Fractional permissions and non-deterministic evaluators in interval temporal logic

Brijesh Dongol\textsuperscript{1}  John Derrick\textsuperscript{1}  Ian J. Hayes\textsuperscript{2}

\textsuperscript{1}Department of Computer Science,
The University of Sheffield

\textsuperscript{2}School of Information Technology and Electrical Engineering,
The University of Queensland

October 15, 2012
Background and goals

- We are after a method for reasoning about fine-grained atomicity
- Assume sequential consistency, but expression evaluation potentially takes several steps
  - Multiple processes may read the same variable at any time
  - A single process may read at most one variable atomically
  - A single process may write to at most one variable atomically
  - A write to a variable blocks all access to the variable
- There can be interference from other processes while an expression is being evaluated
- Expression evaluation is potentially non-deterministic
Background and goals

- We are after a method for reasoning about fine-grained atomicity.
- Assume *sequential consistency*, but expression evaluation potentially takes several steps.
  - Multiple processes may read the same variable at any time.
  - A single process may read at most one variable atomically.
  - A single process may write to at most one variable atomically.
  - A write to a variable blocks all access to the variable.
- There can be *interference* from other processes while an expression is being evaluated.
- Expression evaluation is potentially *non-deterministic*.
- **Goal:** Can we develop a high-level semantics that formalises the interference that may occur during expression evaluation?
Example

Consider the parallel execution:

<table>
<thead>
<tr>
<th>Init: ( u, v = 0, 0 )</th>
</tr>
</thead>
</table>
| \hline
| Process A | Process B |
|\hline
| \( A_1: \textbf{if } u < v \textbf{ then} \) | \( B_1: \ u := 1 \; ; \) |
| \( A_2: \quad \textit{Statement}_1 \) | \( B_2: \ v := 1 \) |
| \( A_3: \quad \textbf{else } \textit{Statement}_2 \) | |
| \( \textbf{fi} \) | |

- In a coarse-grained interpretation, process \( A \) will always execute \textit{Statement}_2 regardless of the state of process \( B \)
- The values of \((u, v)\) prior to guard evaluation at \( A_1 \) are \((0, 0)\), \((1, 0)\) and \((1, 1)\) and hence \( u < v \) always evaluates to false
Example

Consider the parallel execution:

\[
\begin{array}{|c|c|}
\hline
\text{Init: } & u, v = 0, 0 \\
\text{Process } A & \text{Process } B \\
A_1: & \textbf{if } u < v \textbf{ then} \\
A_2: & \textit{Statement}_1 \\
A_3: & \textbf{else } \textit{Statement}_2 \\
& \textbf{fi} \\
B_1: & u := 1 ; \\
B_2: & v := 1 \\
\hline
\end{array}
\]

In reality, expression evaluation takes time and there may be interference while an expression is evaluated.

Consider the following sequence after initialisation:

1. Process A reads \( u \) (i.e., \( u = 0 \))
2. Process B executes \( B_1 \) and \( B_2 \)
3. Process A reads \( v \) (i.e., \( v = 1 \))
4. \( u < v \) evaluates to true!
Lesson learnt: There is a distinction between actual vs. apparent states.

In previous example:

- Actual values of \((u, v)\) during evaluation at \(A_1\) are in the set
  
  \[
  \{(0, 0), (1, 0), (1, 1)\}
  \]

- States of \((u, v)\) apparent to process \(A\) during evaluation at \(A_1\)
  
  are in the set
  
  \[
  \{(0, 0), (1, 0), (1, 1), (0, 1)\}
  \]

Can we define semantics of commands to capture apparent states evaluation?
Actual vs. observable states

- **Lesson learnt:** There is a distinction between actual vs. apparent states
- In previous example:
  - Actual values of \((u, v)\) during evaluation at \(A_1\) are in the set 
    \[ \{(0, 0), (1, 0), (1, 1)\} \]
  - States of \((u, v)\) apparent to process \(A\) during evaluation at \(A_1\) are in the set 
    \[ \{(0, 0), (1, 0), (1, 1), (0, 1)\} \]
- Can we define semantics of commands to capture apparent states evaluation?
  - We know expression evaluation takes time (multiple steps)
  - We consider Moszkowski’s *Interval Temporal Logic*
Interval Temporal Logic: Basics

