Temporal Specification

[Specification Esoterica]

Michael Fisher

Department of Computer Science, University of Liverpool, UK

[MFisher@liverpool.ac.uk]

An Introduction to Practical Formal Methods Using Temporal Logic
Shared Variables (1)

Recall

1. Both elements must see *exactly* the same values for the shared variable.
2. Both elements must see updates to the variable at the same time.
3. Only one element is allowed to write to the shared variable at any one step.
A solution (to 1 and 2)

and ensure $\square(x \Leftrightarrow y)$. 
Solving point 3 is a little more difficult.

One common approach:

Add a \textit{writing\_to}(C, x) predicate into the specification of element \( C \) such that we make this true whenever the shared variable \( x \) is written to by \( C \).

Effectively, this means

\[
[x := 2] = \Diamond ((x = 2) \land \text{writing\_to}(C, x))
\]
Then, if \( x \) in element \( C \) and \( y \) in element \( D \) are to refer to the same shared variable, the specification becomes

\[
Spec_C \land Spec_D \\
\land (x \iff y) \\
\land \neg (\text{writing}_\text{to}(C, x) \land \text{writing}_\text{to}(D, y))
\]
**Shared Variables Example**

\[\text{Spec}_C: \begin{align*}
\text{start} \Rightarrow & \quad x = 0 \\
\wedge & \quad \exists v. \ x = v \land \text{odd}(x) \Rightarrow \ \lozenge(x = (v + 1) \land \\
& \quad \text{writing}_\text{to}(C, x))
\end{align*}\]

\[\text{Spec}_D: \begin{align*}
\exists w. \ y = w \land \text{even}(y) \Rightarrow \ & \quad \lozenge(y = (w + 1) \land \\
& \quad \text{writing}_\text{to}(D, y))
\end{align*}\]

\[\text{Spec}_C \land \text{Spec}_D \land \begin{align*}
\square(x \leftrightarrow y) \\
\lozenge \neg(\text{writing}_\text{to}(C, x) \land \text{writing}_\text{to}(D, y))
\end{align*}\]
Communication Properties

The structure of the *Comms* formula defines the intended communication properties:

- $\square(a \Rightarrow x)$ ........................................... instantaneous
- $\square(a \Rightarrow \bigcirc x)$ ....................................... one step delay
- $\square(a \Rightarrow \Diamond x)$ ................................. guaranteed delivery
- $\square(a \Rightarrow \Diamond (x \lor lost))$  ...................... potential message loss
Some things we missed out
Interleaving Concurrency

Only one of the elements can execute at any one time, and so periods of activity for one element are interspersed (or interleaved) with periods of activity for the other.

For example

$$Spec_{C_1} \land Spec_{C_2}$$

$$\land \square \neg (executing(C_1) \land executing(C_2))$$
Asynchronous Execution

Elements can all have different clocks with one element potentially working *faster* than another.

\[
\begin{align*}
E_1 & \quad \rightarrow \quad \rightarrow \quad \rightarrow \quad \rightarrow \\
E_2 & \quad \rightarrow \quad \rightarrow \quad \rightarrow \quad \rightarrow \\
\end{align*}
\]
Frame Properties

Think of semantics of $x := 0; y := 3$, i.e.

$$[x := 0; y := 3] = \circ((x = 0) \land \circ(y = 3))$$

But, what is the value of $x$ when $y = 3$?

For example, what is

$$[x := 0; y := 3; x := x + 1]$$

Semantics of assignment is more complex than

$$[x := v; S] = \exists w. x = w \land \circ(x = v(x/w) \land [S])$$

We must also say that values of variables other than $x$ remain the same in the next moment.
Verification

Overall, temporal logic is a good formalism for specifying communicating and interacting systems.

However, we have problems proving whether temporal formulae are true or false.

The decide whether a temporal formula is true involves theorem-proving, which is expensive especially within first-order temporal logics.

However, we can combine use of temporal logic as a behavioural description, together with a more efficient form of verification, giving model checking