The Reversible Temporal Process Language

in collaboration with Laura Bocchi, Ivan Lanese & Shoji Yuen

Claudio Antares Mezzina - DCORE Meeting Paris 31/5/22

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

$$\begin{tikzcd} & P \arrow[dl, "\sigma"'] \arrow[dr, "\sigma"] & \\ Q & = & R \end{tikzcd}$$

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 α with σ (time action) without changing state

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

- Maximal progress states that internal/syncrhonization actions τ 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 pid followed by P

$$\overline{pid}.0 \parallel \lfloor pid.P \rfloor(Q) \xrightarrow{\tau} 0 \parallel P$$

- or in case of a delay, continue as Q

$$\sigma.\overline{pid}.0 \parallel \lfloor pid.P \rfloor(Q) \xrightarrow{\sigma} \overline{pid}.0 \parallel Q$$

Running Example in TPL

$$A(0) = Q \quad A(n+1) = \lfloor pid.P \rfloor (A(n)) \quad (n \in \mathbb{N})$$

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

$$\lfloor pid.P \rfloor (A(200)) \parallel B(500)$$

```
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

$$P = \pi.P \mid \lfloor P \rfloor(Q) \mid P + Q \mid P \parallel Q \mid P \setminus a \mid A \mid \mathbf{0} \qquad (\pi = \alpha \mid \sigma)$$

$$X = \pi[i].X \mid \lfloor X \rfloor[\underrightarrow{i}](Y) \mid \lfloor X \rfloor[\underleftarrow{i}](Y) \mid X + Y \mid X \parallel Y \mid X \setminus a \mid P$$

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

$$\textsc{PAct}\ \alpha.P \xrightarrow{\sigma[\star]} \alpha.P \qquad \textsc{RAct}\ \pi.P \xrightarrow{\pi[i]} \pi[i].P$$

$$\textsc{Act}\ \frac{X \xrightarrow{\pi'[u]} X' \qquad u \neq i}{\pi[i].X \xrightarrow{\pi'[u]} \pi[i].X'}$$

revTPL semantics

Timeout

maximal progress

$$\textsc{STout}\ \frac{X \stackrel{\tau}{\not\rightarrow} \qquad \mathtt{std}(X) \qquad \mathtt{std}(Y)}{\lfloor X \rfloor(Y) \xrightarrow{\sigma[i]} \lfloor X \rfloor[\underrightarrow{i}](Y)}$$

$$\textsc{SWait}\ \frac{Y \xrightarrow{\pi[u]} Y' \qquad u \neq i}{\lfloor X \rfloor[\underrightarrow{i}](Y) \xrightarrow{\pi[u]} \lfloor X \rfloor[\underrightarrow{i}](Y')}$$

$$\textsc{Tout}\ \frac{X \xrightarrow{\alpha[i]} X' \qquad \mathtt{std}(Y)}{\lfloor X \rfloor(Y) \xrightarrow{\alpha[i]} \lfloor X' \rfloor[\underleftarrow{i}](Y)}$$

$$\textsc{Wait}\ \frac{X \xrightarrow{\pi[u]} X' \qquad u \neq i}{\lfloor X \rfloor[\underleftarrow{i}](Y) \xrightarrow{\pi[u]} \lfloor X' \rfloor[\underleftarrow{i}](Y)}$$

revTPL semantics

Parallel operator

maximal progress

Time action * is resolved

$$\textsc{SynW}\ \frac{X \xrightarrow{\sigma[u]} X' \qquad Y \xrightarrow{\sigma[v]} Y' \qquad (X \parallel Y) \not\xrightarrow{\tau} \qquad \delta(u,v) = w}{X \parallel Y \xrightarrow{\sigma[w]} X' \parallel Y'}$$

