PDP - Exercício 2

Arthur Adolfo Gabriel Seibel

7 de Maio de 2019

Introdução

Este documento descreve uma solucão elaborada para a tarefa de reescrever o algoritmo de coleta em árvore usando o modelo assíncrono de sistemas distribuídos de Nancy Lynch [@lynch]. Este paradigma pressupõe processos *multithread*, cada um com uma *thread* que realiza *sends*, outra que realiza *receives*, e uma última que implementa a lógica específica da família de processos descrita em código.
O algoritmo em questão, também conhecido como Probe/Echo, é caracterizado por duas fases:

1. Difusão de pedido de informação (*probe*)
2. Coleta de informação (*echo*)

Na primeira fase, cada processo da árvore envia aos seus vizinhos (salvo o nodo "pai") uma mensagem de *probe*. Depois, na segunda fase, o processo espera seus filhos (nodos a quem ele enviou *probe anteriormente*) lhe retornarem um *echo* contendo alguma informação relevante (a topologia do grafo, por exemplo).

Pseudocódigo

O pseudocódigo descrito a seguir é feito à semelhança da linguagem de programação Kotlin. O funcionamento do mecanismo é explicado na sessão seguinte. No código é definida uma classe para representar a topologia do grafo, pois esta foi a escolha de informação a ser coletada pelo algoritmo. Em seguida, é descrita a classe Harvester, subclasse de uma suposta classe Process. É nesta classe "coletora" em que está modelado o algoritmo nos moldes do modelo em questão.

```
class Topology {
    ...
    combine(other: Topology) { ... }
}

class Harvester: Process {
    Signatures {
        Input {
            receive("probe", null) j,i
```

Funcionamento do Mecanismo

A classe Harvester define, como exige o modelo de programação de Lynch, *signature*, *states* e *transitions* de uma família de processos. Sua "assinatura", composta por declarações de transições, é dividida em *input* e *output*, de acordo com a natureza das declaracões. Os "Estados" do processo aglutinam variáveis e suas respectivas inicializações. Por último, são definidas de fato as transicões, invocando rotinas que são executadas se a precondição for verdadeira.

A transição "*start() i*" inicia o processo, executando quando o algoritmo ainda não tiver começado e se *i* (que identifica o processo a atual) for o starterId - que é lido da entrada padrão do programa na inicialização dos processos. A rotina dessa transição manda uma mensagem de "probe" para todos os vizinhos do nodo iniciador. Isso é feito por inserir um par [marcador, topologia] com marcador "probe" e topologia vazia na fila de envios para outro processo, fila esta tratada pela *thread sender*.

As duas transições classificadas como *input*, *"receive("probe", null) j,i"* e *"receive("echo", topo: Topology) j,i"*, podem executar quando

Referências