Functions with parallel execution and communication channel

Team: Alex Costa (ajc2) e Victor Oliveira (vlo2)

Goal

Extend the functional language 3 to include competition.

LF3

BNF

```
Programa ::= Expressao | ExpMultiprocess
Expressao ::= Valor
 | ExpUnaria
 | ExpBinaria
 | ExpDeclaracao
 | Id
 | Aplicacao
 | IfThenElse
 | ExpMultiprocess
 | ExpExecutor

Valor ::= ValorConcreto | ValorAbstrato
ValorAbstrato ::= ValorFuncao

ValorConcreto ::= ValorInteiro | ValorBooleano | ValorString | ValorLista

ValorFuncao ::= "fn" Id Id "." Expressao

ExpUnaria ::= "-" Expressao | "not" Expressao | "length" Expressao
                        | head(Expressao) | tail(Expressao)
                        | ExpCompreensaoLista
                        | wait(Expressao | ListExp)
                        | receive(Expressao)
```

```
ExpCompreensaoLista ::= Expressao Gerador | Expressao Gerador Filtro

ExpExecutor ::= send(ValorString "," ValorString)

Gerador ::= “for” Id “in” Expressao
               | “for” Id “in” Expressao [“,”] Gerador

Filtro ::= “if” Expressao

ExpBinaria ::= Expressao "+" Expressao
 | Expressao "-" Expressao
 | Expressao "*" Expressao
 | Expressao ">" Expressao
 | Expressao "<" Expressao
 | Expressao "and" Expressao
 | Expressao "or" Expressao
 | Expressao "==" Expressao
 | Expressao "++" Expressao
 | Expressao ".." Expressao
 | Expressao ":" Expressao
 | Expressao "^^" Expressao

ExpDeclaracao ::= "let" DeclaracaoFuncional "in" Expressao
DeclaracaoFuncional ::= DecVariavel
 | DecFuncao
 | DecComposta

DecVariavel ::= "var" Id "=" Expressao
DecFuncao ::= "fun" ListId "=" Expressao

ExpMultiprocess ::= DecProcess  |  ListDecProcess
```

```
ListDecProcess ::= DecProcess  |  ListDecProcess

DecProcesso ::= "process" Id ExpDeclaracao "end"

DecComposta ::= DeclaracaoFuncional "," DeclaracaoFuncional

ListId ::= Id  |  Id, ListId

Aplicacao:= Expressao"(" ListExp ")"

ListExp ::= Expressao  |  Expressao, ListExp
```

Centro de Informática U·F·P·E

Parser (.JJ)

```
488 Expressao PExpWait() :
489 {
490     Expressao retorno;
491     Expressao callback;
492     List<Expressao> callbacks = new ArrayList<Expressao>();
493 }
494 {
495     < WAIT > < LPAREN > retorno = PExpressao()
496     (
497       < COMMA > callback = PExpressao()
498       {
499         callbacks.add(callback);
500       }
501     )*
502     < RPAREN >
503     {
504         if (retorno instanceof ValorInteiro) {
505           ValorInteiro val = (ValorInteiro) retorno;
506         }
507
508         return new ExpWait(retorno, callbacks.toArray(new Expressao[callbacks.size()]
509     }
510 }
```

Centro de Informática U·F·P·E

Parser (.JJ)

```
512 Expressao PExpSend() :
513 {
514   Expressao expressao1;
515   Expressao expressao2;
516 }
517 {
518   < SEND > < LPAREN > expressao1 = PExpressao() < COMMA > expressao2 = PExpressao() < RPAREN >
519   {
520      return new ExpSend(expressao1, expressao2);
521   }
522 }
```

Centro de Informática U·F·P·E

Parser (.JJ)

```
524 Expressao PExpReceive() :
525 {
526   Expressao expressao;
527 }
528 {
529   < RECEIVE > expressao = PExpPrimaria()
530   {
531     if (expressao instanceof ValorInteiro) {
532           ValorInteiro val = (ValorInteiro) expressao;
533     }
534
535     return new ExpReceive(expressao);
536   }
537 }
```

Receive - availar

```
@Override
public Valor avaliar(AmbienteExecucao amb) throws VariavelNaoDeclaradaException, VariavelJaDeclaradaException {

    Expressao timeoutInSeconds = getExp();

    Integer integerTimeoutInSeconds = ((ValorInteiro) timeoutInSeconds.avaliar(amb)).valor();

    String processId = amb.getThreadName();
    AmbienteExecucao mainThreadContext = getMainThreadContext(amb);

    String messageReceived = mainThreadContext.takeMessageFromQueue(processId, integerTimeoutInSeconds);
    System.out.println("Message received: " + messageReceived);

    return new ValorString(messageReceived);
}
```