Note that our setup is different from Moszkowski et al.

\[
\begin{align*}
\text{Time} & \triangleq \mathbb{Z} \\
V & \subseteq \text{Var} \\
\text{State}_V & \triangleq V \rightarrow \text{Val} \\
\text{Stream}_V & \triangleq \text{Time} \rightarrow \text{State}_V \\
\text{Interval} & \triangleq \left\{ \Delta \subseteq \text{Time} \mid \forall t_1, t_2 \in \Delta, t \in \text{Time} \cdot t_1 \leq t \leq t_2 \Rightarrow t \in \Delta \right\} \\
\text{StatePred}_V & \triangleq \text{State}_V \rightarrow \mathbb{B} \\
\text{IntvPred}_V & \triangleq \text{Interval} \rightarrow \text{Stream}_V \rightarrow \mathbb{B}
\end{align*}
\]
Interval predicate operators

- Use ‘.’ for function application
- Pointwise lift boolean operators to interval predicates, e.g.,

\[
\begin{align*}
false.\Delta.s & \triangleq false \\
(p_1 \lor p_2).\Delta.s & \triangleq p_1.\Delta.s \lor p_2.\Delta.s
\end{align*}
\]

- Also have additional operators
  - ‘;’ (to model sequential composition)
  - ‘ω’ (to model both finite and infinite iteration)
Chop

$(p_1 ; p_2).\Delta.s$ iff either

- $\Delta$ can be split into adjoining intervals $\Delta_1$ and $\Delta_2$ and $p_1.\Delta_1.s$ and $p_2.\Delta_2.s$, or
- $\text{lub}.\Delta = \infty$ and $p_1.\Delta.s$
Chop

\((p_1 ; p_2).\Delta.s\) iff either

- \(\Delta\) can be split into adjoining intervals \(\Delta_1\) and \(\Delta_2\) and \(p_1.\Delta_1.s\) and \(p_2.\Delta_2.s\), or
- \(lub.\Delta = \infty\) and \(p_1.\Delta.s\)
Define interval predicate \( \text{Empty}\.\Delta.s \triangleq (\Delta = \{\}) \)

Order interval predicates using ‘\( \Rightarrow \)’

\[
p_1 \Rightarrow p_2 \triangleq \forall \Delta, s \cdot p_1.\Delta.s \Rightarrow p_2.\Delta.s
\]

Iteration of \( p \) is defined as:

\[
p^\omega \triangleq \nu z \cdot (p ; z) \lor \text{Empty}
\]
Commands

Definition
For a state predicate $b$, variable $v$ and expression $e$, the abstract syntax of commands is given by $Cmd$ below, where $C, C_1, C_2 \in Cmd$.

$$Cmd ::= False \mid True \mid Idle \mid [b] \mid v := e \mid C_1 ; C_2 \mid C_1 \sqcap C_2 \mid C^\omega \mid C_1 \parallel C_2$$
Interval-based behaviour

- Define behaviour of a command as an interval predicate
- Use \( \text{beh}_X \in \text{Cmd} \rightarrow \text{IntvPred} \) where \( X \) is the set of processes executing the command
- Some behaviours:

\[
\begin{align*}
\text{beh}_X.\text{False} & \triangleq \text{false} \\
\text{beh}_X.\text{True} & \triangleq \text{true} \\
\text{beh}_X.(C_1 ; C_2) & \triangleq \text{beh}_X.C_1 ; \text{beh}_X.C_2 \\
\text{beh}_X.(C_1 \cap C_2) & \triangleq \text{beh}_X.C_1 \lor \text{beh}_X.C_2 \\
\text{beh}_X.C^\omega & \triangleq (\text{beh}_X.C)^\omega \\
\text{beh}_X.(C_1 \parallel C_2) & \triangleq \exists X_1, X_2 \cdot (X_1 \cup X_2 = X) \land (X_1 \cap X_2 = \emptyset) \land \text{beh}_{X_1}.(C_1 ; \text{Idle}) \land \text{beh}_{X_2}.(C_2 ; \text{Idle})
\end{align*}
\]
Semantics of guard evaluation and assignment

- What about semantics of Idle, guard evaluation \([b]\) and assignment \(\nu := e\)?

