Propriedades de Safety e Liveness

Safety: não acontece nada errado durante

Liveness: alguma hora algo bom acontece

Programa sequencial

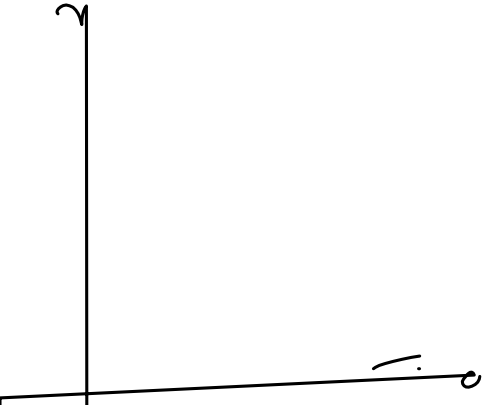
- Safety: estado final correto
- Liveness: chega ao estado final

Programa concorrente

Safety {
- exclusão mútua - não há execução simultânea
- ausência de deadlock - não vai existir a situação "todos esperando"

Livenes:

- Um processo entrará em algum momento



Justiça (Fairness) e políticas de escalonamento

Garantia que todos os processos tem chance de prosseguir

Justiça incondicional - toda ação atômica possível de execução é executada em algum momento (ninguém espera para sempre)

Ex: round-robin

Justiça fraca

- é incondicionalmente justa
- cada ação condicional atômica é executada em algum momento se a condição física fica e permanece verdadeira até que seja vista

Ex.: round-robin, time-slicing

Justiça forte

- é incond. ~~forte~~ justa
- toda ação condicional atômica "executável" é executada em algum momento, assumindo que esta condição é frequentemente verdadeira.

```
boolean continue = true, try = false;
co {
    while (continue.{
        try = true;
        try = false;
    }
    //
    < await (try) continue = false; >
}
```

dá para implementar justiça forte?

não é praticável pois o S.O. teria que entender o programa

Andrews - ex 2.17

Considere:

```
co {
    < await (x >= 3) x = x - 3; >
    //
    < await (x >= 2) x = x - 2; >
    //
    < await (x == 1) x = x + 5; >
}
```

Para que valores de x o programa termina? Com que justiça? Quais os valores finais de x?

incondicional: tenta uma vez somente

fraca: tenta novamente depois que ficar verdadeiro

$x = 1 \Rightarrow x = 1$

6 6

5 } pode acabar, conforme a ordem
4 }

2.33 Considere

```
int x = 10, c = true,
co {
   <await x == 0>; c = false;
   //
   while (c) <x = x-1;>
}
```

a) Termina com justiça fraca?

b) Termina com j. forte!

, e se colocarmos outra barra

```
while (c) { if (x == 0) <x = 10;> }
```

a) não necessariamente

b) depende da linguagem (só se "der a volta" nos inteiros

c) No caso B, termina.

Locks e Barreiras

cap 3 Andrews
cap 3 Ben-Ari

Problema da seção crítica. Neste problema n processos executam repetidamente uma seção crítica e após, uma seção não crítica.

A seção crítica é precedida de um protocolo de entrada e seguida por um protocolo de saída

```
process CS[i = 1 to n] {
    while (true) {
        protocolo de entrada
        seção crítica
        protocolo de saída
        seção normal (não-crítica)
    }
}
```

Suposição: um processo que entra na seção crítica sai dela.

Os protocolos de entrada e saída devem obedecer as propriedades.

Exclusão mútua

SAFETY

Ausência de deadlock - se dois ou mais processos tentam entrar nas suas seções críticas ao menor 1 vai conseguir

Ausência de atraso desnecessário - se um processo está tentando entrar em sua seção crítica, e os outros processos estão executando seções não críticas, ou terminaram, o processo não é impedido de entrar

liveness (Entrada garantida - um processo que está aguardando a entrada na sua seção crítica irá entrar em algum momento.