3C03 Concurrency: Safety

Mark Handley

Goals

- Define the concept of safety.
- Explicit and implicit definition of safety properties.
- Modelling:
  - How safety properties can be specified in FSP.
  - Safety analysis using LTSA.
  - Proof that our approach to locking achieves mutual exclusion.
Safety Properties

- Safety properties assert that nothing ‘bad’ will ever happen during the execution of a concurrent program.

- Examples of safety properties:
  - Mutual Exclusion
  - Deadlock Freedom
  - Monitor Invariants

- We are interested in:
  - Whether our FSP models satisfy safety properties?
  - How we transform safe models into safe implementations?

Safety in FSP: Property

Safety property definition is supported by FSP
- A safety property is itself a process.
- It contains no hidden actions.
- It is denoted using keyword property.

A safety property specifies acceptable behaviour for the process it is composed with.
Safety in FSP: Property Satisfaction

- A system $S$ will satisfy a property $P$ if $S$ can only generate sequences of actions which, when restricted to the alphabet of $P$, are acceptable to $P$.

- Example:
  
  property POLITE=(knock->enter->POLITE).
  HESITANT = (knock->knock->enter->HESITANT).
  IMPATIENT = (enter->IMPATIENT).
  ||CHK_HES = (HESITANT || POLITE).
  ||CHK_IMP = (IMPATIENT || POLITE).

Properties in LTS

- LTS generated for properties have
  - an additional error state (-1)
  - transitions leading to the error state for actions that would violate the property

- Example:

```
-1
  enter

0
  enter

1
  knock
```

© Wolfgang Emmerich, Mark Handley 1998 - 2003
Exercise

- **Draw the LTS for**
  - property FRIEND=(come->tea->leave->FRIEND).

Safety Analysis using LTSA

- **We automate safety analysis using the Labelled Transition System Analyser**

- **LTSA can:**
  - compute the LTS for a safety property.
  - compose the property with the process to be checked.
  - If there is a trace from the initial state to the error state the system is unsafe.
ERROR states

*Processes can be implicit properties if they use the state ERROR*

- ERROR is a special state (like STOP).

The perspective is different:

- Properties specify desirable behaviour.
- Processes which use the ERROR state specify undesirable behaviour.

*Example: mutual exclusion*

---

Ornamental Garden Revisited

const N = 2
range T = 0..N

VAR = VAR[0],

TURNSTILE = (arrive -> INCREMENT),
INCREMENT = (value.read[x:T] -> value.write[x+1] -> TURNSTILE + {value.read[T], value.write[T]}).

||GARDEN = (east:TURNSTILE || west:TURNSTILE || {east, west}::value:VAR).
Ornamental Garden Revisited

\[
\text{const } N = 2 \\
\text{range } T = 0..N \\
VAR = \text{VAR}[0], \\
\text{VAR}[u:T] = (\text{read}[u] \rightarrow \text{VAR}[u] \\
\hspace{1cm} | \text{write}[v:T] \rightarrow \text{VAR}[v]).
\]

\[
\text{TURNSTILE} = (\text{arrive} \rightarrow \text{INCREMENT} \\
\hspace{1cm} | \text{suspend} \rightarrow \text{resume} \rightarrow \text{TURNSTILE}), \\
\text{INCREMENT} = (\text{value.read}[x:T] \\
\hspace{1cm} \rightarrow \text{value.write}[x+1] \rightarrow \text{TURNSTILE} \\
\hspace{2cm} ) + \{\text{value.read}[T], \text{value.write}[T]\}. \\
\]

\[
\text{GARDEN} = (\text{east:TURNSTILE} || \text{west:TURNSTILE} \\
\hspace{1cm} | | \{\text{east,west,display}:\text{value}:\text{VAR} \\
\hspace{2cm} )/\{\text{stop/east.suspend, stop/west.suspend,} \\
\hspace{3cm} \text{start/east.resume, start/west.resume}\}.
\]

Mutual Exclusion as Safety Property

\[
\text{TEST} = \text{TEST}[0], \\
\text{TEST}[v:T] = \\
\hspace{1cm} (\text{when } (v<N) \\
\hspace{2cm} \{\text{east.arrive,west.arrive} \rightarrow \text{TEST}[v+1] \\
\hspace{3cm} | \text{stop} \rightarrow \text{CHECK}[v]\}, \\
\text{CHECK}[v:T] = (\text{display.value.read}[u:T] \rightarrow \\
\hspace{2cm} (\text{when } (u==v) \text{ start} \rightarrow \text{TEST}[v] \\
\hspace{3cm} | \text{when } (u!=v) \text{ wrong} \rightarrow \text{ERROR}) \\
\hspace{3cm} ) + \{\text{display.value.read}[T], \\
\hspace{4cm} \text{display.value.write}[T]\}). \\
\]

\[
\text{TESTGARDEN} = (\text{GARDEN} || \text{TEST}).
\]
FSP Model for Locking

VAR = VAR[0],
VAR[u:T]=({ read[u] -> VAR[u]
          | write[v:T] -> VAR[v] }).

LOCK = (acquire -> release -> LOCK).
||LOCKVAR = (LOCK || VAR).

TURNSTILE = (arrive -> INCREMENT),
INCREMENT = (value.acquire
            -> value.read[x:T] -> value.write[x+1]
            -> value.release -> TURNSTILE
          )+{value.read[T],value.write[T]}.

||GARDEN = (east:TURNSTILE || west:TURNSTILE
           || {east,west}::value:LOCKVAR).
Safety Properties for Locking

\[
\begin{align*}
\text{TEST} & = \text{TEST}[0], \\
\text{TEST}[v:T] & = (\text{when } (v<N)
\{\text{east.arrive, west.arrive} \rightarrow \text{TEST}[v+1] \\
\quad \mid \text{stop } \rightarrow \text{CHECK}[v]), \\
\text{CHECK}[v:T] & = (\text{display.value.read}[u:T] \rightarrow \\
\quad (\text{when } (u==v) \rightarrow \text{TEST}[v]
\quad \mid \text{when } (u!=v) \rightarrow \text{ERROR}) \\
\quad ) + \{\text{display.value.read}[T], \\
\quad \text{display.value.write}[T], \\
\quad \text{display.value.acquire}, \\
\quad \text{display.value.release}\}.
\end{align*}
\]

\[||\text{TESTGARDEN} = (\text{GARDEN } || \text{ TEST})\]

Summary

- Introduced the concept of Safety
- Specification of Safety Properties in FSP
- Checking of Safety Properties using LTSA
- Proof of Mutual Exclusion based on Locking