Aula 4

1 Exclusão Mútua

Um exemplo: exclusão mútua

Exclusão mútua

Quando processos concorrentes partilham um recurso, poderá ser necessário garantir que não o acedam em simultâneo.

Cada processo terá uma *secção crítica* e só um processo poderá estar nessa secção em cada instante.

Vamos modelar um sistema com dois processos ($\mathcal{M}$) e especificar que propriedades deverá satisfazer (ϕ) (protocolo).

Propriedades da exclusão mútua

Safety (Segurança) Só um processo pode estar na sua secção crítica em cada instante

Liveness Sempre que um processo requer entrar na secção crítica, ele irá entrar nessa secção num instante futuro.

Non-blocking Um processo poderá sempre requerer entrar na secção crítica.

Not strict sequencing Os processos não necessitam de entrar na secção crítica em sequência ($c_i \rightarrow c_{i+1}$).

Construção de um modelo

Cada processo i poderá estar num dos seguintes estados:

n_i estado não crítico

t_i requer entrar no estado crítico

c_i estado crítico

A sequência será $n_i \rightarrow t_i \rightarrow c_i \rightarrow n_i \rightarrow \cdots$.

Supomos que cada processo é assíncrono: em cada instante só um processo executa (*interleaving*).

Em cada instante muda-se de estado.

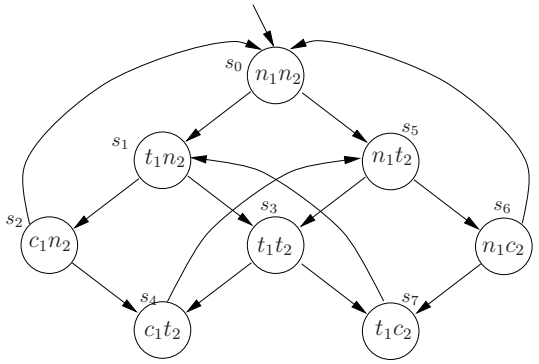
Para dois processos o modelo deverá ter quantos estados?

n_1t_2, n_1c_2, t_1n_2, t_1t_2, t_1c_2, c_1n_2, c_1t_2.

Construção de um modelo

- Construa um modelo $\mathcal{M}$ para o protocolo.
- Exprima as condições de 'safety', 'liveness', 'non-blocking' e 'not strict sequencing' como fórmulas em LTL (se possível).
- Verifique, se as fórmulas são válidas em $\mathcal{M}$.

Construção de um modelo ($\mathcal{M}_1$)



Especificação das propriedades em LTL

Safety $G\neg(c_1 \wedge c_2)$

Esta fórmula é satisfeita no modelo $\mathcal{M}_1$.

Liveness $G(t_1 \rightarrow Fc_1)$

Esta fórmula não é satisfeita no modelo $\mathcal{M}_1$. Existe um caminho a partir de s_0 em que t_1 se verifica mas não c_1. (?)

Non-blocking Cada estado em que n_i se verifica existe um sucessor tal que t_i se verifica. Isto não se pode exprimir em LTL.

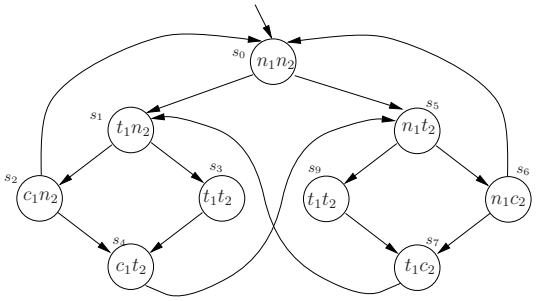
Not strict sequencing Existe um caminho com dois estados que satisfazem c_1, tal que nenhum estado entre eles verifica c_2. Isto não se pode exprimir directamente em LTL. Mas, usando o complementar: Todos os caminhos com um período em que c_1 é satisfeito,mas que termina, não podem ter c_1 antes de ter c_2:

$$G(c_1 \rightarrow c_1 \mathrm{W}(\neg c_1 \wedge \neg c_1 W c_2))$$

Complementando temos a propriedade pretendida, e que se verifica no modelo $\mathcal{M}_1$. Porquê?

Modificação do modelo ($\mathcal{M}_2$)

Para que a *liveness* se verifique é preciso separar o estado t_1t_2 em dois:



2 NuSMV - model checker

NuSMV - model checker

- Extensão do SMV (Symbolic Model Verifier).
- NuSMV: New Symbolic Model Checker.
- Suporta CTL e LTL.
- Input: descrição de um modelo + especificação.
- Output: `true` ou um traço onde a especificação não se verifica.
- http://nusmv.fbk.eu

NuSMV - model checker

Linguagem: permite descrição de sistemas de transições e especificações

Módulos: a descrição pode ser decomposta em módulos que podem ser instanciados diversas vezes

```
MODULE inverter(input)
```

Variáveis: inteiros (n..m), booleanos, enumerações, arrays

Declaração:

$$\begin{array}{l} VAR \\ \quad request : boolean \\ \quad status : \{ready, busy\}; \end{array}$$

Atribuição

$$\begin{array}{l} ASSIGN \\ \quad init(v) = 0; \\ \quad next(v) = !v; \end{array}$$

Estados: obtidos a partir dos valores possíveis das variáveis

NuSMV - model checker

Especificações fórmulas a serem testadas (`LTL` ou `CTL`).

