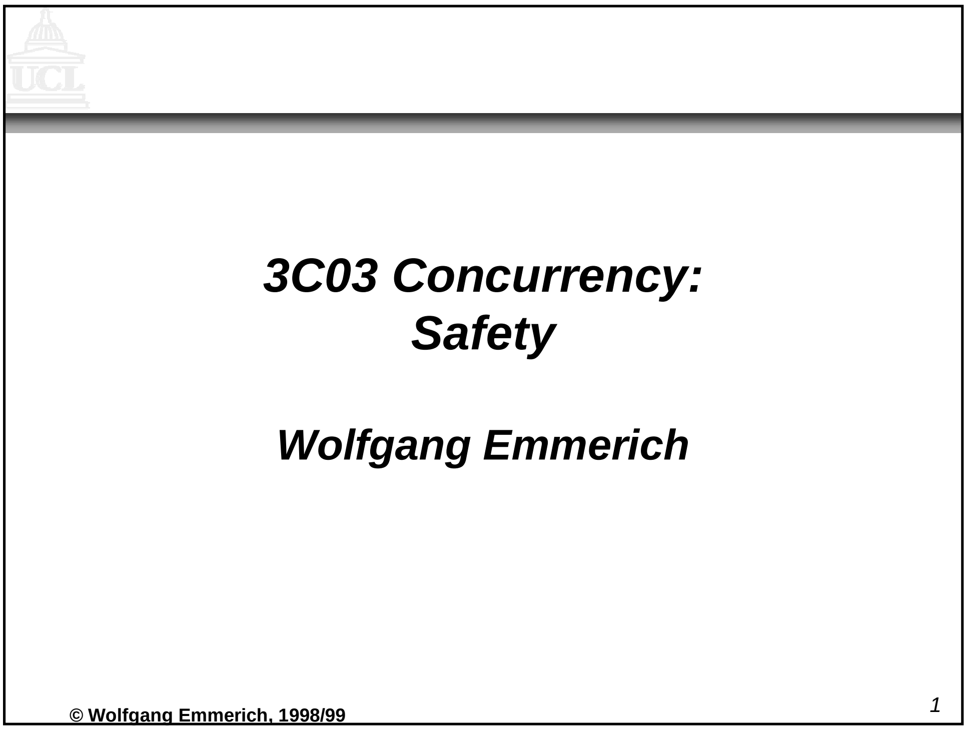
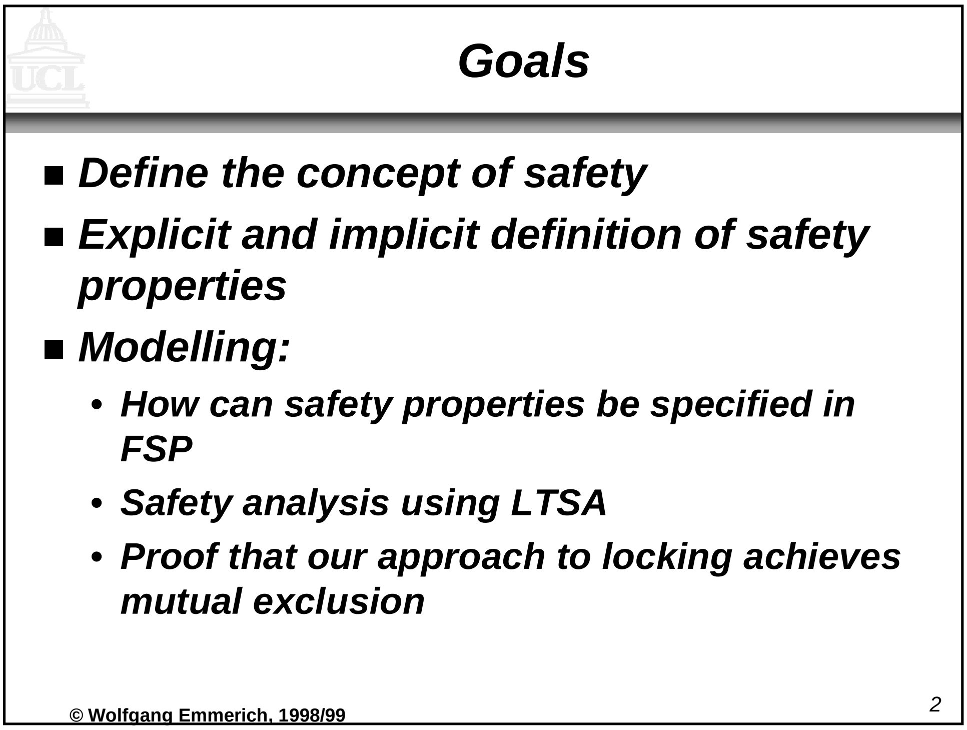


