots\ \mathit{check}_n$

- MVars $\mathit{content}$, $\mathit{check}_1, \ldots, \mathit{check}_n$ are created once and are (globally) visible via x
- Programs for sender $\overline{x}z$ and receiver $x(y)$ are **restricted**:
  - They exchange the message via the content-MVar
  - They perform `takeMVar` & `putMVar` on the check-MVars for synchronisation

Translations with Global MVars (Cont'd)

Reminder: a channel x becomes a binding $x = \texttt{Chan}\ content\ check_1\ \ldots\ check_n$

Questions:

- Are there correct translations under these restrictions?
- How many check-MVars are required?
- What is the smallest correct translation?

Approach:

- enumerate all translations and automatically search for counter-examples
- check correctness of the remaining (potentially correct) translations by hand

Conjecture (Proved in the meantime, not yet published)

With the described restrictions two check-MVars are required.

Correct Translation with Two Check-MVars

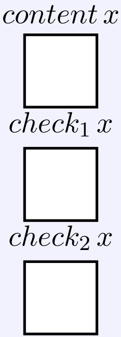
$$T_1(\overline{x}z.Q) = \textbf{do}\ \{\texttt{putMVar}\ (check_1\, x)\, (),\ \texttt{putMVar}\, (content\, x)\, z;\ \texttt{takeMVar}\, (check_2\, x);\ \texttt{takeMVar}\, (check_1\, x); T_1(Q)\}$$

$$T_1(x(y).P) = \textbf{do}\ \{y \leftarrow \texttt{takeMVar}\, (content\, x);\ \texttt{putMVar}\, (check_2\, x); T_1(P)\}$$

Correct Translation with Two Check-MVars

$$T_1(\overline{x}z.Q) = \textbf{do}\ \{\texttt{putMVar}\ (\mathit{check}_1\ x)\ (), \\ \texttt{putMVar}\ (\mathit{content}\ x)\ z; \\ \texttt{takeMVar}\ (\mathit{check}_2\ x); \\ \texttt{takeMVar}\ (\mathit{check}_1\ x); T_1(Q)\}$$

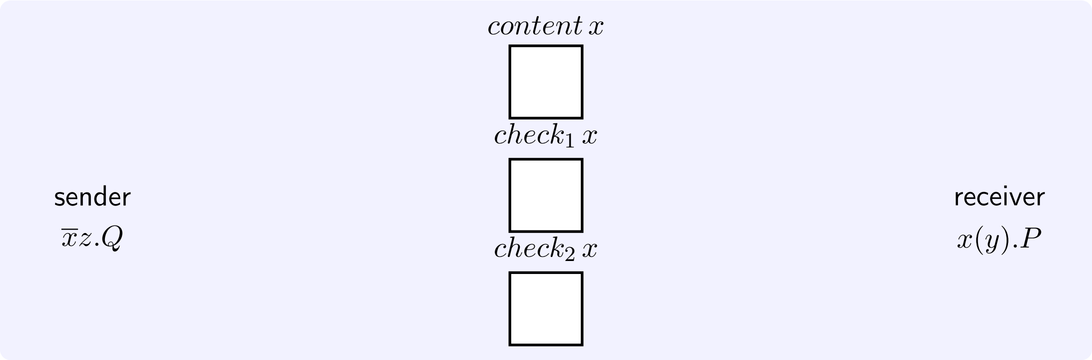
$$T_1(x(y).P) = \textbf{do}\ \{y \leftarrow \texttt{takeMVar}\ (\mathit{content}\ x); \\ \texttt{putMVar}\ (\mathit{check}_2\ x); T_1(P)\}$$



Correct Translation with Two Check-MVars

$$T_1(\overline{x}z.Q) = \textbf{do}\ \{\texttt{putMVar}\ (\mathit{check}_1\ x)\ (), \\ \texttt{putMVar}\ (\mathit{content}\ x)\ z; \\ \texttt{takeMVar}\ (\mathit{check}_2\ x); \\ \texttt{takeMVar}\ (\mathit{check}_1\ x); T_1(Q)\}$$

