Safety and Liveness Properties

SFWRENG 3BB4:

Software Design III — Concurrent System Design

FSP Specifications

Definition

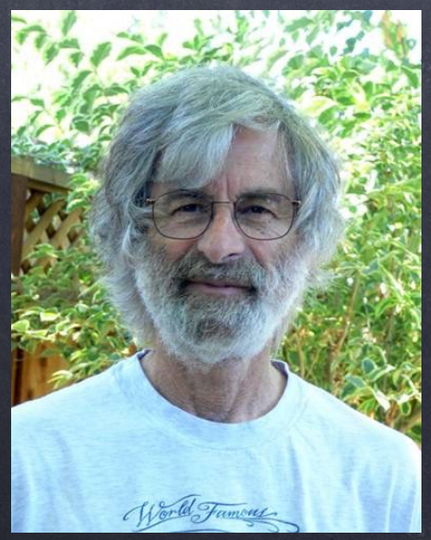
- Specification is the process of describing a system and is desired properties. This means that a specification is a description of a system together with some properties that such a system is meant to satisfy.
- The kinds of system properties may include:
    1. Behaviour
    2. Timing
    3. Performance
    4. Structure
    5. Etc.
- A specification is formal if it uses a language with a mathematically defined syntax and semantics.
- The language of FSP is a formal specification language. More so, it is a formal specification language that is particularly well-suited for specifying action based concurrent systems.

Safety and Liveness

Definition

- The language of FSP is a formal specification language.
- In FSP systems correspond to processes. What about properties?
- A property is an assertion about every possible trace of a designated FSP.
- The kinds of properties that we will concern ourself with in this course fall into two categories:

1. Safety Properties: Nothing bad happens.
2. Liveness Properties: Something good eventually happens.



Leslie B. Lamport

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-3, NO. 2, MARCH 1977 125

Proving the Correctness of Multiprocess Programs

LESLIE LAMPORT

Abstract—The inductive assertion method is generalized to permit formal, machine-verifiable proofs of correctness for multiprocess programs. Individual processes are represented by ordinary flowcharts, and no special synchronization mechanisms are assumed, so the method can be applied to a large class of multiprocess programs. A correctness proof can be designed together with the program by a hierarchical process of stepwise refinement, making the method practical for larger programs. The resulting proofs tend to be natural formalizations of the informal proofs that are now used.

Index Terms—Assertions, concurrent programming, correctness, multiprocessing, synchronization.

no particular synchronization primitive is assumed. Any desired primitive can easily be represented. The method is practical for larger programs because the proof can be designed together with the program by a hierarchical process of stepwise refinement.

To prove the correctness of a program, one must prove two essentially different types of properties about it, which we call *safety* and *liveness* properties.[1] A safety property is one which states that something will *not* happen. For example, the partial correctness of a single process program is a safety

Safety and Liveness

Definition

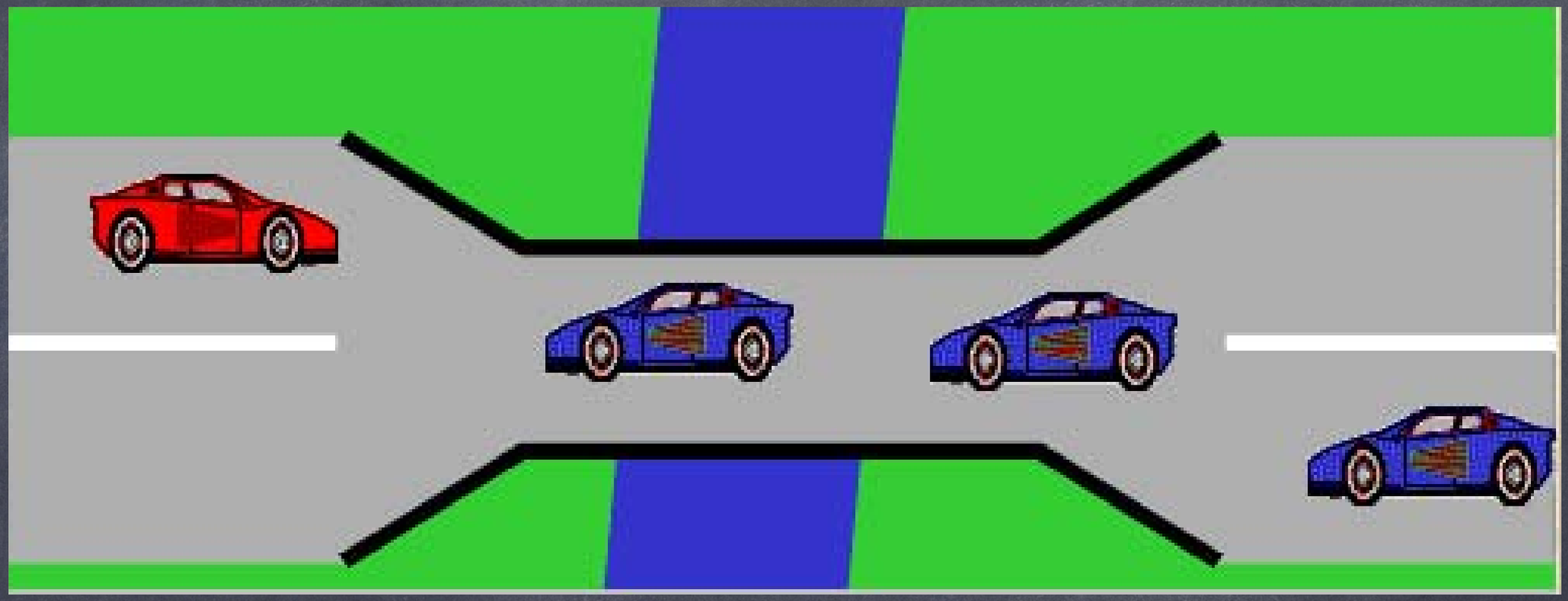
- A safety property is an assertion about every possible trace of a designated FSP stating that nothing bad happens.
- A safety property `P` is defined via the keyword `property` as a deterministic process that asserts that any trace including actions in the alphabet of `P`, is accepted by `P`.

