Minimal Translations
from Synchronous Communication
to Synchronizing Locks

Manfred Schmidt-Schauß
Goethe-University Frankfurt

David Sabel
LMU Munich

EXPRESS/SOS 2021
August 23, 2021
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 $\pi$-calculus into Concurrent Haskell

\[\pi \xrightarrow{\tau} \text{CH}\]

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

shared memory concurrency with synchronising variables (MVars)
concurrent $\lambda$-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

**SYNCSIMPLE**

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

**LOCKSIMPLE**

- 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:
\[ U ::= \checkmark \mid 0 \mid !U \mid ?U \]
(succes) (silence) (send) (receive)

Processes:
\[ P ::= U \mid U \mid P \]
(subprocess) (parallel composition)

Operational semantics:
\[ !U_1 \mid ?U_2 \mid P \xrightarrow{SYS} U_1 \mid U_2 \mid P \]
Source Calculus: SYNCSIMPLE

Subprocesses:
\[ U ::= \checkmark \mid 0 \mid !U \mid ?U \]
(succes\(s\)) (silence) (send) (receive)

Processes:
\[ P ::= U \mid U \mid P \]
(subprocess) (parallel composition)

Operational semantics:
\[ !U_1 \mid ?U_2 \mid P \xrightarrow{SYS} U_1 \mid U_2 \mid P \]

Example:
\[ ?!0 \mid !!\checkmark \mid ?0 \]
\[ ?!0 \mid !\checkmark \mid 0 \]
\[ !0 \mid \checkmark \mid 0 \]
\[ !0 \mid !\checkmark \mid 0 \]
\[ 0 \mid \checkmark \mid 0 \]
\[ !0 \mid \checkmark \mid 0 \]
\[ 0 \mid !\checkmark \mid 0 \]
\[ 0 \mid !\checkmark \mid 0 \]
Source Calculus: SYNSIMPLE

Subprocesses:
\[ U ::= \checkmark | \emptyset | !U | ?U \]
  (success) (silence) (send) (receive)

Processes:
\[ P ::= U | U | P \]
  (subprocess) (parallel composition)

Operational semantics:
\[ !U_1 | ?U_2 | P \xrightarrow{SYS} U_1 | U_2 | P \]

Example:
\[ ?!0 | !!\checkmark | ?0 \xrightarrow{SYS} !0 | !\checkmark | ?0 \]
\[ !0 | !\checkmark | ?0 \xrightarrow{SYS} !0 | \checkmark | 0 \]
\[ ?!0 | !\checkmark | 0 \xrightarrow{SYS} !0 | \checkmark | 0 \]

- \( P \) is successful if \( P = \checkmark | P' \)
- \( P \) is may-convergent if there is some successful process \( P' \) with \( P \xrightarrow{SYS,*} P' \).
- \( P \) is must-convergent if for all \( P' \) with \( P \xrightarrow{SYS,*} P' \), the process \( P' \) is may-convergent.
Target Calculus: \( \text{LOCKSIMPLE}_{k, IS} \)

Subprocesses:
\[
\mathcal{U} ::= \, \checkmark \mid 0 \mid P_i U \mid T_i U
\]
(\text{success}) \hspace{1cm} \text{(silence)} \hspace{1cm} \text{(put on lock } i \text{)} \hspace{1cm} \text{(take on lock } i \text{)}

Processes:
\[
\mathcal{P} ::= \, \mathcal{U} \mid \mathcal{U} \mid \mathcal{P}
\]
\text{(subprocess)} \hspace{1cm} \text{(parallel composition)}

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

Operational semantics:
\[
(P_i U \mid \mathcal{P}, C[C_i = \square]) \xrightarrow{LS} (U \mid \mathcal{P}, C[C_i \mapsto \blacksquare])
\]
(put fills an empty lock / blocks on a filled)

\[
(T_i U \mid \mathcal{P}, C) \xrightarrow{LS} (U \mid \mathcal{P}, C[C_i \mapsto \square])
\]
(take empties the lock, non-blocking)
Target Calculus: \( \text{LOCKSIMPLE}_{k,IS} \)

Subprocesses:
\[
U ::= \checkmark \mid 0 \mid P_iU \mid T_iU
\]
- (success)
- (silence)
- (put on lock \( i \))
- (take on lock \( i \))