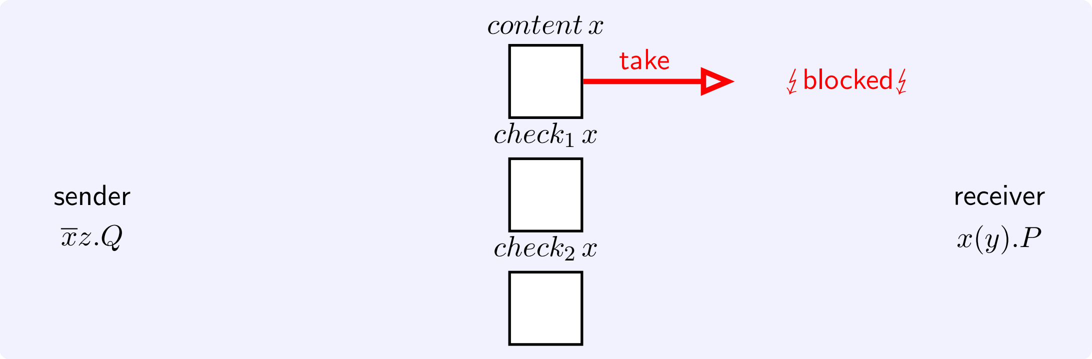
$$T_1(x(y).P) = \textbf{do}\ \{y \leftarrow \texttt{takeMVar}\ (\mathit{content}\ x); \\ \texttt{putMVar}\ (\mathit{check}_2\ x); T_1(P)\}$$



Correct Translation with Two Check-MVars

$T_1(\overline{x}z.Q) = \textbf{do}\ \{\texttt{putMVar}\ (check_1\ x)\ (),$
$\quad \texttt{putMVar}\ (content\ x)\ z;$
$\quad \texttt{takeMVar}\ (check_2\ x);$
$\quad \texttt{takeMVar}\ (check_1\ x); T_1(Q)\}$

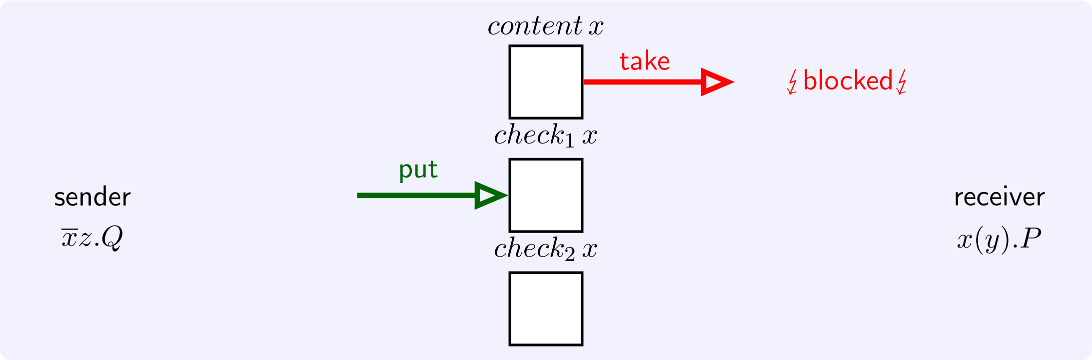
$T_1(x(y).P) = \textbf{do}\ \{y \leftarrow \texttt{takeMVar}\ (content\ x);$
$\quad \texttt{putMVar}\ (check_2\ x); T_1(P)\}$



Correct Translation with Two Check-MVars

$$T_1(\overline{x}z.Q) = \textbf{do}\ \{\texttt{putMVar}\ (\mathit{check}_1\ x)\ (),\ \texttt{putMVar}\ (\mathit{content}\ x)\ z;\ \texttt{takeMVar}\ (\mathit{check}_2\ x);\ \texttt{takeMVar}\ (\mathit{check}_1\ x); T_1(Q)\}$$