Explanation

- If a safety property `P` is composed with a process `S`, then traces of actions in the alphabet of `S` and the alphabet of `P` must also be valid traces of `P`, otherwise a violation occurs.
- Transparency of safety properties: (i) Composing a safety property with a set of processes does not affect their correct behaviour; and (ii) Safety properties must be deterministic.

Safety and Liveness

Example The Single Lane Bridge Problem



A bridge over a river is only wide enough to permit a single lane of traffic. Consequently, cars can only move concurrently if they are moving in the same direction. A safety violation occurs if two cars moving in different directions enter the bridge at the same time.

Safety and Liveness

Definition

- A property is an assertion about every possible trace of a designated FSP.
- A safety property asserts that nothing bad happens.
- A liveness property asserts that something good eventually happens.
- A liveness property `P` is defined via the keyword **`progress`**.

Explanation

- A progress property asserts that it is always the case that an action is eventually executed.
- Progress is the opposite of starvation (the name given to a concurrent programming situation in which an action is never executed).

Progress vs. Fair Choice

- Progress properties assume fair choice of transitions.
- Fair Choice: If a choice over a set of transitions is executed infinitely often, then every transition in the set will be executed infinitely often.
- Progress properties are properties. Fair choice is an assumption over an FSP.

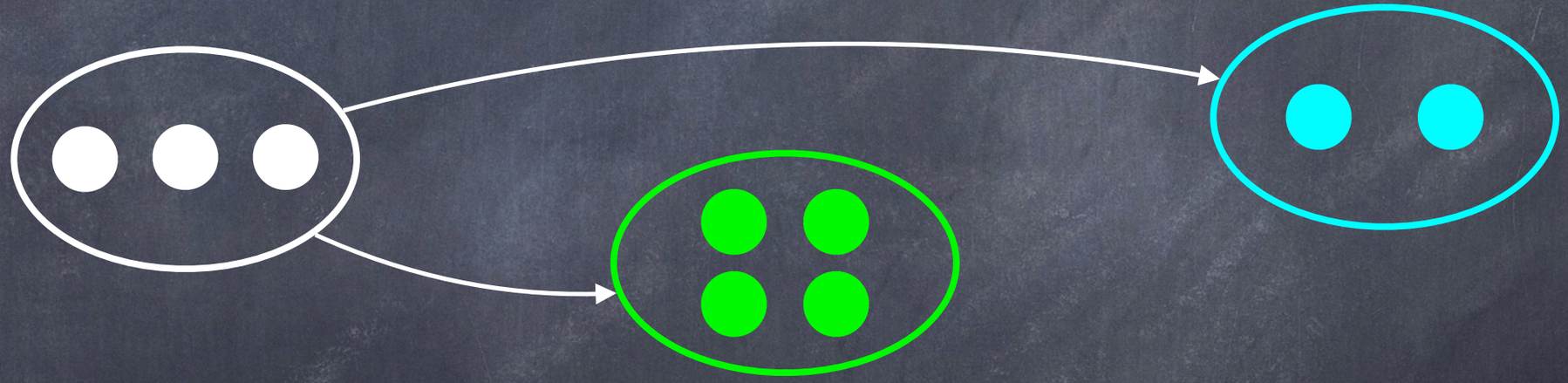
Safety and Liveness

Progress Analysis

- `progress P = {a1, a2, …, an}` defines a progress property `P` which asserts that in an infinite trace of a target FSP, at least one of the actions a_i will be executed infinitely often.

