12. Fairness Assumptions

**Implicit Assumptions of Liveness**
- Every state will be entered eventually.
- The user will request to print infinitely often.
- Does this correctly represent the modelling assumptions?

**Communication Protocols**

**Liveness and Composition**

**The Alternating Bit Protocol**
### Sender and Receiver in SMV

**Module: send (ack)**
- `VAR` produce: boolean;
- `VAR` sending: boolean;
- `VAR` msg: boolean;
- `ASSIGN` produce := case
  - sending: 0, 1;
  - esac;
- `INIT` (sending) := 0;
- `NEXT` (sending) := case
  - produce: produc;
  - esac;
- `INIT` (msg) := 1;
- `NEXT` (msg) := case
  - produce: msg;
  - esac;

**Module: receiver (msg)**
- `VAR` consume: boolean;
- `VAR` waiting: boolean;
- `ASSIGN` consume := case
  - (waiting: 0, 1);
  - esac;
- `INIT` (waiting) := 0;
- `NEXT` (waiting) := case
  - waiting: consume;
  - ack := msg: 0;
- `INIT` (ack) := 1;
- `NEXT` (ack) := case
  - waiting: msg;
  - esac;

### Justice

**Definition:**
A transition is called **just** if it cannot happen that the transition is continuously enabled without being taken eventually.

### A Property ...

“Every message sent by the sender eventually arrives at the receiver.”

- `AG` (sender.msg=0 ⇒ `AF` msg.output=0)
- `AG` (sender.msg=1 ⇒ `AF` msg.output=1)

**Cannot be verified:**
Channels may lose messages indefinitely...

### Justice in NuSMV

**SPEC**
- `AG` (sender.msg=0 ⇒ `AF` msg.output=0)
- `LTL SPEC` `G` `¬loss` ⇒ `G` (sender.msg=0 ⇒ `F` msg.output=0))

**Consider only paths where ¬loss holds infinitely often.**

**Equivalent PLTL formula**

### Assuming Fairness in the Model

**Module: channel (input)**
- `VAR` output: boolean;
- `VAR` loss: boolean;
- `ASSIGN` `INIT` (output) := 1;
- `NEXT` (output) := case
  - loss: output;
  - esac;
- `JUSTICE` `¬loss`

**Fairness assumption:**
The channel does not lose all messages.

### Another Property ...

“Every message produced by the sender is eventually consumed by the receiver.”

- `AG` (sender.produce ⇒ `AF` receiver.consume)
Compassion

**Definition:**
A transition is called **compassionate** if it cannot happen that the transition is enabled infinitely often without being taken eventually.

Another Example ...

```plaintext
MODULE Car(lights)
VAR
  car: {coming, crossing, waiting, gone};
ASSIGN
  init(car) := gone;
  next(car) := case
    car=gone: {gone, coming};
    car=coming & lights=red: waiting;
    car=coming & lights=yellow: {waiting, crossing};
    car=coming & lights=green: crossing;
    car=waiting & lights=red: waiting;
    car=waiting & lights=yellow: waiting;
    car=waiting & lights=green: {waiting, crossing};
    car=crossing: gone;
  esac;
SPEC
  AG AF car=gone
```

Compassion in NuSMV

```plaintext
COMPASSION
  (ϕ, ψ)

Consider only paths that satisfy the following condition.

“If ϕ is true infinitely often, then ψ also must be true infinitely often.”
```

Compassion in NuSMV

```plaintext
LTLSPEC
  G F car=gone

LTLSPEC
  (G F (car=waiting & lights=green) -> G F car=crossing) -> G F car = gone)
```

Equivalent version without compassion

Reading

Bérard et. al.:
Chapter 7 – Safety Properties
Chapter 8 – Liveness Properties
Chapter 10 – Fairness Properties

Last Words

*Do not blindly trust a model checker that prints true.*

- Check properties that you expect to fail.
- Check the preconditions of implications.
- Check your assumptions.