LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Minimal Translations from Synchronous Communication to Synchronizing Locks

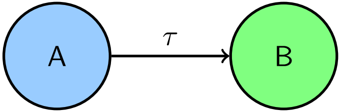
Manfred Schmidt-Schauß
Goethe-University Frankfurt

David Sabel
LMU Munich

EXPRESS/SOS 2021
August 23, 2021

General Motivation

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

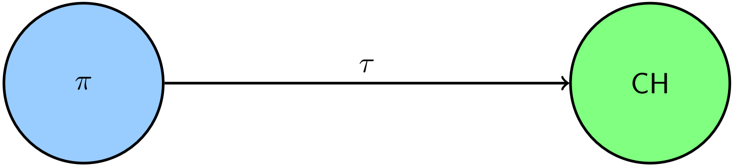


- In particular we consider **concurrent** programming languages
- Questions:
  - **expressivity**: can language B express language A?
  - **correctness of implementations**:
    is the implementation of concurrency primitives of A in language B correct?

Previous Work

Previous work (EXPRESS/SOS 2020):

- Correct translations from the synchronous π-calculus into Concurrent Haskell



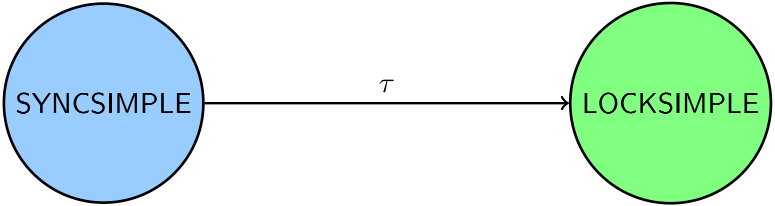
synchronous communication
via message passing
named channels, messages,
mobility, replication

shared memory concurrency with
synchronising variables (MVars)
concurrent λ-calculus with recursive let,
data & case-expressions, monadic I/O

- Correctness w.r.t. observational semantics
- Both models are quite specific, in particular MVars

In this Work

- Analyse translations from synchronous communication to (synchronous) shared memory
- In a minimal setting: source and target are really simple languages



synchronous communication
one global channel
no names, no messages, no replication,
all reductions are finite

synchronous locks
(similar to binary semaphores)
no λ-calculus, no recursion, no data,
all reductions are finite

- Main question:

  What is the minimal number of locks that is required for a correct translation?

Source Calculus: SYNCSIMPLE

Subprocesses:

$$\mathcal{U} ::= \underset{\text{(success)}}{\checkmark} \mid \underset{\text{(silence)}}{\mathbf{0}} \mid \underset{\text{(send)}}{!\mathcal{U}} \mid \underset{\text{(receive)}}{?\mathcal{U}}$$

Processes:

$$\mathcal{P} ::= \underset{\text{(subprocess)}}{\mathcal{U}} \mid \underset{\text{(parallel composition)}}{\mathcal{U} \mathbin{|} \mathcal{P}}$$

Operational semantics: $!\mathcal{U}_1 \mathbin{|} ?\mathcal{U}_2 \mathbin{|} \mathcal{P} \xrightarrow{SYS} \mathcal{U}_1 \mathbin{|} \mathcal{U}_2 \mathbin{|} \mathcal{P}$

Source Calculus: SYNCSIMPLE

Subprocesses:

$$\mathcal{U} ::= \underset{\text{(success)}}{\checkmark} \mid \underset{\text{(silence)}}{\mathbf{0}} \mid \underset{\text{(send)}}{!\mathcal{U}} \mid \underset{\text{(receive)}}{?\mathcal{U}}$$

Processes:

$$\mathcal{P} ::= \underset{\text{(subprocess)}}{\mathcal{U}} \mid \underset{\text{(parallel composition)}}{\mathcal{U} \,\mathbf{|}\, \mathcal{P}}$$

Operational semantics: $!\mathcal{U}_1 \,\mathbf{|}\, ?\mathcal{U}_2 \,\mathbf{|}\, \mathcal{P} \xrightarrow{SYS} \mathcal{U}_1 \,\mathbf{|}\, \mathcal{U}_2 \,\mathbf{|}\, \mathcal{P}$