- To model fine-grained atomicity, we must evaluate expressions using states apparent to a process (as opposed to actual states)

- For our example, semantics must reflect that \(u < \nu\) possibly evaluates to \(true\) even though \(\neg(u < \nu)\) in all actual states
Semantics of guard evaluation and assignment

- What about semantics of Idle, guard evaluation \([b]\) and assignment \(v := e\)?

- To model fine-grained atomicity, we must evaluate expressions using states apparent to a process (as opposed to actual states)

- For our example, semantics must reflect that \(u < v\) possibly evaluates to \(true\) even though \(\neg(u < v)\) in all actual states

- **But first:** Need a method for reasoning about read/write access to shared variables
Fractional permissions

Solution: Boyland’s fractional permissions

\[ \Pi_{x.v.t} \] denotes the permission for process \( x \) to access variable \( v \) at time \( t \)

\[ \Pi_{x.v.t} \] is a rational number such that \( 0 \leq \Pi_{x.v.t} \leq 1 \)

Define:

\[ \mathcal{D}_{x.v.t} \equiv (\Pi_{x.v.t} = 0) \] to mean \( x \) is denied permission to access \( v \) at time \( t \)

\[ \mathcal{R}_{x.v.t} \equiv (0 < \Pi_{x.v.t} < 1) \] to mean \( x \) has read permission to \( v \) at time \( t \)

\[ \mathcal{W}_{x.v.t} \equiv (\Pi_{x.v.t} = 1) \] to mean \( x \) has write permission to \( v \) at time \( t \)
Healthiness conditions

- We have a number of healthiness conditions on permissions.
- E.g., the sum of the permissions of the processes for any variable $v$ in any state of stream $s$ is at most 1, i.e.,

$$\forall v \in \text{Var}, t \in \text{Time} \cdot (\sum_{x \in \text{Proc}} \Pi_x.v.t) \leq 1$$

- These can become part of the rely condition of a program.
- Hence,
  - at most one process can have write permission at any time.
  - read permission at any time can be distributed arbitrarily.
Actual states over an interval

▶ Straightforward to reason about states that actually occur over an interval
▶ Recall

\[ \text{StatePred} \triangleq \text{State} \to \mathbb{B} \]
\[ \text{IntvPred} \triangleq \text{Interval} \to \text{Stream} \to \mathbb{B} \]

▶ Define \( \text{states.} \Delta . s \triangleq \{ \sigma : \sum | \exists t : \Delta \cdot \sigma = s . t \} \)
▶ For a state predicate \( c \), interval \( \Delta \) and stream \( s \), we define:

\[
(\square c).\Delta . s \triangleq \forall \sigma : \text{states.} \Delta . s \cdot c.\sigma \\
(\diamond c).\Delta . s \triangleq \exists \sigma : \text{states.} \Delta . s \cdot c.\sigma
\]
States apparent to a process

To reason about the states apparent to a process $x$, we define function $\text{apparent}_{x,W}$ where $W$ is a set of variables

$$\sigma \in \text{State}_W \mid \forall v \in W \bullet \exists t \in \Delta \bullet (\sigma.v = s.t.v) \land R_{x.v.t}$$

We then define:

$$\forall \sigma : \text{apparent}_{x,\text{vars}}, c.\Delta.\hat{s} = \exists \sigma : \text{apparent}_{x,\text{vars}}, c.\Delta.\hat{s}$$
States apparent to a process

- To reason about the states apparent to a process $x$, we define function $\text{apparent}_{x,W}$ where $W$ is a set of variables.
- $W$ corresponds to the set of variables whose values $x$ needs to determine to fully evaluate an expression.
States apparent to a process

- To reason about the states apparent to a process $x$, we define function $apparent_{x,W}$ where $W$ is a set of variables.
- $W$ corresponds to the set of variables whose values $x$ needs to determine to fully evaluate an expression.

$$apparent_{x,W}.\Delta.s \triangleq \begin{cases} 
\sigma \in State_{W} | \\
\forall v \in W \cdot \exists t \in \Delta \cdot (\sigma.v = s.t.v) \land R_{x}.v.t \end{cases}$$
States apparent to a process

