

Processos e Sincronização

Noemi Rodriguez

August 26, 2009

Execução de um programa

- sequência de ações *atômicas* ou indivisíveis
 - instruções de máquina
- ações fazem processo transitar entre *estados*
 - tipicamente definidos por variáveis globais
 - possivelmente também (ou apenas) pela pilha de execução
- uma execução específica pode ser vista como uma *história* (sequência de estados) $s_0 \rightarrow s_1 \rightarrow \dots \rightarrow s_n$
 - programas concorrentes: número enorme de possíveis histórias
 - sincronização: corte de histórias indesejáveis

- *safety*
 - programa nunca entra em estado ruim
- *liveness*
 - programa em algum momento entra em estado bom

Como demonstramos determinada propriedade?

- raciocínio operacional
 - análise exaustiva de casos
- assertivas
 - assertivas lógicas usadas para caracterizar estados
 - $x > 0$ ou $x \neq y$

Notação: atomicidade e *await*

- instruções de máquina são atômicas numa granularidade fina
- para definir ações atômicas com granularidade maior, Andrews introduz a notação $\langle e \rangle$
- para especificar sincronização, Andrews introduz a notação:

$\langle \text{await}(B)S; \rangle$

que significa que S só deve começar a ser executado quando B for verdadeira, e que S será executado atomicamente

Exemplo

```
int buf, p = 0, c = 0;
process Producer {
    int a[n];
    while (p < n) {
        < await (p == c);>
        buf = a[p];
        p = p+1;
    }
}
process Consumer {
    int b[n];
    while (c < n) {
        < await (p > c);>
        b[c] = buf;
        c = c+1;
    }
}
```

Implementação de atomicidade

- exclusão mútua e regiões críticas

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

- protocolos de entrada e saída devem satisfazer as seguintes condições:
 - 1 exclusão mútua
 - 2 ausência de deadlock (ou livelock)
 - 3 ausência de esperas desnecessárias
 - 4 entrada em algum momento (“eventual” em inglês)

Como implementar o await com protocolos de EM?

- $\langle S \rangle$ implementado por

```
CSenter();  
S;  
CSexit();
```
- e $\langle \text{await}(B)S; \rangle$? podemos testar a condição dentro da região crítica:

```
CSenter();  
while (!B) (???)  
S;  
CSexit();
```

mas se a alteração da condição B depender de outro processo entrar na região crítica, nada feito...

- podemos tentar:

```
CSenter();  
while (!B) (CSexit; CSenter;)  
S;  
CSexit();
```

mas teremos grande disputa pela EM...

- aqui poderíamos reaplicar o conceito de *back-off*