A π-Calculus Internal Domain-Specific Language for Scala

Pedro Matiello
pmatiello@gmail.com

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

π-Calculus — Syntax

Prefixes	$\alpha ::=$	$\bar{y}x$	Output
		$y(x)$	Input
		τ	Silent

π-Calculus — Syntax

Prefixes	$\alpha ::=$	$\bar{y}x$	Output
		$y(x)$	Input
		τ	Silent
Agents	$P ::=$	0	Nil
		$\alpha.P$	Prefix
		$P + P$	Sum
		$P \mid P$	Parallel
		$(\nu x)P$	Restriction
		$[x = y].P$	Match
		$[x \neq y].P$	Mismatch

π-Calculus — Syntax

Prefixes	$\alpha ::=$	$\bar{y}x$	Output
		$y(x)$	Input
		τ	Silent
Agents	$P ::=$	0	Nil
		$\alpha.P$	Prefix
		$P + P$	Sum
		$P \mid P$	Parallel
		$(\nu x)P$	Restriction
		$[x = y].P$	Match
		$[x \neq y].P$	Mismatch
Definitions		$A(x_1, \ldots, x_n) \overset{\text{def}}{=} P$	

π-Calculus — Transition Rules

Prefix

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}$$

Restriction

$$\frac{P \xrightarrow{\alpha} P', x \notin \alpha}{(\nu x).P \xrightarrow{\alpha} (\nu x).P'}$$

Match

$$\frac{\alpha.P \xrightarrow{\alpha} P}{[x=x].P \xrightarrow{\alpha} P}$$

Mismatch

$$\frac{\alpha.P \xrightarrow{\alpha} P, x \neq y}{[x \neq y].P \xrightarrow{\alpha} P}$$

π-Calculus — Transition Rules

Parallel $$\frac{P \xrightarrow{\alpha} P', bn(\alpha) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P'|Q}$$

Communication $$\frac{P \xrightarrow{\alpha(x)} P', Q \xrightarrow{\bar{\alpha}u} Q'}{P|Q \xrightarrow{\alpha} P'\{u/x\}|Q'}$$

Summation $$\frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$$

Scala — Variables and Methods

```scala
val string:String = "Not reassignable"
var string:String = "Reassignable"

def max(x:Int, y:Int):Int = {
  if (x > y) x else y
}
```

Scala — Type Inference

```scala
val string = "Not reassignable"
var string = "Reassignable"

def max(x:Int, y:Int) = {
  if (x > y) x else y
}
```

Scala — Classes and Traits

```
trait Person {
  def sleep { Thread sleep 1000 }
  def talk:Unit
}

class NicePerson extends Person {
  def talk { println("Hello") }
}
```

Scala — Objects

```
object Pineapple {
  def eat { println("tasty") }
}

scala> Pineapple.eat
tasty!
```

Scala — Case Classes and Pattern Matching

```
trait Human
case class Man(name:String) extends Human
case class Woman(name:String) extends Human

def whoIs(human:Human) {
  human match {
    case Man(name) => println("He is "+name)
    case Woman(name) => println("She is "+name)
  }
}

scala> whoIs(Man("Joe Doe"))
He is Joe Doe
```

Scala — Implicit Conversions

```
implicit def Int2String(int:Int) = int.toString

def len(str:String) = str.size

scala> len(1234)
res0: Int = 4
```

Pistache — API

Agent definition:

```
val P = Agent(...)

lazy val recP:Agent = Agent(...)

val restrP = Agent {
    val restrictedName = Name(...)
    ...
}

def argP(arg1:Type1, ..., argN:TypeN):Agent =
    Agent { ... }
```

Pistache — API

Names:

```
val name = Name(some_object)

val name = Name[Type]

name := other_object

value = name.value
```

