Pistache: uma implementação do π-calculus como linguagem de domínio específico para Scala

O π-calculus consiste em uma álgebra de processos para computação concorrente com reconfiguração dinâmica. Agentes (processos) se comunicam por meio da passagem de nomes através de canais de comunicação. Estes canais, que também são nomes, podem ser trocados entre os agentes envolvidos na computação, reconfigurando-a. Neste trabalho, desenvolveu-se uma biblioteca, exposta como uma linguagem de domínio específico, que implementa o modelo de concorrência do π-calculus em Scala. As especificações são escritas através de uma sintaxe similar à usada na teoria e transformadas em uma estrutura de dados que pode ser executada. O sistema de tipos do Scala é aproveitado de forma a garantir que apenas expressões sintaticamente corretas do cálculo sejam compiláveis.

Pedro Matiello
pmatiello@gmail.com

Ana C. V. de Melo
acvm@ime.usp.br

π-calculus — sintaxe

Prefixos	$\alpha ::=$	$\bar{y}x$	Saída
		$y(x)$	Entrada
		τ	Ação Interna
Agentes	$P ::=$	0	Nulo
		$\alpha.P$	Prefixo
		$P + P$	Soma
		$P \mid P$	Composição
		$(\nu x)P$	Restrição
		$[x = y].P$	Igualdade
		$[x \neq y].P$	Desigualdade
Definições		$A(x_1, ..., x_n) \stackrel{\text{def}}{=} P$	

Troca de mensagens

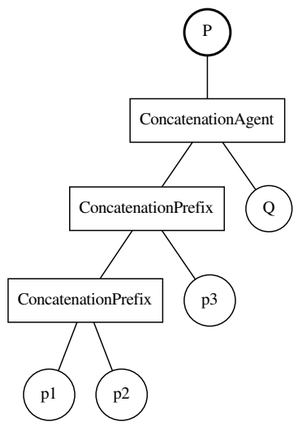
- Canais são buffers
- A comunição é síncrona

Saída $\bar{y}x$
- Espera y estar vazio
- Armazena x em y
- Notifica y preenchido
- Espera y estar vazio

Entrada $y(x)$
- Espera y estar preenchido
- Armazena o conteúdo de y em x
- Notifica y vazio

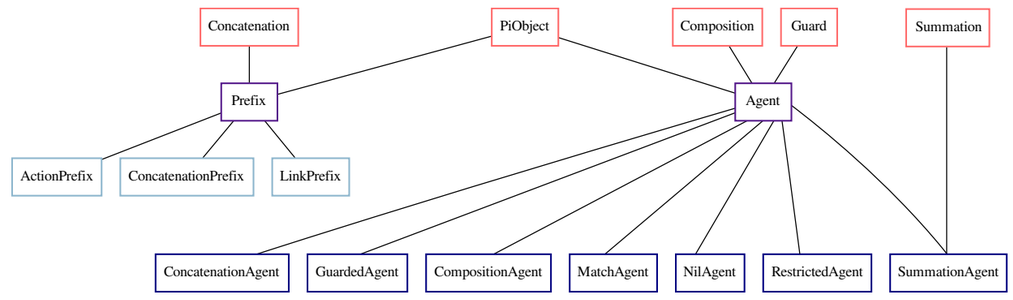
Estrutura de dados

```
Agent ( p1 * p2 * p3 * Q )
```



- A execução da expressão é feita seguindo um percurso em pré-ordem na árvore
- O código exato a ser executado é determinado pela classe do nó avaliado

Hierarquia de tipos



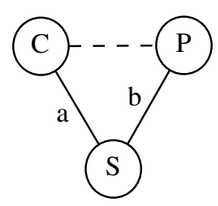
π-calculus — exemplo

Os agentes:

- $C = (\nu p)a(p).\bar{p}x$
- $P = (\nu y)b(y).\tau.P$
- $S = \bar{a}b.S$

A composição:

- $C|P|S$



Pistache — exemplo

```
val a = Link[Link[String]]
val b = Link[String]

val C = Agent {
  val p = Name[Link[String]]
  a(p) * p~"message"
}

lazy val S:Agent = Agent {
  a~b*S
}

lazy val P:Agent = Agent {
  val msg = Name[String]
  val act = Action { println(msg.value) }
  b(msg) * act * P
}

new ThreadedRunner(C | S | P) start
```