$$\textsc{Par}\ \frac{X \xrightarrow{\alpha[i]} X' \qquad i \notin \mathtt{keys}(Y)}{X \parallel Y \xrightarrow{\alpha[i]} X' \parallel Y}$$

$$\textsc{Syn}\ \frac{X \xrightarrow{\alpha[i]} X' \qquad Y \xrightarrow{\overline{\alpha}[i]} Y'}{X \parallel Y \xrightarrow{\tau[i]} X' \parallel Y'}$$

revTPL semantics

Choice operator

time determinism

$$\textsc{ChoW1}\ \frac{X_1 \xrightarrow{\sigma[u]} X_1' \qquad X_2 \xrightarrow{\sigma[v]} X_2' \qquad \delta(u,v) = w \qquad \mathtt{nact}(X_1 + X_2)}{X_1 + X_2 \xrightarrow{\sigma[w]} X_1' + X_2'}$$

$$\textsc{ChoW2}\ \frac{X_1 \xrightarrow{\sigma[u]} X_1' \qquad \mathtt{nact}(X_2) \wedge \neg\mathtt{nact}(X_1)}{X_1 + X_2 \xrightarrow{\sigma[u]} X_1' + X_2} \qquad \textsc{Cho}\ \frac{X_1 \xrightarrow{\alpha[i]} X_1' \qquad \mathtt{nact}(X_2)}{X_1 + X_2 \xrightarrow{\alpha[i]} X_1' + X_2}$$

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

time forgetting

history forgetting

$$\begin{tikzcd}
& \mathtt{revTPL} \arrow[dl, "\phi^{\mathtt{t}}"'] \arrow[dr, "\phi^{\mathtt{h}}"] & \\
CCSK \arrow[dr, "\phi^{\mathtt{h}}"] & & TPL \arrow[dl, "\phi^{\mathtt{t}}"'] \\
& CCS &
\end{tikzcd}$$

Properties of revTPL

Reversible semantics

Lemma 1 (Loop Lemma). *If X is a reachable process, then $X \xrightarrow{\pi[u]} X' \iff X' \xhookrightarrow{\pi[u]} X$*

Definition 13 (Conflict and independence). *Given a reachable process X, two coinititial transitions $t : X \xmapsto{\pi_1[i]}_n Y$ and $s : X \xmapsto{\pi_2[j]}_n Z$ 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* $X \xhookrightarrow{\pi_2[j]} Z$ *with* $j \leq_Y i$; ← bk step cannot remove a cause of a fw one
3. $X = \mathbf{C}[Y' + Z']$, $Y' \xrightarrow{\pi_1[i]} Y''$ *and* $Z' \xrightarrow{\pi_2[j]} Z''$. ← branches of a + are in conflict

Conflicting Transitions

Examples

$$\begin{tikzcd}[column sep=small]
 & \lfloor b.\mathbf{0} \rfloor(\mathbf{0}) \arrow[dl, "\sigma[i]"'] \arrow[dr, "b[j]"] & \\
\lfloor b.\mathbf{0} \rfloor[\underrightarrow{i}](\mathbf{0}) & & \lfloor b[j].\mathbf{0} \rfloor[\underleftarrow{j}](\mathbf{0})
\end{tikzcd}$$

$$\begin{tikzcd}[column sep=small]
 & a.\mathbf{0} + b.\mathbf{0} \arrow[dl, "a[i]"'] \arrow[dr, "b[j]"] & \\
a[i].\mathbf{0} + b.\mathbf{0} & & a.\mathbf{0} + b[j].\mathbf{0}
\end{tikzcd}$$

Properties of revTPL

Reversible semantics

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

1. *if* $t : X \xmapsto{\pi_1[u]} Y_1$ *and* $s : X \xmapsto{\pi_2[v]} Y_2$ *are independent, and* $s' : Y_1 \xmapsto{\pi_2[v]} Z$, $t' : Y_2 \xmapsto{\pi_1[u]} Z$ *then* $ts' \asymp st'$;
2. $t\underline{t} \asymp \epsilon$ *and* $\underline{t}t \asymp \epsilon$

Definition 9 (Causal Consistency (CC)). *If ρ and ω are coinitial and cofinal paths then $\rho \asymp \omega$.*

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