Processes:
\[
P ::= U \mid U \mid U \mid P
\]
- (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:
\[
(P_iU \mid P, C[C_i = \square]) \xrightarrow{LS} (U \mid P, C[C_i \mapsto \blacksquare])
\]
- (put fills an empty lock / blocks on a filled)

\[
(T_iU \mid P, C) \xrightarrow{LS} (U \mid P, C[C_i \mapsto \square])
\]
- (take empties the lock, non-blocking)

Example:
\[
(P_2P_1\checkmark \mid T_10 \mid T_20, (\blacksquare, \blacksquare)) \xrightarrow{LS} (P_2P_1\checkmark \mid T_10 \mid 0, (\blacksquare, \square)) \xrightarrow{LS} (P_1\checkmark \mid T_10 \mid 0, (\blacksquare, \blacksquare)) \xrightarrow{LS} (P_1\checkmark \mid 0 \mid 0, (\square, \blacksquare)) \xrightarrow{LS} (\checkmark \mid 0 \mid 0, (\blacksquare, \blacksquare))
\]

- success, may- and must-convergence: analogous, but starting with initial storage \( IS \)
Translations

Compositional translations $\tau$
- map $\tau(!)$ and $\tau(?)$ to sequences consisting of $P_i$- and $T_i$-operations
- for all other constructs: translation is the identity
  
  $$
  (\tau(0) = 0, \tau(\checkmark) = \checkmark, \tau(P_1 \parallel P_2) = \tau(P_1) \parallel \tau(P_2) \ldots)
  $$

Translation $\tau$ is correct iff for all 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
Theorem (correct translation with 3 locks)

For \( k = 3 \), the translation \( \tau \) with

\[
\tau(!) = P_1 T_3 P_2 T_1 \quad \text{and} \quad \tau(?) = P_3 T_2
\]

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:

\[
\tau(!) = P_2 P_1 T_3 P_1 T_1 T_2 \quad \text{and} \quad \tau(?) = P_3 T_1
\]

is correct for initial store \((\square, \square, \blacksquare)\).
Results: Minimality

**Theorem (1 lock is insufficient)**
There is no correct compositional translation \( \text{SYNCSIMPLE} \rightarrow \text{LOCKSIMPLE}_{1,IS} \).

**Main Theorem (2 locks are insufficient)**
There is no correct compositional translation \( \text{SYNCSIMPLE} \rightarrow \text{LOCKSIMPLE}_{2,IS} \).

Both theorems hold for any initial storage!
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 $\tau$ 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 $S$ is
- $P_i$ if $S = R_1 P_i R_2$, where $R_1$ does not contain $P_i$ or $T_i$ and a deadlock occurs after executing $R_1$ on the initial storage $IS$
- $P_i P_i$ iff $S = R_1 P_i R_2 P_i R_3$, where $R_2$ does not contain $P_i$ or $T_i$, and a deadlock occurs after executing $R_1 P_i R_2$ on the initial storage $IS$

Proof shows impossibility for the blocking types $(P_1 P_1, P_1 P_1)$, $(P_1 P_1, P_2 P_2)$, $(P_1 P_1, P_1)$, $(P_1 P_1, P_2)$, $(P_1, P_1)$, and $(P_1, P_2)$ (other cases are symmetric)
**Claim**

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

Proof: While $!✓ | ?✓$ is must-convergent, we show that $τ(!✓ | ?✓)$ can deadlock:

- since $W_1 = P_1P_1$, $τ(!)$ must be of the form $R_1P_1{P_2,T_2} * P_1R_2$
- since $W_2 = P_1$, $τ(?)$ must be of the form $\{P_2,T_2\} * P_1R_3$ and $IS_1 = □$
- on storage $(IS_1, IS_2) = (□, IS_2)$ first execute $R_1P_1{P_2,T_2} * P_1R_2$ until it blocks with remainder $P_1R_2$. Then still $C_1 = □$ holds.
- Now execute $\{P_2,T_2\} * P_1R_3$: It either blocks at some $P_2$ or at $P_1$ with remainder $P_1R_3$.
- In all cases we have a deadlock.

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
Thank You!
If a correct compositional translation $\psi$ exists, then also a correct translation $\tau$ exists: Apply $\psi$ to SYNCSIMPLE and verify that the image $\psi(\text{SYNCSIMPLE})$ is in LOCKSIMPLE.