- To reason about the states apparent to a process \( x \), we define function \( \text{apparent}_{x,W} \) where \( W \) is a set of variables.
- \( W \) corresponds to the set of variables whose values \( x \) needs to determine to fully evaluate an expression.

\[
\begin{align*}
\text{apparent}_{x,W}.\Delta.s & \equiv \left\{ \sigma \in \text{State}_W \mid \forall v \in W \cdot \exists t \in \Delta \cdot (\sigma.v = s.t.v) \land R_x.v.t \right\} \\
\end{align*}
\]

- We then define
\[
\begin{align*}
(\Box_x c).\Delta.s & \equiv \forall \sigma: \text{apparent}_{x,\text{vars}.c}.\Delta.s \cdot c.\sigma \\
(\Diamond_x c).\Delta.s & \equiv \exists \sigma: \text{apparent}_{x,\text{vars}.c}.\Delta.s \cdot c.\sigma
\end{align*}
\]
Behaviour of idle, guard evaluation and assignment

\[ beh_{\{x\}}.\text{Idle} \triangleq (\forall v \cdot \Box \neg W_{x}.v) \]
Behaviour of idle, guard evaluation and assignment

\[ beh\{x\}.\text{Idle} \equiv (\forall v \cdot \Box \neg W_x.v) \]

\[ beh\{x\}.[b] \equiv (\Diamond_x b) \land beh\{x\}.\text{Idle} \]
Behaviour of idle, guard evaluation and assignment

\[ beh_{\{x\}}.\text{Idle} \equiv (\forall v \cdot \Box \lnot \mathcal{W}_x.v) \]

\[ beh_{\{x\}}.[b] \equiv (\#_x b) \land beh_{\{x\}}.\text{Idle} \]

\[ beh_{\{x\}}.(v := e) \equiv \exists k: Val \cdot beh_{\{x\}}.[e = k] ; \\
\left( \Box (v = k \land \mathcal{W}_x.v) \land \\
(\forall u: \text{Var} \setminus \{v\} \cdot \Box \lnot \mathcal{W}_x.u) \land \\
\lnot \text{Empty} \right) \]
Example: Apparent states evaluation

\[
\begin{array}{|c|c|}
\hline
\text{Init: } u, v = 0, 0 \\
\hline
\text{Process } A & \text{Process } B \\
\hline
A_1: \textbf{if } u < v \textbf{ then} & B_1: u := 1 \\
A_2: \text{Statement}_1 & B_2: v := 1 \\
A_3: \textbf{else } \text{Statement}_2 & \\
A_4: \textbf{fi} & \\
\hline
\end{array}
\]

We can prove both of the following properties

\[
\square (pc_A = A_1) \implies \square (u \geq v)
\]

\[
\square (pc_A = A_1) \land \Diamond R_A.u \land \Diamond R_A.v \implies \Box_A (u < v) \land \Box_A (u \geq v)
\]
Compositional reasoning

- Use rely/guarantee-style reasoning where *rely* specifies behaviour of the environment (formalised as an interval predicate)

- $\text{beh}_X. (\text{RELY } r \cdot C).\Delta \equiv r.\Delta \Rightarrow (\text{beh}_X. C).\Delta$

- Unlike Jones,
  - the rely condition holds in the *same interval* as the command
  - *fractional permissions* can be used to ensure that a program and its environment do not simultaneously modify the same variable
Refinement

Definition
We say $A$ is refined by $C$ with respect to set of processes $X$ (denoted $A \subseteq_X C$) iff

$$beh_X.C \Rightarrow beh_X.A$$
Decomposition theorems

Definition
$$(\square p).\Delta.s \equiv \forall\Delta' \in \text{Interval} \cdot \Delta' \subseteq \Delta \Rightarrow p.\Delta'.s$$

Definition
$r$ splits iff $r \Rightarrow \square r$
Decomposition theorems

Definition
\[(\square p).\Delta.s \equiv \forall \Delta' \in Interval \cdot \Delta' \subseteq \Delta \Rightarrow p.\Delta'.s\]

