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.

\[ \sigma.P + a.Q \xrightarrow{\sigma} P + a.Q \]

A consequence of time determinism is that choices cannot be resolved by time actions.
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 $\lfloor pid.P \rfloor (Q)$ models a timeout, it can

- either immediately do an action $\text{pid}$ followed by $P$
  \[ \text{pid}.0 \parallel \lfloor pid.P \rfloor (Q) \xrightarrow{\tau} 0 \parallel P \]

- or in case of a delay, continue as $Q$
  \[ \sigma.\text{pid}.0 \parallel \lfloor pid.P \rfloor (Q) \xrightarrow{\sigma} \text{pid}.0 \parallel Q \]
Running Example in TPL

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

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

---

```plaintext
process_a() \to
    receive
        X \to \text{handle_message()}
        after 200 \to \text{handle_timeout()}
    end.

process_b(Pid) \to
    \text{timer:sleep(500)},
    Pid! msg
end.

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

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

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

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

Terms are not destroyed by reductions, but used as history information.
revTPL semantics

Prefixes

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

\[\text{PACT } \alpha.P \xrightarrow{\sigma[*]} \alpha.P\]

\[\text{RACT } \pi.P \xrightarrow{\pi[i]} \pi[i].P\]

\[\text{Act } \frac{X \xrightarrow{\pi'[u]} X'}{\pi[i].X \xrightarrow{\pi'[u]} \pi[i].X'}
\text{ for } u \neq i\]
**revTPL semantics**

**Timeout**

\[
\text{STOut} \quad X \xrightarrow{\not\tau} \quad \text{std}(X) \quad \text{std}(Y) \\
[X](Y) \xrightarrow{\sigma[i]} [X][i\rightarrow](Y)
\]

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

**maximal progress**

**SWAIT**

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

**WAIT**

\[
X \xrightarrow{\pi[u]} X' \quad u \neq i \\
[X][i\leftarrow](Y) \xrightarrow{\pi[u]} [X'][i\leftarrow](Y)
\]
revTPL semantics

Parallel operator

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

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

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

**Choice operator**

\[
\text{ChOW1} \quad \frac{X_1 \xrightarrow{\sigma[u]} X_1'} \quad X_2 \xrightarrow{\sigma[v]} X_2' \quad \delta(u, v) = w \quad \text{nact}(X_1 + X_2) \]

\[
X_1 + X_2 \xrightarrow{\sigma[w]} X_1' + X_2'
\]

\[
\text{ChOW2} \quad \frac{X_1 \xrightarrow{\sigma[u]} X_1'} \quad \text{nact}(X_2) \land \neg\text{nact}(X_1) \quad X_1 + X_2 \xrightarrow{\sigma[u]} X_1' + X_2
\]

\[
\text{CHO} \quad \frac{X_1 \xrightarrow{\alpha[i]} X_1'} \quad \text{nact}(X_2) \quad X_1 + X_2 \xrightarrow{\alpha[i]} X_1' + X_2
\]
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 $X \xrightarrow{\pi[u]} X' \iff X' \xleftarrow{\pi[u]} X$

Definition 13 (Conflict and independence). Given a reachable process $X$, two coinitial transitions $t : X \xrightarrow{\pi_1[i]} Y$ and $s : X \xrightarrow{\pi_2[j]} Z$ 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

\[
\begin{align*}
\sigma[i] & \quad [b.0](0) & b[j] & \quad [b[j].0][j](0) \\
& \quad [b.0][i](0) & \quad [b[j].0][j](0) \\
\end{align*}
\]

\[
\begin{align*}
a[i] & \quad a.0 + b.0 & b[j] & \quad a.0 + b[j].0 \\
& \quad a[i].0 + b.0 & \quad a.0 + b[j].0 \\
\end{align*}
\]
Properties of revTPL
Reversible semantics

Definition 8 (Causal Equivalence). Let \( \approx \) 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 \( s' : Y_1 \xrightarrow{\pi_2[v]} Z \), \( t' : Y_2 \xrightarrow{\pi_1[u]} Z \) then \( ts' \approx st' \);
2. \( tt \approx e \) and \( tt \approx e \)

Definition 9 (Causal Consistency (CC)). If \( \rho \) and \( \omega \) are coinital and cofinal paths then \( \rho \approx \omega \).
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 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
• Simplify the timed semantics by dealing with a timer process in parallel with the system (like OS)