Formas de verificar as propriedades:

- Teste/Depuração ⇒ verificar um nº limitado
- Análise Exaustiva ⇒ existem geral/e MUITAS
- Análise Abstrata

histórias

13/03

Exemplo: Paralelização da busca de padrões em arquivo

grep "padrão" [arquivo].

Sequencial:

```
string line;
lê uma linha da entrada em line.
while (! EOF) {
    procure o padrão em line
    if ( padrão ∈ line)
        write line
    leia a próxima linha
}.
```

Procurar partes independentes.

Definição: Seja o conjunto de leitura de um programa as variáveis lidas, mas não alteradas. Seja o conjunto de escrita as variáveis que são gravadas (e possivelmente lidas). Duas partes de um programa são independentes se a interseção do conjunto de escrita de cada uma delas é disjunto dos conjuntos de leitura e escrita da outra.

As vezes é possível que dois processos possam ser executados em paralelo, mesmo com gravação nos mesmos elementos, mas isto só é possível apenas quando esta ordem não é importante.

Possível concorrência:

busca e leitura simultânea.

```
while (!EOF) {
    co {
        procura o padrão em line
        if (padrão ∈ line)
            write line
        // leia a próxima linha
    }
}
```

busca e leitura são independentes, mas busca e leitura na mesma variável não são independentes → string line

Então isso tá errado

```
string line1, line2;
leia uma linha em line1;
while (!EOF) {
    co {
        procure o padrão em line1
        if (padrão ∈ line1)
            write line1;
        // leia a próxima linha em line2;
    }
    line1 = line2;
}
```

← sincronização implícita: só pode jogar line2 em line1 quando acabar a busca e a procura. (→ semântica do co)

Problema: procurar padrão e ler linha é rápido. Aqui temos o ~~dinheiro~~ custo de criar processo. Custo de criar e destruir processo o tempo todo, melhor paralelizar de outro jeito.

Como melhorar? Tirar o co do laço.

```
string buffer;
boolean done = false;
co {        # processo 1.
    string line1;
    while (true) {
        wait buffer cheio (ou done == true;)   condição de sincronização - explícita.
        if (done)
            break;
        line1 = buffer;
        signal buffer vazio   manda um aviso pro wait.
        procura padrão na line1
(*)     if (padrão ∈ line1)                    O signal não se perde
            write line1
    }
// # processo 2.
    string line2
    while (true) {
(*)     leia próxima linha em line2
        if (eof) {
            done = true; signal buffer cheio;
            break;
        }
        wait buffer vazio;
        buffer = line2;
        signal buffer cheio;
    }
}
```

Problemas ① depende que tá o programa, ele trava, pq → deadlock no começo os dois processos vão parar no wait.

Eu tenho que liberar um wait na 1ª iteração → libera o wait de buffer vazio

→ solução: colocar um signal buffer vazio antes de qualquer while

(2)

Pode perder a última linha se achar EOF.

Solução: fica ou done == true do wait do 1º processo.
e coloca wait buffer vazio antes do if (EOF) no 2º processo

Encontrar o máximo de um vetor.

Encontrar o máximo em a[n] com elementos positivos e inteiros

Sequencial:
```
int m = 0;
fa [i = 0 to n-1] {
    if (a[i] > m)
        m = a[i];
}.
```

1ª) troca fa por co.
2ª) cria regiões críticas.
3ª) cria regiões críticas de m = a[i]

4ª solução:
```
int m = 0;                                   double check.
co [i = 0 to n-1] {
    if (a[i] > m)   ← ato mais if's melhor.
        < if (a[i] > m)
            m = a[i]; >
}.
```

17/03

Ações atômicas e o comando await.

duas formas de obtermos "ordem" corretas.

- exclusão mútua: combinar ações em blocos atômicos
- condição de sincronização: atrasar a execução até que uma condição seja válida.

Ações atômicas de grão fino são implementadas pelo hardware.

Exemplo: int y = 0, z = 0;

```
co {
    <x = y + z;>
//  <y = 1>; <z = 2>
}
```

quais os valores possíveis de x?
0, 1, 3, 2.