Programação com variáveis compartilhadas 10/03
(contexto de memória compartilhada → multicore por exemplo).
Capítulo 2 { Andrews
Ben-Ari.

Definição: Um programa concorrente consiste de um número finito de processos. Cada processo é escrito usando um conjunto finito de instruções atômicas (indivisíveis).

Definição: O estado de um programa é o conteúdo de suas variáveis em um dado momento.

variáveis { explícitas
implícitas { registradores
ponteiro de controle (onde tá a base e o topo da pilha, PC = program counter)

No contexto concorrente, cada processo tem seu PC.

ponteiro de controle { - único para cada processo.
- indica a próxima instrução a ser executada.

um processo executa uma sequência de instruções.

instruções: sequência de uma ou mais ações atômicas { indivisível
programa e/ou altera o estado do programa.

A execução de um programa concorrente consiste de sequências de instruções atômicas intercaladas (= não tem execução simultânea)

História uma sequência de execução de ações atômicas.

Exemplo: 3 processos P: p1, p2, p3, p4 cada processo com 4
Q: q1, q2, q3, q4 ações atômicas.
R: r1, r2, r3, r4.

Aqui tem muitas histórias possíveis

Algumas delas são inválidas por exemplo: não pode fazer q3 antes de p1 (porque p1 altera algo que q3 olha) Como faz isso?
(colocar várias instruções antes que q1 não é solução → não funciona sempre).
Para isso, vamos usar uma primitiva, uma barreira.

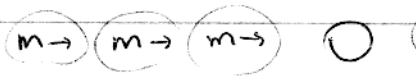
Quais as possíveis execuções de P: p1, p2 ? { p1 p2 q1 q2 ; q1 q2 p1 p2 ; p1 q1 p2 q2
Q: q1, q2.

O compilador tem o direito de olhar as instruções p1, p2 e observar que elas são independentes e executar p2 antes de p1
Então pode acontecer p2 q1 q2 p1
O processador também pode mudar a ordem do código compilado

Papel da sincronização ⇒ restringir o conjunto de histórias possíveis a um conjunto desejável. Garantindo que q3 não seja executada antes de p1
2 formas: I) exclusão mútua "amarra" ações atômicas
Ex: P: <p1, p2>, p3, p4
Significa que p1 e p2 são atômicas. p1 executa p1, logo tem que executar p2. Se p1 coloca 1000 numa variável e p2 coloca zero, nenhum outro processo vê esse 1000 [criar regiões que "pareçam" atômica].

II) Condição de sincronização espera alguma coisa acontecer para continuar [atrasar uma ação até que seja válida uma condição].

Exemplo Frog Puzzle.



dois movimentos possíveis: 1) pular ao espaço vazio ao lado.
2) pular um sapo e cair em um espaço vazio.

• Dá pra chegar no estado : m f f ○ m m m ?

Imagina cada sapo como um processo e vê quais os estados possíveis:

inicial: m m ○ f f

1) ○ m m f f travou: estado final
2) m ○ m f f
3) m m f ○ f
4) m m f f ○ travou: estado final.
5) m f m ○ f ←...
6) m ○ f m f ←...

Análise final de estados de concorrência. Dá pra ver a chance de chegar no estado final desejável. (nesse caso ⇒ 1/6 aprox)

Exemplo: n = 0;
co {
n = n+1; ①
// n = n+1; ②
}

Qual o valor final de n?

Se exclusão mútua : n = 2

Pode dar 1 se quebrar a instrução.

Mas também pode dar qualquer coisa: se interrompe a soma no meio. O outro vai acessar o n onde tem uma soma feita parcialmente

safe

Outro exemplo:

```
n = 0;
co {
    temp1 = n;
    n = temp1 + 1;
// temp2 = n
    n = temp2 + 1;
}
```

No final: pode ser 2; pode ser 1.

Assume-se que a atribuição é atômica. Então não pode ter qualquer coisa → só vai atribuir ali se a soma estiver completa.

Propriedades de um programa

atributo verdadeiro para todas execuções parciais.

dois tipos:

- safety - o programa nunca entra em um estado ruim (estoura a pilha por exemplo)
  ex: em SO o mouse sempre é tratado
- liveness - em algum momento o programa entra em um estado desejável (prog acaba)
  ex: em SO ao clicar temos uma resposta

exemplo de safety: corretude parcial: se o programa termina ele produz a resposta certa

liveness: toda história possível termina

Corretude TOTAL: os dois juntos → O programa sempre termina e em um estado desejável

Em um contexto de Concorrência

- safety: exclusão mútua
- liveness: entrada na seção crítica

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: