Temporal Specification

[Specification Esoterica]

Michael Fisher

Department of Computer Science, University of Liverpool, 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)

\[ (x \equiv y) \]

and ensure \( (x \equiv y) \).

Shared Variables (2)

Solving point 3 is a little more difficult.

One common approach:

Add a \( writing\_to(c, x) \) 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

\[ [x := 2] = \square ((x = 2) \land writing\_to(c, x)) \]
Shared Variables (3)

Then, if \( x \) in element \( C \) and \( y \) in element \( D \) are to refer to the same shared variable, the specification becomes

\[
Spec_C \land Spec_D \\
\land [x \iff y] \\
\land \neg (writing_to(C, x) \land writing_to(D, y))
\]

Communication Properties

The structure of the \( Comms \) formula defines the intended communication properties:

- \( \square (a \Rightarrow x) \) .... instantaneous
- \( \square (a \Rightarrow \Diamond x) \) .... one step delay
- \( \square (a \Rightarrow \Box x) \) .... guaranteed delivery
- \( \square (a \Rightarrow \Diamond (x \lor lost)) \) .... potential message loss

Shared Variables Example

\[
Spec_C: \square [start \Rightarrow x = 0 \\
\land \exists v. x = v \land odd(x) \Rightarrow \Box (x = (v + 1) \land writing_to(C, x))]
\]

\[
Spec_D: \square [\exists w. y = w \land even(y) \Rightarrow \Box (y = (w + 1) \land writing_to(D, y))]
\]

Some things we missed out

\[
Spec_C \land Spec_D \\
\land [x \iff y] \\
\land \neg (writing_to(C, x) \land writing_to(D, y))
\]
**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

\[ Spec_{C_1} \land Spec_{C_2} \land \neg (executing(C_1) \land executing(C_2)) \]

**Asynchronous Execution**

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

**Frame Properties**

Think of semantics of \( x := 0; y := 3 \), i.e.

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

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

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

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*