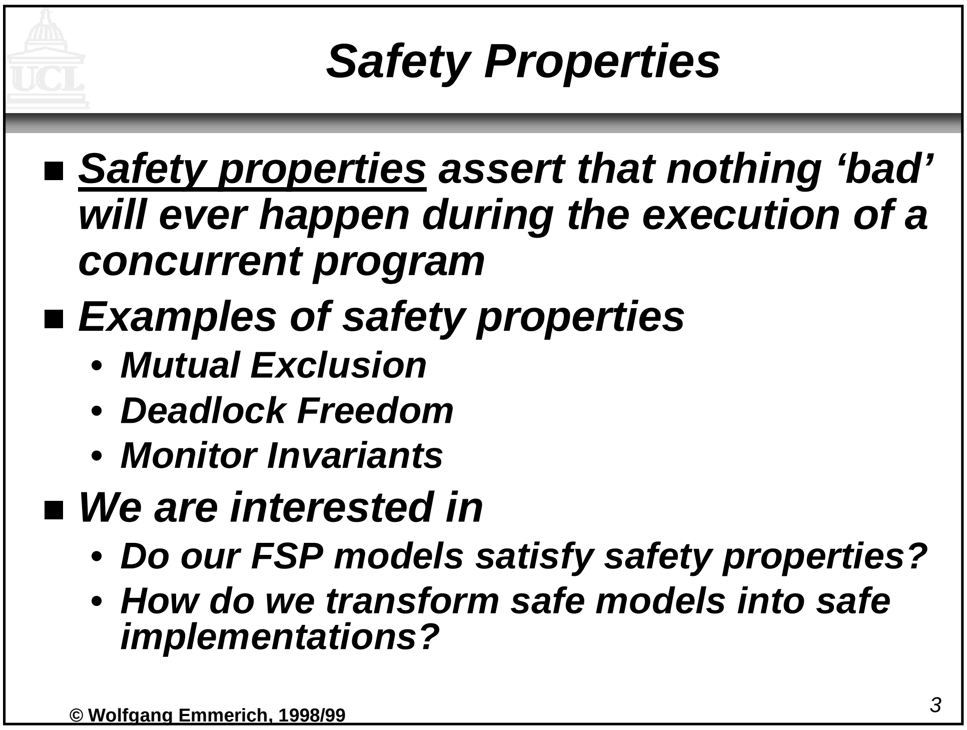
3C03 Concurrency: Safety

Wolfgang Emmerich



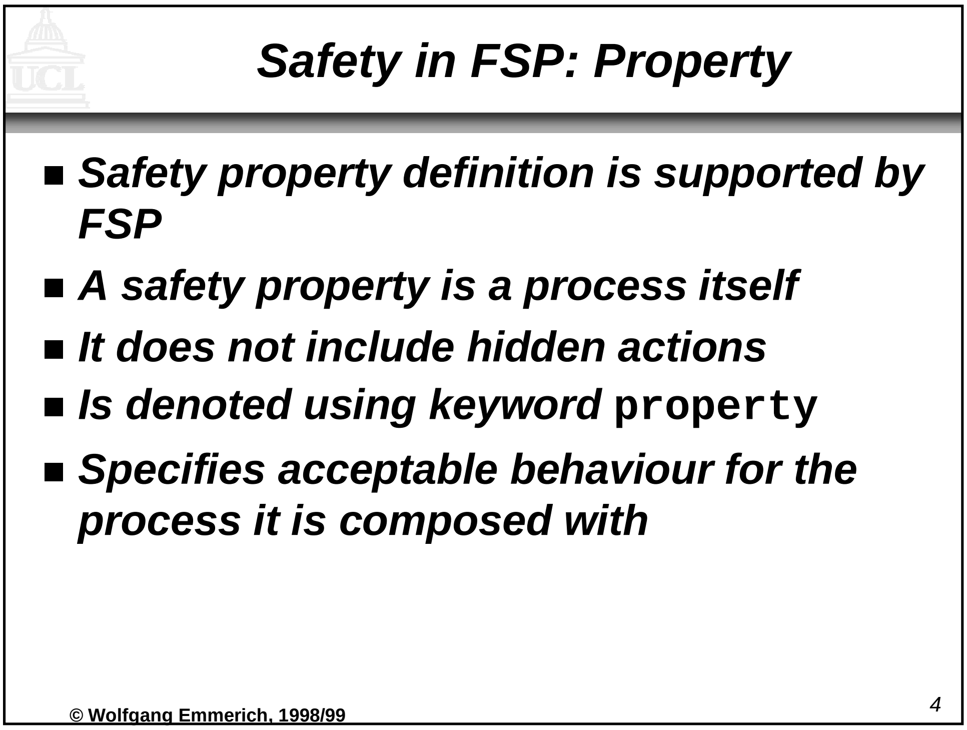
Goals

- Define the concept of safety
- Explicit and implicit definition of safety properties
- Modelling:
  - How can safety properties be specified in FSP
  - Safety analysis using LTSA
  - Proof that our approach to locking achieves mutual exclusion



Safety Properties

- ***Safety properties*** ***assert that nothing 'bad' will ever happen during the execution of a concurrent program***
- ***Examples of safety properties***
  - ***Mutual Exclusion***
  - ***Deadlock Freedom***
  - ***Monitor Invariants***
- ***We are interested in***
  - ***Do our FSP models satisfy safety properties?***
  - ***How do we transform safe models into safe implementations?***



Safety in FSP: Property

- ***Safety property definition is supported by FSP***
