3C03 Concurrency: Liveness & Progress

Mark Handley

Outline

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

Single-lane Bridge

- 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.

Single-lane Bridge

```
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 <u>progress property</u> asserts that whatever state a system is in, it is always the case that a specified action will eventually be executed

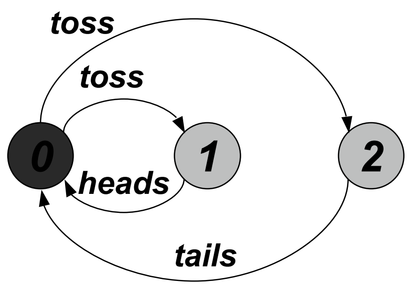
- *Progress is the opposite of starvation*
- *Notion of progress is sufficiently powerful to capture wide range of liveness properties*
- *Progress properties are simple to specify in FSP*

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 <u>fair.</u>*
- `progress P={a1,a2,…,an}` *defines a progress property* `P` *which asserts that in an infinite execution at least one of the actions* `a1,a2,…,an` *will be executed infinitely often.*

Example: Tossing Coins

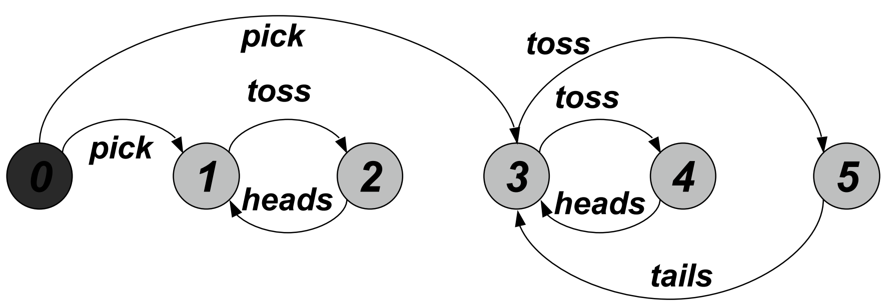
```
COIN = ( toss -> heads -> COIN
       | toss -> tails -> COIN).
```




```
progress HEADS = {heads}
progress TAILS = {tails}
```


Example: Tossing Trick Coins

```
TWOCOIN = (pick->COIN | pick->TRICK),
COIN    = ( toss -> heads -> COIN
          | toss -> tails -> COIN),
TRICK   = (toss->heads->TRICK).
progress HEADS = {heads}
progress TAILS = {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 <u>terminal set of states</u>.*

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

LTSA

Default Progress Properties

<u>Default progress properties</u> 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?

LTSA

Priorities

Default progress analysis of single lane bridge <u>does not</u> 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.

LTSA

High Priority in FSP

- `||C = (P||Q)<<{a1,…,an}` *specifies a composition in which the actions* `a1,…,an` *have <u>higher priority</u> 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* `a1,…,an` *labelling a transition, the transitions labelled with lower priority actions are <u>discarded</u>.*

Low Priority in FSP

- `||C = (P||Q)>>{a1,…,an}` *specifies a composition in which the actions* `a1,…,an` *have lower 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 transitions not labelled by* `a1,…,an`*, the transitions labelled by* `a1,…,an` *are discarded.*

Simplification of LTS

Priorities simplify the LTS resulting of the composition.

Example:

```
NORMAL=(work->play->NORMAL
       |sleep->play->NORMAL).
||HIGH=(NORMAL)<<{work}.
||LOW= (NORMAL)>>{work}.
```

- *Use of priorities lead to more realistic liveness checks.*

LTSA

Progress in Single-lane Bridge

```
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***