Example:

$$\begin{tikzcd}
 & & & \mathbf{0} \,\mathbf{|}\, !\checkmark \,\mathbf{|}\, \mathbf{0} \\
 & !\mathbf{0} \,\mathbf{|}\, !\checkmark \,\mathbf{|}\, ?\mathbf{0} \arrow[urr, "SYS"] \arrow[drr, "SYS"] & & \\
 & & & !\mathbf{0} \,\mathbf{|}\, \checkmark \,\mathbf{|}\, \mathbf{0} \\
?!\mathbf{0} \,\mathbf{|}\, !!\checkmark \,\mathbf{|}\, ?\mathbf{0} \arrow[uur, "SYS"] \arrow[dr, "SYS"] & & & \\
 & ?!\mathbf{0} \,\mathbf{|}\, !\checkmark \,\mathbf{|}\, \mathbf{0} \arrow[rr, "SYS"] & & !\mathbf{0} \,\mathbf{|}\, \checkmark \,\mathbf{|}\, \mathbf{0}
\end{tikzcd}$$

Source Calculus: SYNCSIMPLE

Subprocesses:

$$\mathcal{U} ::= \underset{\text{(success)}}{\checkmark} \mid \underset{\text{(silence)}}{\mathbf{0}} \mid \underset{\text{(send)}}{!\mathcal{U}} \mid \underset{\text{(receive)}}{?\mathcal{U}}$$

Processes:

$$\mathcal{P} ::= \underset{\text{(subprocess)}}{\mathcal{U}} \mid \underset{\text{(parallel composition)}}{\mathcal{U} \,|\, \mathcal{P}}$$

Operational semantics: $!\mathcal{U}_1 \,|\, ?\mathcal{U}_2 \,|\, \mathcal{P} \xrightarrow{SYS} \mathcal{U}_1 \,|\, \mathcal{U}_2 \,|\, \mathcal{P}$

Example:

$$\begin{tikzcd}
 & & \mathbf{0} \,|\, !\checkmark \,|\, \mathbf{0} \\
 & !\mathbf{0} \,|\, !\checkmark \,|\, ?\mathbf{0} \arrow[ru, "SYS"] \arrow[rd, "SYS"] & \\
 & & !\mathbf{0} \,|\, \checkmark \,|\, \mathbf{0} \\
?!\mathbf{0} \,|\, !!\checkmark \,|\, ?\mathbf{0} \arrow[ruu, "SYS"] \arrow[rd, "SYS"] & & \\
 & ?!\mathbf{0} \,|\, !\checkmark \,|\, \mathbf{0} \arrow[r, "SYS"] & !\mathbf{0} \,|\, \checkmark \,|\, \mathbf{0}
\end{tikzcd}$$

- $\mathcal{P}$ is successful if $\mathcal{P} = \checkmark \,|\, \mathcal{P}'$
- $\mathcal{P}$ is may-convergent if there is some successful process $\mathcal{P}'$ with $\mathcal{P} \xrightarrow{SYS,*} \mathcal{P}'$.
- $\mathcal{P}$ is must-convergent if for all $\mathcal{P}'$ with $\mathcal{P} \xrightarrow{SYS,*} \mathcal{P}'$, the process $\mathcal{P}'$ is may-convergent.

Target Calculus: $\text{LOCKSIMPLE}_{k,IS}$

Subprocesses:

$\mathcal{U} ::= \checkmark \mid \mathbf{0} \mid P_i\mathcal{U} \mid T_i\mathcal{U}$

(success) (silence) (put on lock i) (take on lock i)

Processes:

$\mathcal{P} ::= \mathcal{U} \mid \mathcal{U} \mathbin{|} \mathcal{P}$

(subprocess) (parallel composition)

Storage: locks $C_1, \ldots, C_k$ which are either $\square$ (empty) or $\blacksquare$ (full), IS is the initial storage

Operational semantics:

$(P_i\mathcal{U} \mathbin{|} \mathcal{P}, \mathcal{C}[C_i = \square]) \xrightarrow{LS} (\mathcal{U} \mathbin{|} \mathcal{P}, \mathcal{C}[C_i \mapsto \blacksquare])$

