The Reversible Temporal Process Language

in collaboration with Laura Bocchi, Ivan Lanese & Shoji Yuen
Debugging Timed Systems

- An important class of concurrent systems are embedded systems.
- Time is instrumental for the functioning of embedded systems where some events are triggered by system clock.
- Soft/Real time applications.
- What about debugging these systems?
  - Soft-real time systems written in Erlang uses intensively the `after` primitive.
  - Time constraints add hidden dependencies among actions.
- Reversible Debugging.
  - Reversible debuggers help programmers to quickly find the source of misbehaviours in concurrent programs.
  - Can be defined on top of a causal-consistent reversible semantics.
  - No approaches on causal-consistent reversibility and time.
Example

process_a() ->
  receive
    X -> handle_message()
  end.

process_b(Pid) -> Pid! msg end.

PidA = spawn(?MODULE,process_a,[]),
spawn(?MODULE,process_b,[PidA]).

- There is a clear dependency among A and B
- Pid! Msg < handle_message
Timed Example

process_a() ->
   receive
       X -> handle_message()
       after 200 -> handle_timeout()
   end.

process_b(Pid) ->
   timer:sleep(500),
   Pid! msg
end.

PidA = spawn(?MODULE,process_a,[]),
spawn(?MODULE,process_b,[PidA]).

- A and B are supposed to communicate
- B after 200 ms does something else
- A sends the message after 500 ms
- There is a dependency without actual synchronisation
- timer:sleep < handle_timeout
Our approach

• We start from a simple temporal process algebra: the Temporal Process Language (TPL) by Hennessy & Harrison

• TPL is a simple extension of CCS with a timeout operator and an idling one

• Well-established behavioural theory

• TPL is based on 3 design choices:
  • Time determinism
  • Patience
  • Maximal progress
TPL properties
Time determinism

Time determinism tells that time actions from one state can never reach distinct states.

A consequence of time determinism is that choices cannot be resolved by time actions.

\[ \sigma.P + a.Q \xrightarrow{\sigma} P + a.Q \]
TPL properties
Patience and maximal progress

- **Patience** ensures that a communicating process $\alpha.P$ can indefinitely delay communication $\alpha$ with $\sigma$ (time action) without changing state

$$\alpha.P \xrightarrow{\sigma} \alpha.P$$

- **Maximal progress** states that internal/synchronization actions $\tau$ cannot be delayed

- Patience allows for time actions when communication is not possible, and maximal progress disallow time actions when communication is possible
TPL overview

Process $[\text{pid}.P](Q)$ models a timeout, it can

- either immediately do an action $\text{pid}$ followed by $P$
  \[
  \text{pid}.0 \parallel [\text{pid}.P](Q) \rightarrow 0 \parallel P
  \]
- or in case of a delay, continue as $Q$
  \[
  \sigma.\text{pid}.0 \parallel [\text{pid}.P](Q) \rightarrow \text{pid}.0 \parallel Q
  \]
Running Example in TPL

\[ A(0) = Q \quad A(n+1) = [pid.P](A(n)) \quad (n \in \mathbb{N}) \]

\[ B(0) = \overline{pid} \quad B(n+1) = \sigma.B(n) \quad (n \in \mathbb{N}) \]

\[ [pid.P](A(200)) \parallel B(500) \]

```
process_a() ->
  receive
    X -> handle_message()
    after 200 -> handle_timeout()
  end.

process_b(Pid) ->
  timer:sleep(500),
  Pid! msg
end.

PidA = spawn(?MODULE,process_a,[]),
spawn(?MODULE,process_b,[PidA]).
```
The Reversible Temporal Process Language

\[ P = \pi.P \mid [P](Q) \mid P + Q \mid P \parallel Q \mid P \setminus a \mid A \mid 0 \quad (\pi = \alpha \mid \sigma) \]

\[ X = \pi[i].X \mid [X][\downarrow_i](Y) \mid [X][\leftarrow_i](Y) \mid X + Y \mid X \parallel Y \mid X \setminus a \mid P \]

We use as reversing technique the static approach of Ulidowski&Phillips (e.g., CCSK)
revTPL semantics

Prefixes

Passive time action, the key will be resolved during parallel composition
**revTPL semantics**

**Timeout**

\[
\text{STout} \quad \quad \frac{X \xrightarrow{\tau} \text{std}(X) \quad \text{std}(Y)}{[X](Y) \xrightarrow{\sigma[i]} [X][i](Y)}
\]

\[
\text{Tout} \quad \quad \frac{X \xrightarrow{\alpha[i]} X' \quad \text{std}(Y)}{[X](Y) \xrightarrow{\alpha[i]} [X'][i](Y)}
\]

**SWait**

\[
\text{SWait} \quad \quad \frac{Y \xrightarrow{\pi[u]} Y' \quad u \neq i}{[X][i](Y) \xrightarrow{\pi[u]} [X][i](Y')}
\]

**Wait**

\[
\text{Wait} \quad \quad \frac{X \xrightarrow{\pi[u]} X' \quad u \neq i}{[X][i](Y) \xrightarrow{\pi[u]} [X'][i](Y)}
\]

Arrows indicate transitions, and the symbols \(\tau\), \(\sigma[i]\), \(\alpha[i]\), and \(\pi[u]\) represent specific actions or processes in the context.
revTPL semantics