Centro de Informática U·F·P·E

Send - avaliar

```
@Override
public Valor avaliar(AmbienteExecucao amb) throws VariavelNaoDeclaradaException, VariavelJaDeclaradaException {

    Expressao[] expressions = getExp();

    try {
        Expressao processId = expressions[0];
        Expressao message = expressions[1];

        String valorProcessId = ((ValorString) processId.avaliar(amb)).valor();
        String valorMessageProcess = ((ValorString) message.avaliar(amb)).valor();

        AmbienteExecucao mainThreadContext = getMainThreadContext(amb);
        mainThreadContext.putMessageInQueue(valorProcessId, valorMessageProcess);
        System.out.println("Send msg: " + valorMessageProcess);
        System.out.println("To: " + valorProcessId);

        return new ValorString(valorMessageProcess);
    } catch (ArrayIndexOutOfBoundsException e) {
        throw new TipoParametrosException(TipoPrimitivo.STRING);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    return null;
}
```

Centro de Informática U·F·P·E

ExpMultiprocess

```
@Override
public Valor avaliar(AmbienteExecucao amb) throws VariavelNaoDeclaradaException, VariavelJaDeclaradaException {

    Executor executor = Executors.newFixedThreadPool(100);
    CompletionService<Valor> completionService = new ExecutorCompletionService<Valor>(executor);

    for (DecProcesso decProcesso : decProcessoArray) {
        AmbienteExecucao ambExec = new ContextoExecucao(decProcesso.getId().toString(), amb);
        completionService.submit(decProcesso.avaliar(ambExec));
    }
    int received = 0;
    boolean errors = false;

    Valor valor = null;
    while (received < decProcessoArray.length && !errors) {
        try {
            Future<Valor> resultFuture = completionService.take();
            valor = resultFuture.get();
            received++;
        } catch (Exception e) {
            e.printStackTrace();
            errors = true;
        }
    }

    return valor;
}
```

Centro de Informática U·F·P·E

AmbienteExecucao

```
package lf3.plp.expressions2.memory;

import lf3.plp.expressions2.expression.Valor;

public interface AmbienteExecucao extends Ambiente<Valor> {

    public AmbienteExecucao clone();

    public AmbienteExecucao getParent();

    public String getThreadName();

    public void putMessageInQueue(String processId, String message) throws InterruptedException;

    public String takeMessageFromQueue(String processId, int timeoutInSeconds);

}
```

ContextoExecucao

```
@Override
public void putMessageInQueue(String processId, String message) throws InterruptedException {

    if (messageMap.containsKey(processId)) {
        BlockingQueue<String> blockingQueue = messageMap.get(processId);
        blockingQueue.put(message);

    } else {
        BlockingQueue<String> blockingQueue = new ArrayBlockingQueue<>(DEFAULT_QUEUE_SIZE);
        blockingQueue.put(message);
        messageMap.put(processId, blockingQueue);
    }
}
```

ContextoExecucao

```java
@Override
public String takeMessageFromQueue(String processId, int timeoutInSeconds) {

    BlockingQueue<String> blockingQueue = messageMap.get(processId);

    long startTime = System.currentTimeMillis();
    long waitTime = timeoutInSeconds * 1000;
    long endTime = startTime + waitTime;

    while (blockingQueue == null && System.currentTimeMillis() < endTime) {

        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        blockingQueue = messageMap.get(processId);
    }

    String message = "";

    if (blockingQueue != null) {
        try {
            message = blockingQueue.poll(timeoutInSeconds, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            throw new RuntimeException();
        }
    }

    return message;
}
```

Centro de Informática U·F·P·E

Sintax - creating a wait expression

```
{
  wait(10)
}
```

```
{
  let fun sleep time = wait(time) in sleep(10)
}
```

```
{
  let fun esperar time callback dois = wait(time, callback, dois) in esperar(10, "mensagem1", "mensagem2")
}
```

Centro de Informática U·F·P·E

Sintax - creating a process

```
{
  {
   process p1
       let fun sum x = x in sum(1)
   end
  }
}
```

Sintax - creating a process

```
{
  {
    process p2
    // This will wait until some message is received or 120 seconds timeout
    let fun printRemoteMsg timeout = receive(timeout) in printRemoteMsg(120)
    end

    process p1
    / This will send message 'Teste' to the processe p2
    let fun enviarMensagem idProcesso mensagem = send(idProcesso, mensagem)
          in wait(10, enviarMensagem("p2", "Teste"))
    end
  }
}
```

Centro de Informática U·F·P·E

Repository

https://github.com/victorlaerte/cin-plp-project

Centro de Informática U·F·P·E

References

Martin Brown (10 de maio de 2011). «Introduction to programming in Erlang, Part 1: The basics» (em inglês). IBM Developer Works. Consultado em 01 de maio de 2017

Erlang (linguagem de programação). In: Wikipédia: a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/Erlang_(linguagem_de_programa%C3%A7%C3%A3o)> Acesso em: 29 abril 2017.