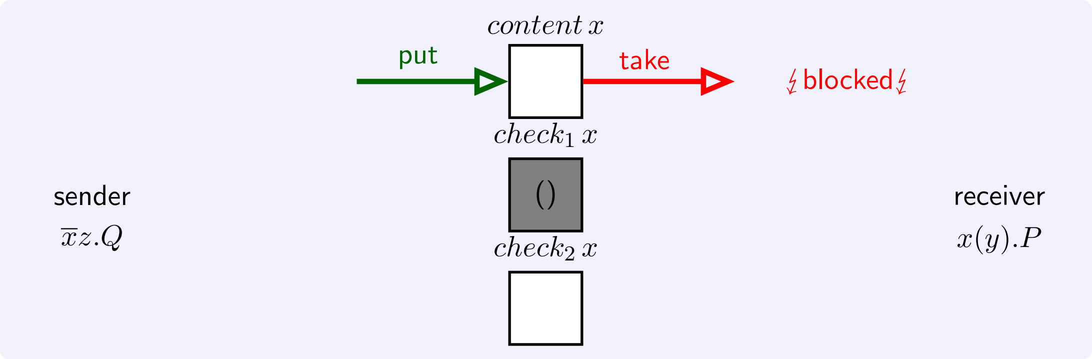
$$T_1(x(y).P) = \textbf{do}\ \{y \leftarrow \texttt{takeMVar}\ (\mathit{content}\ x);\ \texttt{putMVar}\ (\mathit{check}_2\ x); T_1(P)\}$$



Correct Translation with Two Check-MVars

$$T_1(\overline{x}z.Q) = \textbf{do}\ \{\texttt{putMVar}\ (check_1\ x)\ (),\ \texttt{putMVar}\ (content\ x)\ z;\ \texttt{takeMVar}\ (check_2\ x);\ \texttt{takeMVar}\ (check_1\ x); T_1(Q)\}$$

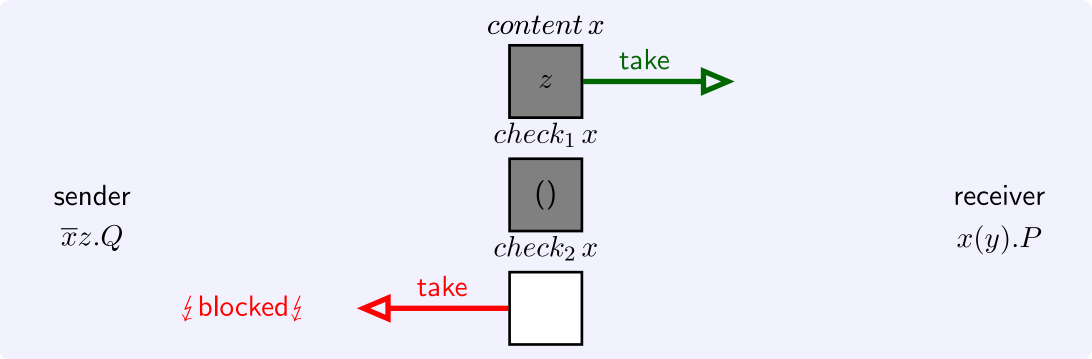
$$T_1(x(y).P) = \textbf{do}\ \{y \leftarrow \texttt{takeMVar}\ (content\ x);\ \texttt{putMVar}\ (check_2\ x); T_1(P)\}$$



Correct Translation with Two Check-MVars

$$T_1(\overline{x}z.Q) = \textbf{do}\ \{\texttt{putMVar}\ (\mathit{check}_1\ x)\ (),\\ \texttt{putMVar}\ (\mathit{content}\ x)\ z;\\ \texttt{takeMVar}\ (\mathit{check}_2\ x);\\ \texttt{takeMVar}\ (\mathit{check}_1\ x); T_1(Q)\}$$