Parallel operator

\[
\begin{align*}
\text{SYNW} & \quad X \xrightarrow{\sigma[u]} X' \quad Y \xrightarrow{\sigma[v]} Y' \\
& \quad (X \parallel Y) \not\xrightarrow{\gamma} \\
& \quad X \parallel Y \xrightarrow{\sigma[w]} X' \parallel Y'
\end{align*}
\]

\[
\begin{align*}
\text{PAR} & \quad X \xrightarrow{\alpha[i]} X' \quad i \notin \text{keys}(Y) \\
& \quad X \parallel Y \xrightarrow{\alpha[i]} X' \parallel Y
\end{align*}
\]

\[
\begin{align*}
\text{SYN} & \quad X \xrightarrow{\alpha[i]} X' \quad Y \xrightarrow{\overline{\alpha}[i]} Y' \\
& \quad X \parallel Y \xrightarrow{\tau[i]} X' \parallel Y'
\end{align*}
\]
**revTPL semantics**

**Choice operator**

CHO\textsubscript{W1} $\frac{X_1 \xrightarrow{\sigma[u]} X'_1 \quad X_2 \xrightarrow{\sigma[v]} X'_2 \quad \delta(u, v) = w}{X_1 + X_2 \xrightarrow{\sigma[w]} X'_1 + X'_2}$ \hspace{1cm} nact($X_1 + X_2$)

CHO\textsubscript{W2} $\frac{X_1 \xrightarrow{\sigma[u]} X'_1 \quad nact(X_2) \land \neg nact(X_1)}{X_1 + X_2 \xrightarrow{\sigma[u]} X'_1 + X_2}$

CHO $\frac{X_1 \xrightarrow{\alpha[i]} X'_1 \quad nact(X_2)}{X_1 + X_2 \xrightarrow{\alpha[i]} X'_1 + X_2}$

**time determinism**
Properties of revTPL

Embedding

• revTPL is a reversible extension of TPL, but it is also a timed extension of reversible CCS (in this case CCSK)

• We can then define two forgetful maps: a time forgetting one and an history forgetting one
Properties of revTPL

Reversible semantics

Lemma 1 (Loop Lemma). If $X$ is a reachable process, then $X \xrightarrow{\pi[u]} X'$ if and only if $X' \xleftarrow{\pi[u]} X$.

Definition 13 (Conflict and independence). Given a reachable process $X$, two coinitital transitions $t : X \xrightarrow{\pi_1[i]}_n Y$ and $s : X \xrightarrow{\pi_2[j]}_n Z$ are conflicting, written $t \# s$, if and only if one of the following conditions holds:

1. $X \xrightarrow{\sigma[i]} Y$ and $X \xrightarrow{\alpha[j]} Z$; delay cannot be swapped with a communication
2. $X \xrightarrow{\pi_1[i]} Y$ and $X \xrightarrow{\pi_2[j]} Z$ with $j \leq Y i$; bk step cannot remove a cause of a fw one
3. $X = C[Y' + Z']$, $Y' \xrightarrow{\pi_1[i]} Y''$ and $Z' \xrightarrow{\pi_2[j]} Z''$. branches of a $+$ are in conflict
Conflicting Transitions

Examples

\[ \sigma[i] \quad [b.0](0) \quad b[j] \]

\[ [b.0][i](0) \quad [b[j].0][j](0) \]

\[ a.0 + b.0 \]

\[ a[i] \quad b[j] \]

\[ a[i].0 + b.0 \quad a.0 + b[j].0 \]
Properties of revTPL
Reversible semantics

Definition 8 (Causal Equivalence). Let $\simeq$ be the smallest equivalence on paths closed under composition and satisfying:

1. if $t : X \xrightarrow{\pi_1[u]} Y_1$ and $s : X \xrightarrow{\pi_2[v]} Y_2$ are independent, and $s' : Y_1 \xrightarrow{\pi_2[u]} Z$, $t' : Y_2 \xrightarrow{\pi_1[v]} Z$ then $ts' \simeq st'$;
2. $tt \simeq e$ and $tt \simeq e$

Definition 9 (Causal Consistency (CC)). If $\rho$ and $\omega$ are cofinal and cofinal paths then $\rho \preceq \omega$. 
Properties of revTPL
Reversible semantics

• Unfortunately for this kind of semantics CC does not hold
• If we take the trace $\rho : \alpha.P \xrightarrow{\sigma[\star]} \alpha.P$ and the empty trace starting in $\alpha.P$
  • they are coinitial and cofinal
  • but not causally equivalent
• We hence revise the semantics in order to drop $\sigma[\star]$ labels
  • A compositional semantics can be obtained by replacing premises $X \xrightarrow{\sigma[\star]} X$
    with a (decidable) predicate
• In the revised semantics CC holds
Conclusions

• We have studied the interplay between time and reversibility

• A reversible semantics for TPL cannot be easily derived by using standard techniques

  • Extra care has to be taken with time action and the + operator

• Double interpretation of our results:

  • Timed version of CCSK and reversible extension of TPL

  • We can derive a notion of causality for TPL and CCSK which is not available in literature
Future work

• A further improvement would be to trigger a rollback as reaction to a time out
• Study more expressive time operators
• Add asynchrony
• Study a behavioural theory for revTPL
• Apply the learned lesson on reversible time to Erlang construct for time