Definition
r splits iff \(r \Rightarrow \square r\)

Theorem
If r splits, then both of the following hold.

\[\begin{align*}
(\text{Rely } r \cdot C_1 ; C_2) & \sqsubseteq_X (\text{Rely } r \cdot C_1) ; (\text{Rely } r \cdot C_2) \quad (1) \\
\text{Rely } r \cdot C^\omega & \sqsubseteq_X (\text{Rely } r \cdot C)^\omega \quad (2)
\end{align*}\]
Decomposition theorems

Theorem
If there exist $X_1, X_2 \subseteq X$ where $X = X_1 \cup X_2$ and $X_1 \cap X_2 = \emptyset$ and rely conditions $r_1$ and $r_2$, such that both of the following hold

$$\left( \text{RELY } r \land r_1 \cdot A_1 \right) \sqsubseteq_{X_1} C_1 \land \left( \text{RELY } r \land r_2 \cdot A_2 \right) \sqsubseteq_{X_2} C_2$$

$$\left( r \land \text{beh}_{X_2}.C_2 \Rightarrow r_1 \right) \land \left( r \land \text{beh}_{X_1}.C_1 \Rightarrow r_2 \right)$$

then

$$\left( \text{RELY } r \cdot A_1 \parallel A_2 \right) \sqsubseteq_X C_1 \parallel C_2$$
\[(\text{RELY } r \land r_1 \cdot A_1) \sqsubseteq_{X_1} C_1 \quad \land \quad (\text{RELY } r \land r_2 \cdot A_2) \sqsubseteq_{X_2} C_2\]  
\[(r \land \text{beh}_{X_2}.C_2 \Rightarrow r_1) \quad \land \quad (r \land \text{beh}_{X_1}.C_1 \Rightarrow r_2)\]

then

\[(\text{RELY } r \cdot A_1 \parallel A_2) \sqsubseteq_X C_1 \parallel C_2\]

Proof.

\[(3) = \text{definition of } \sqsubseteq_X, \text{logic } (a \Rightarrow (b \Rightarrow c)) = (a \land b \Rightarrow c)\]
\[(r \land r_1 \land \text{beh}_{X_1}.C_1 \Rightarrow \text{beh}_{X_1}.A_1) \land (r \land r_2 \land \text{beh}_{X_2}.C_2 \Rightarrow \text{beh}_{X_2}.A_2)\]
\[\Rightarrow \quad \text{logic}\]
\[r \land r_1 \land \text{beh}_{X_1}.C_1 \land r_2 \land \text{beh}_{X_2}.C_2 \Rightarrow \text{beh}_{X_1}.A_1 \land \text{beh}_{X_2}.A_2\]
\[\Rightarrow \quad (4)\]
\[r \land \text{beh}_{X_1}.C_1 \land \text{beh}_{X_2}.C_2 \Rightarrow \text{beh}_{X_1}.A_1 \land \text{beh}_{X_2}.A_2\]

Hence we have:

\[\exists X_1, X_2 \cdot (X_1 \cup X_2 = X) \land (X_1 \cap X_2 = \emptyset) \land (3)\]
\[\Rightarrow \quad \text{calculation above and logic and definitions}\]
\[(\text{RELY } r \cdot A_1 \parallel A_2) \sqsubseteq_X C_1 \parallel C_2\]
Conclusions and future work

- Used interval-based reasoning and fractional permissions to formalise semantics that reflect fine-grained atomicity
- Interference is possible during interval of expression evaluation
- Evaluation is non-deterministic when multiple variables are accessed
- Evaluation takes place in states apparent to a process
- Can define rely/guarantee-style rules to allow compositional reasoning
Conclusions and future work

- Used interval-based reasoning and fractional permissions to formalise semantics that reflect fine-grained atomicity
- Interference is possible during interval of expression evaluation
- Evaluation is non-deterministic when multiple variables are accessed
- Evaluation takes place in states apparent to a process
- Can define rely/guarantee-style rules to allow compositional reasoning

- We have encoded part of this theory in Isabelle/HOL
- Applying this theory to prove linearisability
- Can we define high-level command semantics for weaker memory models?
- Generalise interval-based rely/guarantee theory
Questions?