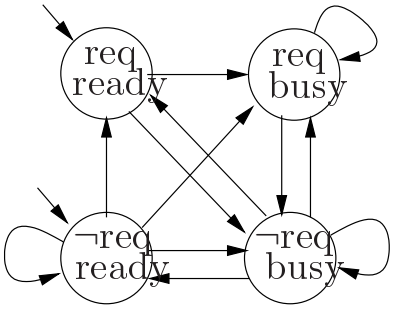
$\neg$		!
$\wedge$		&
$\vee$	$\Longrightarrow$	|
$\rightarrow$		->
$\leftrightarrow$		<->

$$\begin{array}{l} LTLSPEC \\ \quad G(request -> Fstatus = busy) \end{array}$$

NuSMV - model checker

```
MODULE main
VAR
  request : boolean;
  status : {ready,busy};
ASSIGN
  init(status) := ready;
  next(status) := case
                    request = TRUE: busy;
                    TRUE : {ready,busy};
                  esac;
LTLSPEC
  G(request -> F status=busy)
```

NuSMV - model checker




```
MODULE main
VAR
  request : boolean;
  status : {ready,busy};
ASSIGN
  init(status) := ready;
  next(status) := case
                    request = TRUE : busy;
                    TRUE : {ready,busy};
                  esac;
LTLSPEC
  G(request -> F status=busy)
```

Módulos síncronos ou assíncronos

Por omissão a composição de módulos é síncrona: em cada instante os módulos são executados em paralelo (relógio global).

Directiva `process` permite que os módulos sejam compostos assíncronamente:em cada instante um dos módulos é escolhido de forma não-determinística.

```
MODULE cycle
VAR
  v : boolean;
  ASSIGN
    init(v) := 0;
    next(v) := !v;


MODULE main
Var
  a : process cycle()
  c : process cycle()
```

Bit counter: 000 $\rightarrow$ 111

```
MODULE main
VAR
  bit0 : counter_cell(1);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);

MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := 0;
  next(value) := (value + carry_in) mod 2;
DEFINE
  carry_out := value & carry_in;
```

Exclusão Mútua

Variáveis usadas por cada processo:

`st` estado do processo (`n`,`t` ou `c`); o processo pode-se manter no estado em que está

`other-st` estado do outro processo

`turn` indica qual o processo que entra no estado crítico (se ambos o poderem fazer)

Com a introdução de `turn` o modelo agora tem mais estados; pode ter lacetes; e as transições podem indicar qual o processo que executa.

Exclusão Mútua

```
MODULE prc(other-st, turn, myturn)
   VAR
      st: {n, t, c};
   ASSIGN
      init(st) := n;
      next(st) :=
         case
            (st = n)                                    : {t,n};
            (st = t) & (other-st = n)                   : c;
            (st = t) & (other-st = t) & (turn = myturn): c;
            (st = c)                                    : {c,n};
            1                                           : st;
         esac;
      next(turn) :=
```

```
        case
           turn = myturn & st = c : !turn;
           TRUE                   : turn;
        esac;
   FAIRNESS running
   FAIRNESS  !(st = c)
```

`FAIRNESS`

Pode-se restringir a computação a caminhos onde uma determinada fórmula seja verdadeira um número infinito de vezes.

Isto permite modelar que o acesso aos recursos é justo (`fair`).

A instrução

$$\texttt{FAIRNESS}\ \phi$$

garante que o SMV apenas verifica uma especificação ψ em caminhos que ϕ é satisfeita um número infinito de vezes.

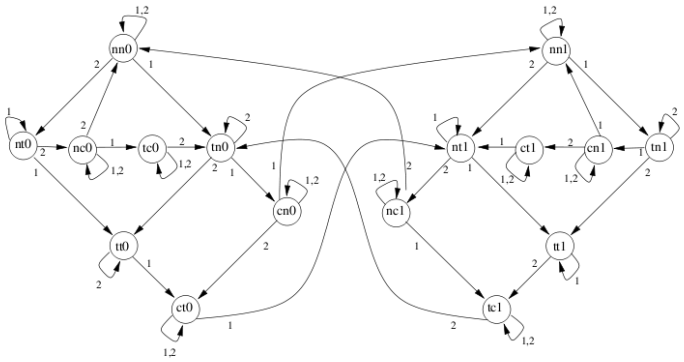
No caso da exclusão mútua, consideram-se:

`FAIRNESS running` (o módulo onde aparece está a executar n.i.v.)

`FAIRNESS !(st=c)`

Exclusão Mútua

```
MODULE main
   VAR
      pr1: process prc(pr2.st, turn, FALSE);
      pr2: process prc(pr1.st, turn, TRUE);
      turn: boolean;
   ASSIGN
      init(turn) := FALSE;
   -- safety
   LTLSPEC  G!((pr1.st = c) & (pr2.st = c))
   -- liveness
   LTLSPEC  G((pr1.st = t) -> F (pr1.st = c))
   LTLSPEC  G((pr2.st = t) -> F (pr2.st = c))
   -- negation of strict sequencing (desired to be false)
   LTLSPEC G(pr1.st=c -> ( G pr1.st=c | (pr1.st=c U
             (!pr1.st=c & G !pr1.st=c |
             ((!pr1.st=c) U pr2.st=c)))))
```



As etiquetas nas transições correspondem ao processo que executa. Neste caso não há a obrigatoriedade de um processo mudar de estado. Daí os lacetes e a necessidade das condições de `Fairness`.