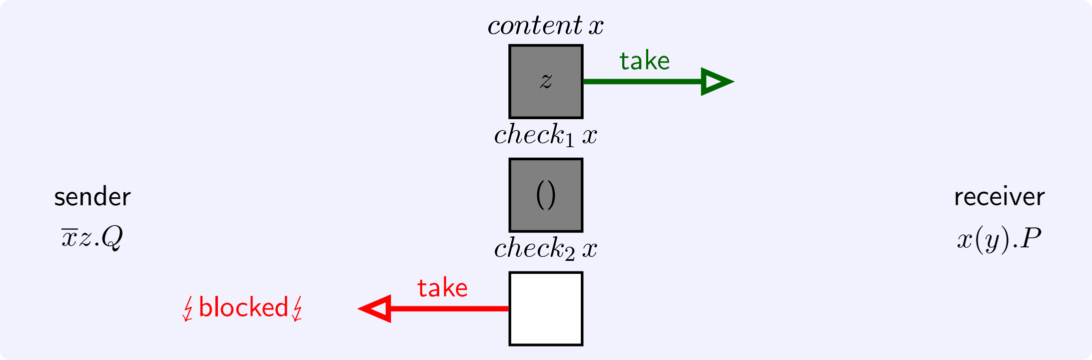
$$T_1(x(y).P) = \textbf{do}\ \{y \leftarrow \texttt{takeMVar}\ (\mathit{content}\ x);\\ \texttt{putMVar}\ (\mathit{check}_2\ x); T_1(P)\}$$



Correct Translation with Two Check-MVars

$$T_1(\overline{x}z.Q) = \textbf{do}\ \{\texttt{putMVar}\ (\mathit{check}_1\, x)\ (), \\ \texttt{putMVar}\,(\mathit{content}\, x)\ z; \\ \texttt{takeMVar}\,(\mathit{check}_2\, x); \\ \texttt{takeMVar}\,(\mathit{check}_1\, x); T_1(Q)\}$$

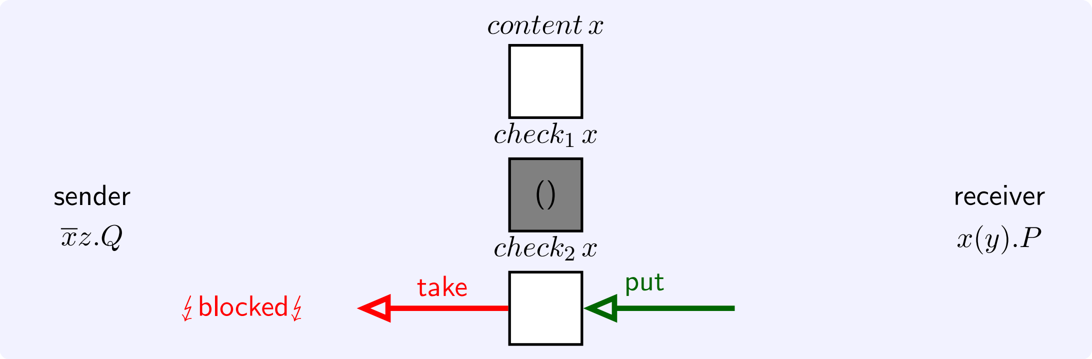
$$T_1(x(y).P) = \textbf{do}\ \{y \leftarrow \texttt{takeMVar}\,(\mathit{content}\, x); \\ \texttt{putMVar}\,(\mathit{check}_2\, x); T_1(P)\}$$



Correct Translation with Two Check-MVars

$$T_1(\overline{x}z.Q) = \textbf{do}\ \{\texttt{putMVar}\ (\mathit{check}_1\ x)\ (),\ \texttt{putMVar}\,(\mathit{content}\,x)\ z;\ \texttt{takeMVar}\,(\mathit{check}_2\,x);\ \texttt{takeMVar}\,(\mathit{check}_1\,x); T_1(Q)\}$$