- ***A safety property is a process itself***
- ***It does not include hidden actions***
- ***Is denoted using keyword*** `property`
- ***Specifies acceptable behaviour for the process it is composed with***

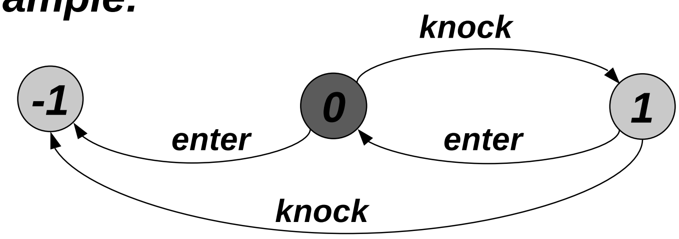
Safety in FSP: Property Satisfaction

- *A system S will satisfy a property P if S can only generate sequences of actions which when restricted to the alphabet of P, are acceptable to P.*
- *Example:*

```
property POLITE=(knock->enter->POLITE).
HESITANT = (knock->knock->enter->HESITANT).
IMPATIANT = (enter->IMPATIANT).
||CHK_HES = (HESITANT || POLITE).
||CHK_IMP = (IMPATIANT || POLITE).
```

Properties in LTS

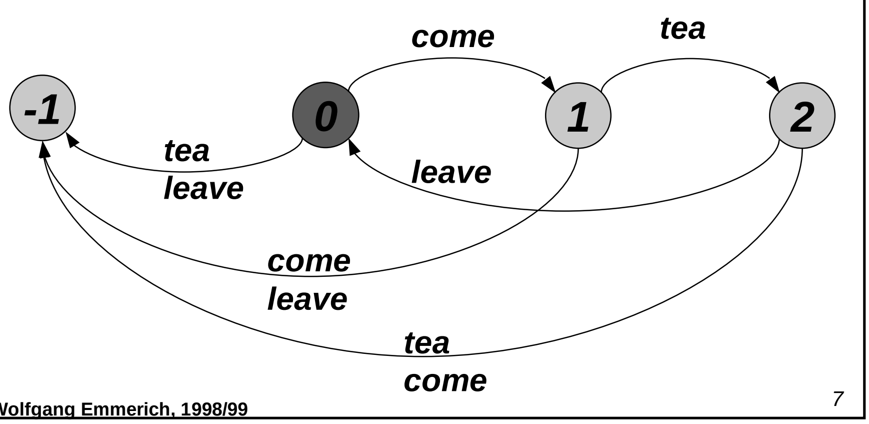
- *LTS generated for properties have*
  - *an additional error state (-1)*
  - *transitions leading to the error state for actions that would violate the property*
- *Example:*



