3C03 Concurrency: Liveness & Progress

Mark Handley

Outline

- Liveness
- Progress
- Progress Specification in FSP
- Progress-Analysis of LTS
- Priorities
A small country road goes over a wooden bridge that connects two shores of a river. The bridge is rather narrow and cars can only go north-bound or south-bound. Given that it is constructed out of wood, the bridge can only carry three cars at a time.

```
const CAPACITY=3
range T=0..CAPACITY
FLOW_OF_CARS=FLOW_OF_CARS[0],
FLOW_OF_CARS[i:T]=
  ( when (i<CAPACITY) enter -> FLOW_OF_CARS[i+1]
    | when (i>0) leave -> FLOW_OF_CARS[i-1]
    | going[i] -> FLOW_OF_CARS[i]
  ).
BRIDGE_CONTROLLER={
  south.going[s:T] -> north.going[n:T] ->
  ( when (n==0) south.enter -> BRIDGE_CONTROLLER
    | when (s==0) north.enter -> BRIDGE_CONTROLLER
    | south.leave -> BRIDGE_CONTROLLER
    | north.leave -> BRIDGE_CONTROLLER).
||BRIDGE = (north:FLOW_OF_CARS || south:FLOW_OF_CARS ||
          BRIDGE_CONTROLLER ).
```
Motivation

Problem with single lane bridge:

- Cars cannot pass from north to south if there is a continuous stream of cars from south to north!
- We would like to guarantee that cars will eventually cross the bridge.

In more general terms this is referred to as liveness.

Liveness

A liveness property asserts that something good eventually happens.

- We want to specify liveness for our FSP models
- We want to analyze our FSP models to be certain that the liveness properties hold

General form of liveness requires consideration of temporal precedence relationship between states.
- We use more restricted form of progress.
Progress

A progress property asserts that whatever state a system is in, it is always the case that a specified action will eventually be executed.

- Progress is the opposite of starvation.
- Notion of progress is sufficiently powerful to capture wide range of liveness properties.
- Progress properties are simple to specify in FSP.

Progress Properties in FSP

- Specification of progress needs assumption of a fair scheduling policy.

- If a transition from a set is chosen infinitely often and every transition in the set will be executed infinitely often, the scheduling policy is said to be fair.

- \( \text{progress } P=\{a_1, a_2, \ldots, a_n\} \) defines a progress property \( P \) which asserts that in an infinite execution at least one of the actions \( a_1, a_2, \ldots, a_n \) will be executed infinitely often.
Example: Tossing Coins

\[ \text{COIN} = (\text{toss} \rightarrow \text{heads} \rightarrow \text{COIN} \mid \text{toss} \rightarrow \text{tails} \rightarrow \text{COIN}) \].

progress HEADS = \{\text{heads}\}
progress TAILS = \{\text{tails}\}

Example: Tossing Trick Coins

\[ \text{TWOCOIN} = (\text{pick} \rightarrow \text{COIN} \mid \text{pick} \rightarrow \text{TRICK}), \]
\[ \text{COIN} = (\text{toss} \rightarrow \text{heads} \rightarrow \text{COIN} \mid \text{toss} \rightarrow \text{tails} \rightarrow \text{COIN}), \]
\[ \text{TRICK} = (\text{toss} \rightarrow \text{heads} \rightarrow \text{TRICK}). \]

progress HEADS = \{\text{heads}\}
progress TAILS = \{\text{tails}\}
Progress Analysis

We can automate analysis of progress properties

- A set of states, where every state is reachable from every other state in the set, and no state has transitions to states outside the set, is a terminal set of states.

Terminal set of states can be found using a graph algorithm that searches for a strongly connected component.

Default Progress Properties

Default progress properties assert in a system with fair choices that every action in the alphabet will be executed infinitely often.

Default progress properties of example:

- progress P1 = \{pick\}
- progress P2 = \{toss\}
- progress P3 = \{heads\}
- progress P4 = \{tail\}

How many violations?
Priorities

Default progress analysis of single lane bridge does not reveal violation.

- Problem is scheduling policy.
- Northbound cars arriving get ‘priority’ if there are already northbound cars on the bridge.

To detect such progress violations we have to reflect such priorities in the FSP model.

High Priority in FSP

- $||C = (P | Q) << \{a_1, \ldots, a_n\}$ specifies a composition in which the actions $a_1, \ldots, a_n$ have higher priority than any other action in the alphabet of $P | Q$ including the silent action $\tau$.

- In any choice in this system which has one or more of the actions $a_1, \ldots, a_n$ labelling a transition, the transitions labelled with lower priority actions are discarded.
Low Priority in FSP

- \((\mathcal{P} \mid \mathcal{Q}) \gg \{a_1, \ldots, a_n\}\) specifies a composition in which the actions \(a_1, \ldots, a_n\) have lower priority than any other action in the alphabet of \(\mathcal{P} \mid \mathcal{Q}\) including the silent action \(\tau\).

- In any choice in this system which has one or more transitions not labelled by \(a_1, \ldots, a_n\), the transitions labelled by \(a_1, \ldots, a_n\) are discarded.

Simplification of LTS

Priorities simplify the LTS resulting of the composition.

Example:

\[
\text{NORMAL} = (\text{work} \rightarrow \text{play} \rightarrow \text{NORMAL} \\
| \text{sleep} \rightarrow \text{play} \rightarrow \text{NORMAL}).
\]

\[
\mid \mid \text{HIGH} = (\text{NORMAL}) \ll \{\text{work}\}.
\]

\[
\mid \mid \text{LOW} = (\text{NORMAL}) \gg \{\text{work}\}.
\]

- Use of priorities lead to more realistic liveness checks.
Progress in Single-lane Bridge

```plaintext
const CAPACITY=3
range T=0..CAPACITY
FLOW_OF_CARS=FLOW_OF_CARS[0],
FLOW_OF_CARS[i:T]=
  { when (i<CAPACITY) enter -> FLOW_OF_CARS[i+1]
  | when (i>0) leave -> FLOW_OF_CARS[i-1]
  | going[i] -> FLOW_OF_CARS[i]
  }.
BRIDGE_CONTROLLER=
  south.going[s:T] -> north.going[n:T] ->
  ( when (n==0) south.enter -> BRIDGE_CONTROLLER
  | when (s==0) north.enter -> BRIDGE_CONTROLLER
  | south.leave -> BRIDGE_CONTROLLER
  | north.leave -> BRIDGE_CONTROLLER)
  ||BRIDGE = (north:FLOW_OF_CARS || south:FLOW_OF_CARS ||
  BRIDGE_CONTROLLER )>>(north.leave, south.leave).
progress NORTHBOUND = {north.enter}
progress SOUTHBOUND = {south.enter}
```

Summary

- **Liveness**
- **Progress**
- **Progress Specification in FSP**
- **Progress-Analysis of LTS**
- **Priorities**