(put fills an empty lock / blocks on a filled)

$(T_i\mathcal{U} \mathbin{|} \mathcal{P}, \mathcal{C}) \xrightarrow{LS} (\mathcal{U} \mathbin{|} \mathcal{P}, \mathcal{C}[C_i \mapsto \square])$

(take empties the lock, non-blocking)

Target Calculus: $\text{LOCKSIMPLE}_{k,IS}$

Subprocesses:

$$\mathcal{U} ::= \checkmark \mid \mathbf{0} \mid P_i\mathcal{U} \mid T_i\mathcal{U}$$

(success) (silence) (put on lock i) (take on lock i)

Processes:

$$\mathcal{P} ::= \mathcal{U} \mid \mathcal{U} \mathrel{|} \mathcal{P}$$

(subprocess) (parallel composition)

Storage: locks $C_1, \ldots, C_k$ which are either $\square$ (empty) or $\blacksquare$ (full), IS is the initial storage

Operational semantics:

$$(P_i\mathcal{U} \mathrel{|} \mathcal{P}, \mathcal{C}[C_i = \square]) \xrightarrow{LS} (\mathcal{U} \mathrel{|} \mathcal{P}, \mathcal{C}[C_i \mapsto \blacksquare])$$

(put fills an empty lock / blocks on a filled)

$$(T_i\mathcal{U} \mathrel{|} \mathcal{P}, \mathcal{C}) \xrightarrow{LS} (\mathcal{U} \mathrel{|} \mathcal{P}, \mathcal{C}[C_i \mapsto \square])$$

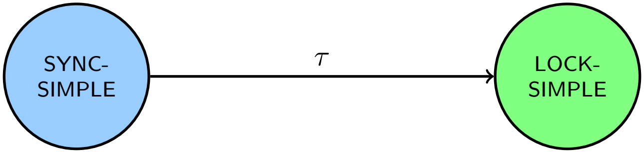
(take empties the lock, non-blocking)

Example:

$$\begin{array}{lll}
 & (P_2P_1\checkmark \mathrel{|} T_1\mathbf{0} \mathrel{|} T_2\mathbf{0}, & (\blacksquare, \blacksquare)) \\
\xrightarrow{LS} & (P_2P_1\checkmark \mathrel{|} T_1\mathbf{0} \mathrel{|} \mathbf{0}, & (\blacksquare, \square)) \\
\xrightarrow{LS} & (P_1\checkmark \mathrel{|} T_1\mathbf{0} \mathrel{|} \mathbf{0}, & (\blacksquare, \blacksquare)) \\
\xrightarrow{LS} & (P_1\checkmark \mathrel{|} \mathbf{0} \mathrel{|} \mathbf{0}, & (\square, \blacksquare)) \\
\xrightarrow{LS} & (\checkmark \mathrel{|} \mathbf{0} \mathrel{|} \mathbf{0}, & (\blacksquare, \blacksquare))
\end{array}$$

- success, may- and must-convergence: analogous, but starting with initial storage IS

Translations



Compositional translations τ

- map $\tau(!)$ and $\tau(?)$ to sequences consisting of P_i- and T_i-operations
- for all other constructs: translation is the identity
  $(\tau(\mathbf{0}) = \mathbf{0}$, $\tau(\checkmark) = \checkmark$, $\tau(\mathcal{P}_1 \mid \mathcal{P}_2) = \tau(\mathcal{P}_1) \mid \tau(\mathcal{P}_2) \ldots)$

Translation τ is correct iff for all $\mathrm{SYNCSIMPLE}$-processes $\mathcal{P}$:

$\mathcal{P}$ is may-convergent iff $\tau(\mathcal{P})$ is may-convergent,
and
$\mathcal{P}$ is must-convergent iff $\tau(\mathcal{P})$ is must-convergent

Results: 3 Locks Suffice

Theorem (correct translation with 3 locks)

For $k = 3$, the translation τ with

$$\tau(!) = P_1 T_3 P_2 T_1 \qquad \text{and} \qquad \tau(?) = P_3 T_2$$

is correct for initial store $(\square, \blacksquare, \blacksquare)$.