Checking Progress

- A terminal set of states is one in which: (i) Every state is reachable from every other state; and (ii) There is no transition from within the set to any state outside the set.



- Given fair choice, each terminal set represents a set of infinite traces in which each action appearing a trace in the set is executed infinitely often.
- Since there is no transition out of a terminal set, any action that is not used in the set cannot occur infinitely often in all executions of the system; and hence represents a potential progress violation.

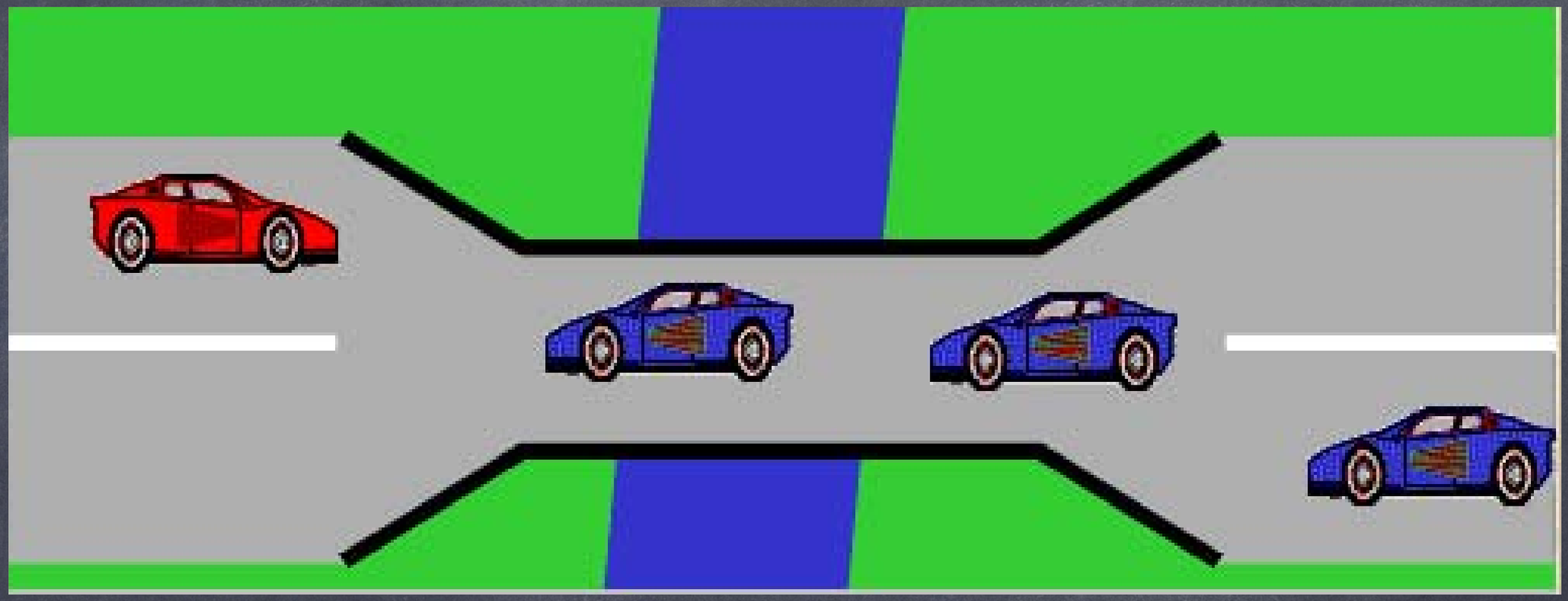
Liveness

Example The Three Shells and a Pea Game

In the game, three or more identical containers are placed face-down on a surface. A small ball is placed beneath one of these containers so that it cannot be seen. The containers are then shuffled by the operator in plain view. The players are invited to bet on which container holds the ball. The players win if they guess correctly, otherwise they lose.

Safety and Liveness

Example The Single Lane Bridge Problem



A bridge over a river is only wide enough to permit a single lane of traffic. Consequently, cars can only move concurrently if they are moving in the same direction. A safety violation occurs if two cars moving in different directions enter the bridge at the same time.

Safety and Liveness Properties

Reading Material

Chapter 7 of
Magee and Kramer Concurrency: State, Models, and Java Programming.