Pistache — API

Links:

```
val link = Link[Type]

link~name

link(another_name)
```

Silent Transitions:

```
val silent = Action{ doSomething() }
```

Pistache — API

Prefix Concatenation:

```
val P = Agent { p1 * p2 * Q }
```

Matching:

```
val P = Agent(If (condition) {Q})
```

Pistache — API

Agent Composition:

```
val P = Agent { Q1 | Q2 | Q3 }
```

Summation:

```
val P = Agent {
    (p1 :: Q1) + (p2 :: Q2) + (p3 :: Q3)
}
```

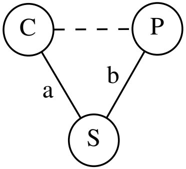
π-Calculus — Example

The agents:

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

The composition:

- $C|P|S$



Pistache — Example

```
object Printserver {
  def main (args:Array[String]) {
    val a = Link[Link[String]]
    val b = Link[String]
  }
}
```

Pistache — Example

```
object Printserver {
  def main (args:Array[String]) {
    val a = Link[Link[String]]
    val b = Link[String]

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

  }
}
```

Pistache — Example

```
object Printserver {
  def main (args:Array[String]) {
    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
}

  }
}
```

Pistache — Example

```
object Printserver {
  def main (args:Array[String]) {
    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
}

  }
}
```

Pistache — Example

```
object Printserver {
  def main (args:Array[String]) {
    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


  }
}
```

Pistache — Message Passing

- Channels are buffers
- Communication is synchronous

Pistache — Message Passing

- Channels are buffers
- Communication is synchronous

Output $\bar{y}x$

- Wait until y is empty
- Put x on y
- Signal y not empty
- Wait until y is empty

Pistache — Message Passing

- Channels are buffers
- Communication is synchronous

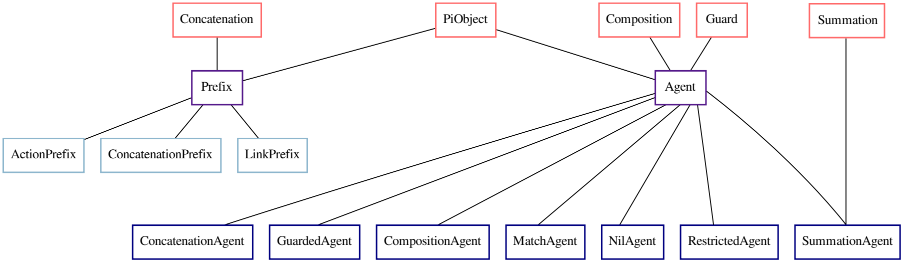
Output $\bar{y}x$

- Wait until y is empty
- Put x on y
- Signal y not empty
- Wait until y is empty

Input $y(x)$

- Wait until y is not empty
- Put the contents of y in x
- Signal y empty

Pistache — Type Hierarchy

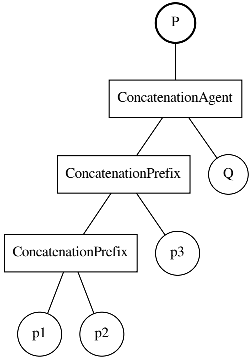


Pistache — Data Structure

```
val P = Agent ( p1 * p2 * p3 * Q )
```

Pistache — Data Structure

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



Pistache — Execution

```
private def execute(agent:PiObject) {
  agent match {
    case Type1(*args) => ...
    case Type2(*args) => ...
    ...
  }
```

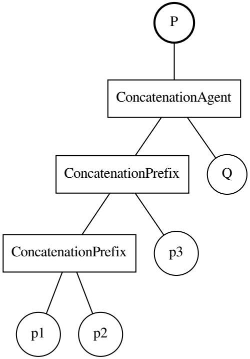
Pistache — Executing Prefix Sequences

```
private def execute(expr:PiObject) {
  expr match {
    ...
    case ConcatenationPrefix(left, right) =>
      execute(left apply)
      execute(right apply)

    case ConcatenationAgent(left, right) =>
      execute(left)
      execute(right)
    ...
  }
}
```