Exercise

- ***Draw the LTS for***

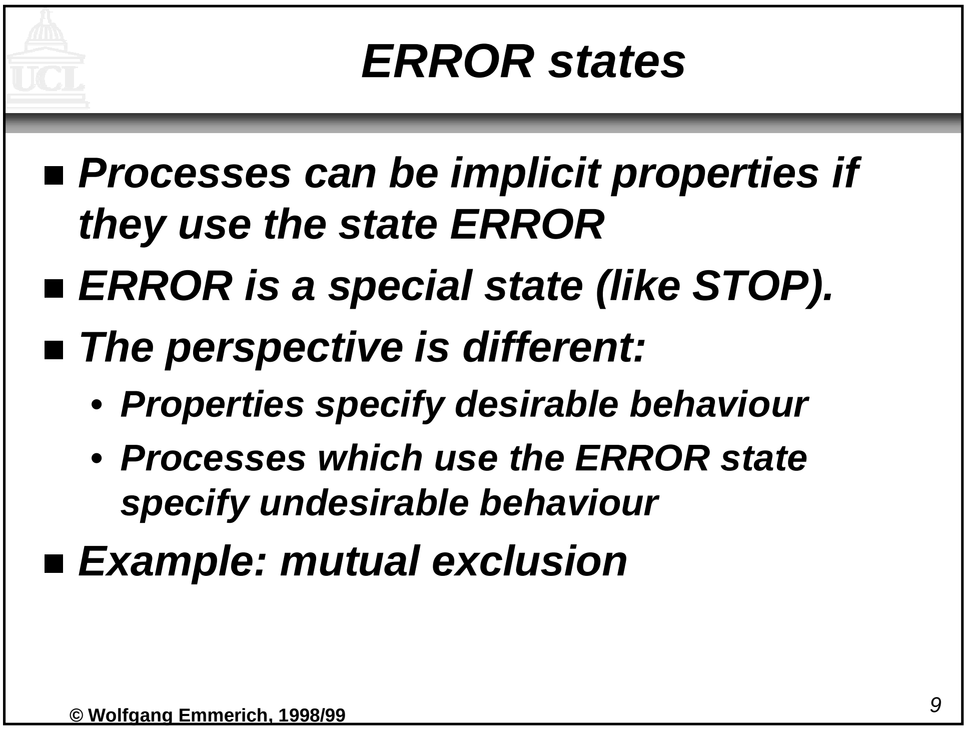
```
property FRIEND=(come->tea->leave->FRIEND).
```



Safety Analysis using LTSA

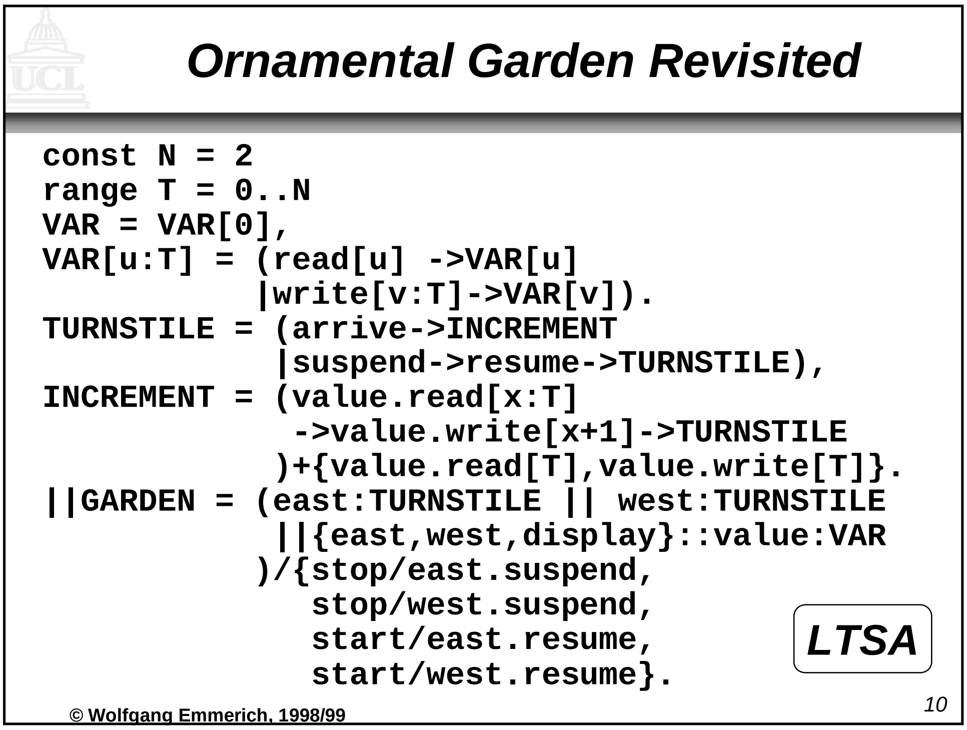
- ***We automate safety analysis using the Labelled Transition System Analyser***
- ***LTSA can***
  - ***compute the LTS for a safety property***
  - ***compose the property with the process to be checked***
  - ***If there is a trace from the initial state to the error state the system is unsafe***