- $P_1 \ldots T_1$ ensures that only one sender (atomically) communicates
- T_3 signals that sender is available
- P_2 waits that receiver is available
- P_3 waits that a sender is available
- T_2 signals that receiver is available

We also found other correct translations:

$\tau(!) = P_2 P_1 T_3 P_1 T_1 T_2$ and $\tau(?) = P_3 T_1$ is correct for initial store $(\square, \square, \blacksquare)$.

Results: Minimality

Theorem (1 lock is insufficient)

There is no correct compositional translation $\mathrm{SYNCSIMPLE} \rightarrow \mathrm{LOCKSIMPLE}_{1,IS}$.

Main Theorem (2 locks are insufficient)

There is no correct compositional translation $\mathrm{SYNCSIMPLE} \rightarrow \mathrm{LOCKSIMPLE}_{2,IS}$.

Both theorems hold for any initial storage!

Results: Variations and Open Questions

Variants

- No difference, if we change the blocking behavior
  (i.e. fix for each i: P_i blocks or T_i blocks but not both)
- Reason: we can adapt the initial storage

Results: Variations and Open Questions

Variants

- No difference, if we change the blocking behavior (i.e. fix for each i: P_i blocks or T_i blocks but not both)
- Reason: we can adapt the initial storage

Open cases:

- Blocking put and blocking take: Are 3 locks required?
- Correct translations with 3 locks for each combination of blocking behavior and initial storage

Proof Structure of the Main Theorem

Remember: Main Theorem says that there is no correct compositional translation for 2 locks.

Main idea of the proof: classify the translations by their blocking type:

The blocking type of a correct translation τ is (W_1, W_2) where

- W_1 is the blocking type of $\tau(!\checkmark)$
- W_2 is the blocking type of $\tau(?\checkmark)$

The blocking type of a sequence/subprocess $\mathcal{S}$ is

- P_i if $\mathcal{S} = \mathcal{R}_1 P_i \mathcal{R}_2$, where R_1 does not contain P_i or T_i and a deadlock occurs after executing $\mathcal{R}_1$ on the initial storage IS
- $P_i P_i$ iff $\mathcal{S} = \mathcal{R}_1 P_i \mathcal{R}_2 P_i \mathcal{R}_3$, where $\mathcal{R}_2$ does not contain P_i or T_i, and a deadlock occurs after executing $\mathcal{R}_1 P_i \mathcal{R}_2$ on the initial storage IS

Proof shows impossibility for the blocking types (P_1P_1, P_1P_1), (P_1P_1, P_2P_2), (P_1P_1, P_1), (P_1P_1, P_2), (P_1, P_1), and (P_1, P_2) (other cases are symmetric)

Exemplary Proof

Claim

For a correct translation, the blocking type (P_1P_1, P_1) is impossible

Proof: While $!\checkmark \mid ?\checkmark$ is must-convergent, we show that $\tau(!\checkmark \mid ?\checkmark)$ can deadlock:

- since $W_1 = P_1P_1$, $\tau(!)$ must be of the form $\mathcal{R}_1P_1\{P_2, T_2\}^*P_1\mathcal{R}_2$
- since $W_2 = P_1$, $\tau(?)$ must be of the form $\{P_2, T_2\}^*P_1\mathcal{R}_3$ and $IS_1 = \blacksquare$
- on storage $(IS_1, IS_2) = (\blacksquare, IS_2)$ first execute $\mathcal{R}_1P_1\{P_2, T_2\}^*P_1\mathcal{R}_2$ until it blocks with remainder $P_1\mathcal{R}_2$. Then still $C_1 = \blacksquare$ holds.
- Now execute $\{P_2, T_2\}^*P_1\mathcal{R}_3$: It either blocks at some P_2 or at P_1 with remainder $P_1\mathcal{R}_3$.
- In all cases we have a deadlock. $\square$

Note: The proofs for some cases are more complex and require further case distinctions.

Conclusion & Future Work

Conclusion

- we proved that a correct compositional translation from SYNCSIMPLE into LOCKSIMPLE requires at least three locks (independently of the initial storage!)
- we showed that there is a correct translation with three locks

Future work

- correct translations with three locks for any initial storage values
- locks where take and put are blocking
- transfer of the result to full languages