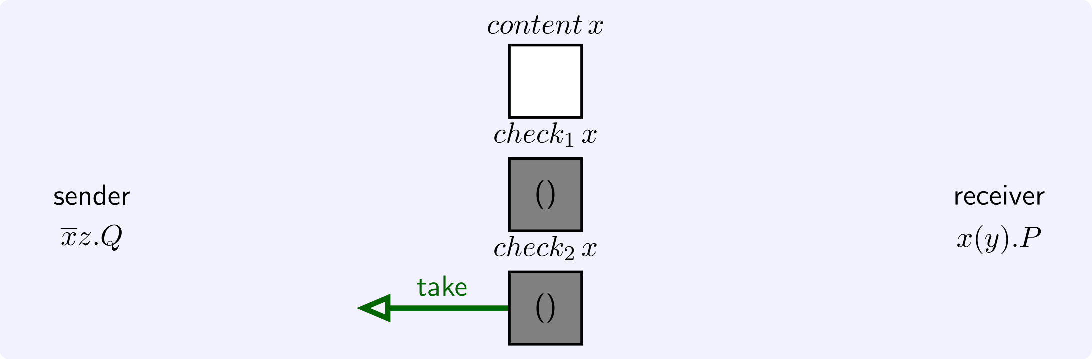
$$T_1(x(y).P) = \textbf{do}\ \{y \leftarrow \texttt{takeMVar}\,(\mathit{content}\,x);\ \texttt{putMVar}\,(\mathit{check}_2\,x); T_1(P)\}$$



Correct Translation with Two Check-MVars

$T_1(\overline{x}z.Q) = \textbf{do}\ \{\texttt{putMVar}\ (\mathit{check}_1\ x)\ (),$
$\quad \texttt{putMVar}\ (\mathit{content}\ x)\ z;$
$\quad \texttt{takeMVar}\ (\mathit{check}_2\ x);$
$\quad \texttt{takeMVar}\ (\mathit{check}_1\ x); T_1(Q)\}$

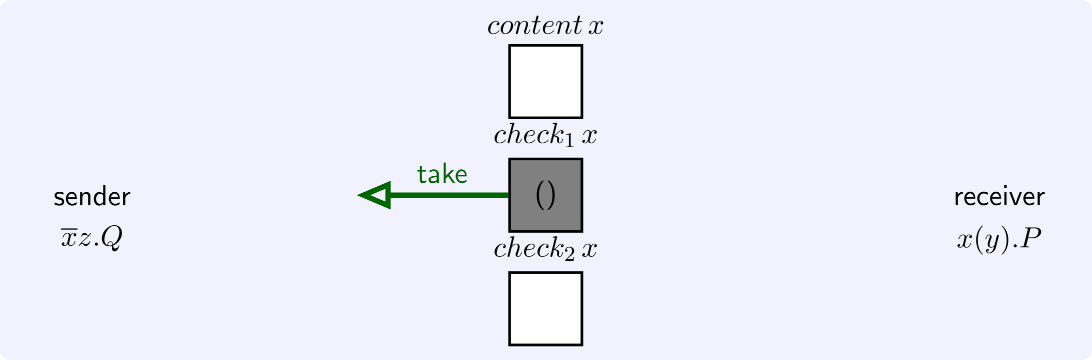
$T_1(x(y).P) = \textbf{do}\ \{y \leftarrow \texttt{takeMVar}\ (\mathit{content}\ x);$
$\quad \texttt{putMVar}\ (\mathit{check}_2\ x); T_1(P)\}$



Correct Translation with Two Check-MVars

$$T_1(\overline{x}z.Q) = \textbf{do}\ \{\texttt{putMVar}\ (check_1\ x)\ (),\ \texttt{putMVar}\ (content\ x)\ z;\ \texttt{takeMVar}\ (check_2\ x);\ \texttt{takeMVar}\ (check_1\ x); T_1(Q)\}$$

