PDP - Exercício 2

Arthur Adolfo Gabriel Seibel

7 de Maio de 2019

1 Introdução {introducao}

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 1996).

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).

2 abaco1

3 Pseudocódigo {pseudocodigo}

O psuedocódigo descrito a seguir é feito à semelhança da linguagem de programação Kotlin e reflete a estrutura de codificação e representação de sistemas distribuídos de Lynch (Lynch 1996). O funcionamento do mecanismo é explicado na sessão (**???**).

No código anaixo é definida uma classe para representar a topologia do grafo, pois esta foi a escolha de informação a ser coletada. 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
            receive("echo", topo : Topology) j,i
            start() i
        }
    }

    States {
        val starterId: Int = readStdin()
```

```
        val started: Bool = false

        val parentId: Int = 0

        var receivedEchos: Int = 0

        val topology = Topology()

        // clear send buffers
        for (j in nbrs) {
            sendBuffer(j) = emptyQueue<Tag, Topology>()
        }
    }

    Transitions {
        start() i {
            Precondition:
                i == starterId
                started == false
            Effect:
                started = true
                for (nbr in nbrs)
                    sendBuffer(nbr).push(["probe", null])
        }

        receive("probe", null) j,i {
            Precondition:
                started == true
            Effect:
                parentId = j
                for (nbr in nbrs) {
                    if (nbr != parentId)
                        sendBuffer(nbr).push(["probe", null])
                }
        }

        receive("echo", topo: Topology) j,i {
            Precondition:
                started == true
            Effect:
                receivedEchos++
                topology.combine(topo)
                if (receivedEchos == nbrs-1)
                    sendBuffer(parent).push(["echo", topology])
        }
    }
}
```

4 Funcionamento do Mecanismo {funcionameto}

5 abaco

6 Referências {referencias}

Lynch, Nancy A. 1996. *Distributed Algorithms*. Elsevier.