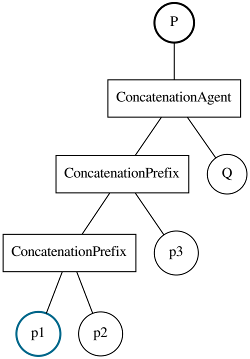
Pistache — Executing Prefix Sequences

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



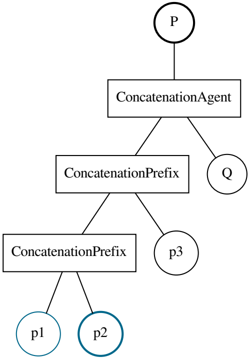
Pistache — Executing Prefix Sequences

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



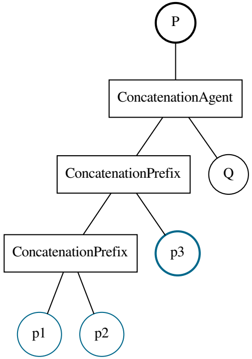
Pistache — Executing Prefix Sequences

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



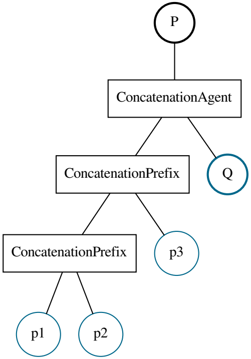
Pistache — Executing Prefix Sequences

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



Pistache — Executing Prefix Sequences

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



Pistache — Executing Agent Compositions

```
private def execute(agent:PiObject) {
  agent match {
    ...
    case CompositionAgent(left, right) =>
      executeInNewThread(left apply)
      executeInNewThread(right apply)
    ...
  }
}
```

Pistache — Spawning Agents

```
private def executeInNewThread(agent:PiObject) {

  val runnable = new Runnable() {
    override def run() { execute(agent) }
  }

  new Thread(runnable) start
}
```

Pistache — Spawning Agents

```
private def executeInNewThread(agent:PiObject) {

  val runnable = new Runnable() {
    override def run() { execute(agent) }
  }

  new Thread(runnable) start
}
```

23743ms to spawn 100,000 agents.

Pistache — Spawning Agents

```
private def executeInNewThread(agent:PiObject) {

  val runnable = new Runnable() {
    override def run() { execute(agent) }
  }

  executor.execute(runnable)
}
```

Pistache — Spawning Agents

```
private def executeInNewThread(agent:PiObject) {

  val runnable = new Runnable() {
    override def run() { execute(agent) }
  }

  executor.execute(runnable)
}
```

2089ms to spawn 100,000 agents.

CachedThreadPool

- Caches finished threads
- Reuses cached threads
- Creates new threads if none are available
- Deletes from the pool threads that have not been reused for 60 seconds

CachedThreadPool

- Caches finished threads
- Reuses cached threads
- Creates new threads if none are available
- Deletes from the pool threads that have not been reused for 60 seconds

Spawning threads is expensive!

Pistache — Termination

```
private def executeInNewThread(agent:PiObject) {

  val runnable = new Runnable() {
    override def run() {
      execute(agent)
    }
  }

  executor.execute(runnable)
}
```

Pistache — Termination

```
private def executeInNewThread(agent:PiObject) {

  increaseThreadCount()

  val runnable = new Runnable() {
    override def run() {
      execute(agent)
      decreaseThreadCount()
    }
  }

  executor.execute(runnable)
}
```

Pistache — Termination

```
private def increaseThreadCount() {
  synchronized {
    threadCount += 1
    notify
  }
}
```

Pistache — Termination

```
private def increaseThreadCount() {
  synchronized {
    threadCount += 1
    notify
  }
}

private def decreaseThreadCount() {
  synchronized {
    threadCount -= 1
    notify
  }
}
```

Pistache — Termination

```
private def waitAllThreads() {
  synchronized {
    while (threadCount != 0) wait;
  }
}
```