LTSA



ERROR states

- Processes can be implicit properties if they use the state ERROR
- ERROR is a special state (like STOP).
- The perspective is different:
  - Properties specify desirable behaviour
  - Processes which use the ERROR state specify undesirable behaviour
- Example: mutual exclusion



Ornamental Garden Revisited

```
const N = 2
range T = 0..N
VAR = VAR[0],
VAR[u:T] = (read[u] ->VAR[u]
           |write[v:T]->VAR[v]).
TURNSTILE = (arrive->INCREMENT
            |suspend->resume->TURNSTILE),
INCREMENT = (value.read[x:T]
             ->value.write[x+1]->TURNSTILE
            )+{value.read[T],value.write[T]}.
||GARDEN = (east:TURNSTILE || west:TURNSTILE
            ||{east,west,display}::value:VAR
           )/{stop/east.suspend,
              stop/west.suspend,
              start/east.resume,
              start/west.resume}.
```

LTSA

Mutual Exclusion as Safety Property

```
TEST = TEST[0],
TEST[v:T] =
  (when (v<N)
     {east.arrive,west.arrive}->TEST[v+1]
  |stop->CHECK[v]),
CHECK[v:T] = (display.value.read[u:T] ->
             (when (u==v) start -> TEST[v]
             |when (u!=v) wrong -> ERROR)
             )+{display.value.read[T],
                display.value.write[T]}.
||TESTGARDEN = (GARDEN || TEST).
```

LTSA

FSP Model for Locking

```
VAR = VAR[0],
VAR[u:T]=(read[u]->VAR[u]
         | write[v:T]->VAR[v]).
LOCK = (acquire->release->LOCK).
||LOCKVAR = (LOCK || VAR).
TURNSTILE = (arrive->INCREMENT
             |suspend -> resume -> TURNSTILE),
INCREMENT = (value.acquire->value.read[x:T]
             ->value.write[x+1]
             ->value.release->TURNSTILE
            )+ {value.read[T],value.write[T]}.
||GARDEN = (
  east:TURNSTILE || west:TURNSTILE ||
 {east,west}::value:LOCKVAR)
 /{stop/east.suspend,stop/west.suspend,
   start/east.resume,start/west.resume}.
```

Safety Properties for Locking

```
TEST       = TEST[0],
TEST[v:T]  = (when
  (v<N){east.arrive,west.arrive}->TEST[v+1]
  | stop->CHECK[v]),
CHECK[v:T] = (display.value.read[u:T] ->
               (when (u==v) start -> TEST[v]
               |when (u!=v) wrong -> ERROR)
             )+{display.value.read[T],
                display.value.write[T],
                display.value.acquire,
                display.value.release}.
||TESTGARDEN = (GARDEN || TEST).
```

LTSA

Summary

- *Introduced the concept of Safety*
- *Specification of Safety Properties in FSP*
- *Checking of Safety Properties using LTSA*
- *Proof of Mutual Exclusion based on Locking*
- *Next Session: Revision and Tutorial on Model Checking*