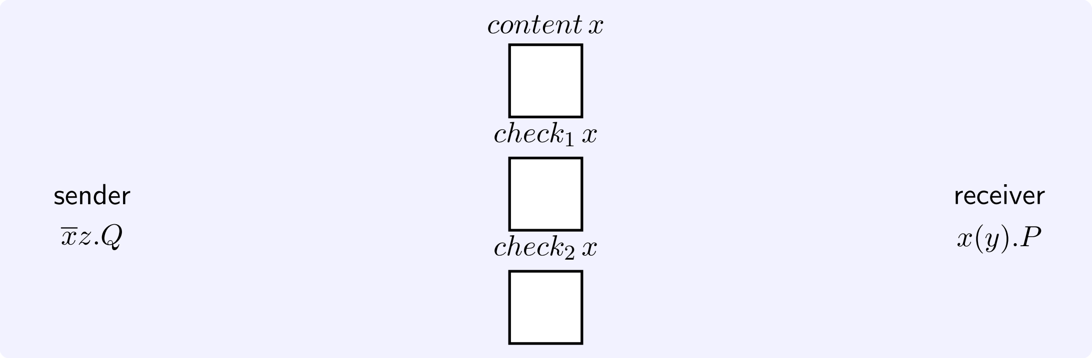
$$T_1(x(y).P) = \textbf{do}\ \{y \leftarrow \texttt{takeMVar}\ (content\ x);\ \texttt{putMVar}\ (check_2\ x); T_1(P)\}$$



Correct Translation with Two Check-MVars

$$T_1(\overline{x}z.Q) = \textbf{do}\ \{\texttt{putMVar}\ (check_1\ x)\ (),\ \texttt{putMVar}\ (content\ x)\ z;\ \texttt{takeMVar}\ (check_2\ x);\ \texttt{takeMVar}\ (check_1\ x); T_1(Q)\}$$

$$T_1(x(y).P) = \textbf{do}\ \{y \leftarrow \texttt{takeMVar}\ (content\ x);\ \texttt{putMVar}\ (check_2\ x); T_1(P)\}$$



Correct Translation with Two Check-MVars

$$T_1(\overline{x}z.Q) = \textbf{do}\ \{\texttt{putMVar}\ (\mathit{check}_1\ x)\ (),\ \texttt{putMVar}\ (\mathit{content}\ x)\ z;\ \texttt{takeMVar}\ (\mathit{check}_2\ x);\ \texttt{takeMVar}\ (\mathit{check}_1\ x); T_1(Q)\}$$

$$T_1(x(y).P) = \textbf{do}\ \{y \leftarrow \texttt{takeMVar}\ (\mathit{content}\ x);\ \texttt{putMVar}\ (\mathit{check}_2\ x); T_1(P)\}$$

Theorem

T_1 is convergence-equivalent, adequate, and on closed processes also fully-abstract.

Main arguments:

- MVar $(\mathit{check}_1\ x)$ is used as a mutex for the receivers on x
- execution of the sender/receiver protocol is non-overlapping

Translations with Global MVars and Interprocess Restriction

Interprocess restriction:

One put/take-pair for each check-MVar and it is distributed between sender/receiver.

Translations with Global MVars and Interprocess Restriction

Interprocess restriction:

One put/take-pair for each check-MVar and it is distributed between sender/receiver.

Theorem

Under the interprocess restriction, **three** check-MVars are **necessary** and **sufficient**.

Correct translation:

$$T_2(\overline{x}z.Q) = \textbf{do}\ \{\texttt{putMVar}\,(content\,x)\,z;\ \texttt{putMVar}\,(check_1\,x)\,();\ \texttt{takeMVar}\,(check_2\,x);\ \texttt{putMVar}\,(check_3\,x)\,();T_2(Q)\}$$

$$T_2(x(y).P) = \textbf{do}\ \{\texttt{takeMVar}\,(check_1\,x);\ \texttt{putMVar}\,(check_2\,x);\ \texttt{takeMVar}\,(check_3\,x);\ y \leftarrow \texttt{takeMVar}\,(content\,x);T_2(P)\}$$

Results of the automated search for counter-examples:

- for 1 check-MVar 8 of 8 translations are refuted
- for 2 check-MVars 72 of 72 translations are refuted
- for 3 check-MVars 762 of 768 translations are refuted

Conclusion & Future Work

Conclusion

- Correct translations from $\Pi_{\texttt{Stop}}$ into Concurrent Haskell
- Translation with private MVars
- Smallest translations with global MVars
- Translations are convergence equivalent and adequate (fully abstract on closed processes)
- Refuted incorrect translations by automated search for counter-examples

Future Work

- Variations and extensions of $\Pi_{\texttt{Stop}}$ (recursion, sums, name matching, ...)
- Other target languages?
- Publish proof of conjecture