Course 02158

Properties of Concurrent Programs

Hans Henrik Løvengreen

DTU Compute

Properties of Concurrent Programs

- Concurrent programs are (usually) *reactive*
- Properties must deal with *behaviour*, not results
- Two kinds of *functional properties*:
  - **Safety properties** Program does nothing wrong
  - **Liveness properties** Program does something (good)
- Real-time and performance requirements may be added

Formal treatment

- Let $\alpha = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \cdots$ be a (finite or infinite) execution
- Let $\phi[\alpha]$ denote that property ϕ holds for execution α
- A property ϕ holds for a program P iff

$$\forall \alpha \in \mathit{Exec}(P) : \phi[\alpha]$$

where $\mathit{Exec}(P)$ is the set of all executions of P

Safety Properties

- A safety property ensures that the program does nothing wrong

Examples

- *At most one process may use the printer at a time*
- *The variable x never decreases*
- *The motor turns off only if the key has been removed*

Formal treatment

- Property ϕ is satisfied for execution α: $\phi[\alpha]$
- ϕ is a *safety property* iff

$$\forall \alpha : \neg\phi[\alpha] \Rightarrow \exists \beta \prec \alpha : \forall \gamma : \neg\phi[\beta\gamma]$$

- Can be stated by *invariants* (and history variables)
- Can be shown by *model-checking* or *inductive proofs*

Liveness Properties

- A liveness property ensures that the program makes progress

Examples

- *The program will return to input mode again and again*
- *The variable x will never become constant*
- *The green light is lit when the Go button is pressed*

Formal treatment

- ϕ is a (pure) *liveness property* iff

$$\forall \beta : \mathsf{fin}(\beta) \Rightarrow \exists \gamma : \phi[\beta\gamma]$$

- Can be stated using *Temporal Logic*
- Can be shown by *temporal reasoning*, e.g. proof lattices

Race Conditions

- A *race condition* is the possibility of an **unintended** result or effect
- Caused by a particular execution

Example (Java)

```
class job extends Thread {

private boolean finish;

public void run() {
  finish = false;

  while (finish==false) {
    ⋮
  }

public void endThread() { finish = true; }
⋮
```

Lack-of-progress Properties

Deadlock

- *Deadlock* = cycle of processes waiting for each other (for ever)
- Typical cause: incremental reservation of shared resources

Starvation

- A process suffers from *starvation* if it could make progress, but never does so
- Typical causes: Unfair scheduling, priorities, bad luck

Livelock

- *Livelock* = mutual starvation (*after-you-after-you*)
- Like deadlock, but can be *escaped*
- Typical cause: Symmetrical strategies
- **Note:** Andrews sets *livelock* = *deadlock* — we don't