VIS Technology Transfer Course

Session 7

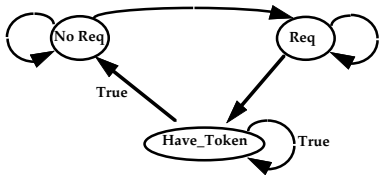
Fairness Constraints and Monitors

Serdar Tasiran

Why fairness constraints?

- **Non-determinism is used to hide irrelevant detail.**
- **May introduce behavior that is not part of actual system.**

Example: Client module



- **Non-det. model allows**
  No Req, Req, Req, Have_Token, Have_Token ...
  but actual client eventually gives up token.
- **Want to refine the non-det. description to model the client more closely.**
- **Must remove traces that stay in Have_Token forever.**

Fairness Constraints and Monitors
Serdar Tasiran, VIS - TTC May 15, 1996

Infinite Traces and Fairness Constraints

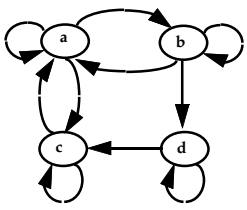
- **A trace is a sequence of states.**
- **Infinite traces are useful for reasoning about eventualities and arbitration.**

Examples:

- **"If Have_Token then eventually No Req."**
  . . ., Have_Token, Have_Token, NoReq, NoReq, . . . ✓
  . . ., Have_Token, Have_Token, Have_Token, . . . ✗
- **An environment model that randomly generates input patterns A, B, and C.**
  "It is always the case that sometime in the future an A is generated. " Similarly for B and C.
  A, B, C, A, B, C, A, B, C, ...
  A, B, C, B, A, B, C, B, A, ... both satisfy constraints.
  A, B, C, A, B, A, B, A, B, ... does not.

Infinity Sets

- **Number of states is finite, so an infinite trace visits some states infinitely often.**
- **These states are called the infinity set of a trace:**
  1. **a, b, a, b, a, b, ...** → **{a, b}**
  2. **a, b, a, c, c, c, ...** → **{c}**
  3. **a, b, a, c, a, c, a, c ...** → **{a, c}**



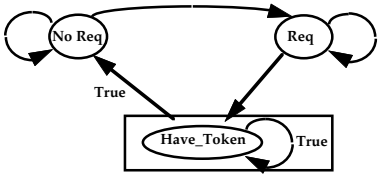
- **Fairness constraints are conditions on the infinity set.**
- **F^{inf}{a, b} : "Infinitely often {a,b}" means some state in {a, b} must be in the infinity set.**

  Satisfied by 1 and 3, but not 2.

Fairness Constraints and Monitors
Serdar Tasiran, VIS - TTC May 15, 1996

Fairness Constraints in VIS

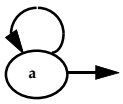
- **VIS allows fairness constraints of the form**
  $F^{inf} S_1 \ \& \ F^{inf} S_2 \ \& \ ... \ \& \ F^{inf} S_n$
- **S_i: A set of states represented by a CTL formula.**
- **The fairness constraint is satisfied if some state in each S_i is visited infinitely often.**
- **Example: Let S_1 be given by !(client_state = Have_Token)**
  S_1 = {NoReq, Req}



- **$F^{inf} S_1$ means {NoReq, Req} is visited infinitely often, i.e., Have_Token does not remain asserted forever.**
- **Usage: vis> read_fairness file.fair**
- **Restricts attention to traces that satisfy the fairness constraints.**
- **Like listing a set of false paths.**

Common Uses of Fairness Constraints

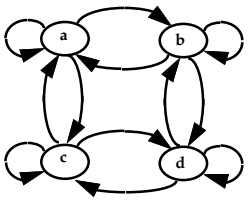
- **Modeling an unknown amount of wait:**
  " Don't stay in "a" forever. "



  F^{inf} **!(state_var = a)**

- **"It is always the case that sometime in the future an "a" is generated. " Similarly for "b", "c" and "d".**

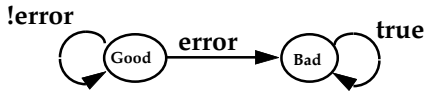
  $F^{inf}\{a\} \ \& \ F^{inf}\{b\} \ \& \ F^{inf}\{c\} \ \& \ F^{inf}\{d\}$



Bad Behaviors and Monitors

- **Want to see if a design can have bad behavior.**
- **Example of undesired behavior:**
  "Clients B and C are in the BUSY state at the same time."
  Mutual exclusion is violated.
- **A monitor is a module that watches for bad behavior.**
  Example: Mutual exclusion.

assign error = (B.state == BUSY) && (C.state == BUSY)

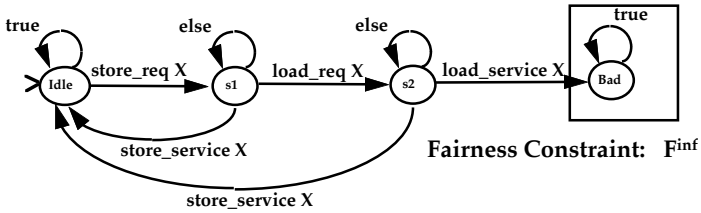


Fairness Constraint:
F^{inf} **{Bad}**

- **Monitor can satisfy its fairness constraint** **iff** **System can generate undesired behavior**

Why write monitors?

- **Some properties are hard or impossible to express in CTL.**
  Example: Store -Load order. Let X be a memory location.
  "If load_req X follows store_req X ,
  then the store must be serviced before the load."
- **Undesired behavior:**
  store_req X . . . load_req X . . . load_service X . . .



Fairness Constraint: F^{inf} **(Bad)**

- **Note: In the initial state, we non-deterministically start checking for the bad sequence.**

→ **Undesired behavior can be detected anywhere along the trace.**

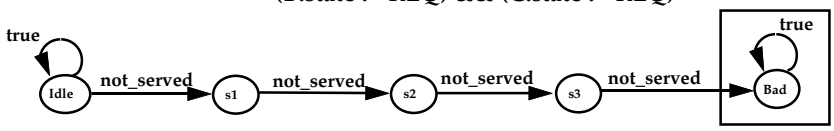
Monitor Example

Property: If A requests resource, and resource is free, A gets resource within 3 clock cycles unless another client requests resource in the meantime.

Bad Behavior: For more than 3 clock cycles,
A is in state REQ and ackA is false,
B and C are in state NO REQ.

Monitor:

```
assign not_served = (A.state = REQ) && !(ackA) &&
                    (B.state != REQ) && (C.state != REQ)
```



Fairness Constraint: F^{inf} **(Bad)**

Using monitors in VIS

- **Property holds iff the system + monitor combination has no fair traces.**
- **Procedure:**
  - **Write verilog code for the monitor, append to the system description.**
  - **Write CTL file specifying fairness constraints for system + monitor.**
    Fairness constraints of monitor should allow bad behavior only.
  - **Use the "lang_empty" command.**
  - **If there exists bad behavior, VIS will give a short error trace.**