Universidade do Minho

Mestrado em Engenharia Informática

UCE30: Engenharia de Linguagens

Módulo: Engenharia Gramatical

Exercício para Avaliação n.º1

**Trabalho desenvolvido por:**

Bruno Azevedo

Miguel Cosa

Índice

[Descrição do problema 3](#_Toc310093707)

[Resolução 3](#_Toc310093708)

[Gramática Independente do Contexto 3](#_Toc310093709)

[Gramática de Atributos 3](#_Toc310093710)

[Atributos (A) 4](#_Toc310093711)

[Regra de Cálculo (RC), Condição Contextual (CC) e Regra de Tradução (RT) 4](#_Toc310093712)

[Resolução utilizando o VisualLisa 6](#_Toc310093713)

[Produções 6](#_Toc310093714)

[Lista -> Elementos 7](#_Toc310093715)

[Elementos -> Elemento 7](#_Toc310093716)

[Elementos -> Elemento ‘,’ Elementos 8](#_Toc310093717)

[Elemento -> int 8](#_Toc310093718)

[Elemento -> str 9](#_Toc310093719)

[Regras 10](#_Toc310093720)

[Lista -> Elementos 10](#_Toc310093721)

[Elementos -> Elemento 12](#_Toc310093722)

[Elementos -> Elemento ‘,’ Elementos 13](#_Toc310093723)

[Elemento -> int 15](#_Toc310093724)

[Elemento -> str 18](#_Toc310093725)

# Descrição do problema

Era pretendido que se usasse o processador da Lista de Elementos Mistos (palavras e inteiros), que foi desenvolvido nas aulas, e alterar a sua Gramática de Atributos (GA) de modo a calcular o somatório de cada sequência se inteiros que surjam a seguir à palavra “soma”.

Exemplo:

A frase “[a,1,2,b,soma,3,a,4,soma,b,2,7]”

Dá como resultado: [7,9]

# Resolução

## Gramática Independente do Contexto

Observando o problema formulámos a seguinte Gramática Independente do Contexto (GIC):

**GIC = (T, N, S, P)**

**Símbolos terminais (T):** {str, int, ‘[‘ , ‘]’, ‘,’}

**Símbolos não terminais (N):** {Lista, Elementos, Elemento}

**Símbolo Inicial (S):** Lista

**Produções (P):**

**P0:** Lista -> ‘[‘ Elementos ‘]’

**P1:** Elementos -> Elemento

**P2:** | Elemento ‘,’ Elementos

**P3:** Elemento -> int

**P4:** | str

str = [a-zA-Z]+

int = [0-9]+

## Gramática de Atributos

Depois de definida e analisada a GIC, definimos a Gramática de Atributos (GA) como:

**GA = (GIC, A, RC, CC, RT)**

Para resolver este problema, usamos 3 variáveis:

* Sum
* Sum\_flag
* Result

A variável *sum\_flag* é inicializada a 0 e quando for encontrada a palavra “soma” fica 1 e coloca a variável *sum* a 0, a partir deste momento quando encontrar um elemento inteiro vai adiciona-lo a *sum*.

*Result* é um array que vai conter o resultado, ele é alterado quando se encontrada a palavra “soma” e a variável *sum* é maior que 0, vai ficar: result = result.add(sum).

Os símbolos não terminais podem ter atributos sintetizados e herdados, por isso, a forma que encontramos para resolver o problema de saber quando adicionar ao array *result* o *sum* foi dizer que os símbolos não terminais tem:

* Atributos sintetizados:
  + out\_sum
  + out\_sum\_flag
  + out\_result
* Atributos herdados:
  + in\_sum
  + in\_sum\_flag
  + in\_result

O que é pretendido com esta solução, é que o símbolo não terminal receba a informação do estado atual (atributos in) e depois devolva a informação atualizada (atributos out).

### Atributos (A)

|  |  |
| --- | --- |
| Lista | result : ArrayList<Integer> |
| Elementos | in\_result : ArrayList<Integer>  out\_result : ArrayList<Integer>  in\_sum : int  out\_sum : int  in\_sum\_flag : int  out\_sum\_flag :int |
| Elemento | in\_result : ArrayList<Integer>  out\_result : ArrayList<Integer>  in\_sum : int  out\_sum : int  in\_sum\_flag : int  out\_sum\_flag :int |

### Regra de Cálculo (RC), Condição Contextual (CC) e Regra de Tradução (RT)

P0: Lista -> ‘[‘ Elementos ‘]’

Lista.result = Elementos.result

Elementos.in\_result = new ArrayList<Integer>();

Elementos.in\_sum = 0

Elementos.in\_sum\_flag = 0

P1: Elementos -> Elemento

Elemento.in\_result = Elementos.in\_result

Elemento.in\_sum = Elementos.in\_sum

Elemento.in\_sum\_flag = Elementos.in\_sum\_flag

Elementos.out\_result = Elemento.out\_result

Elementos.out\_sum = Elemento.out\_sum

Elementos.out\_sum\_flag = Elemento.out\_sum\_flag

P2: Elementos0 -> Elemento ‘,’ Elementos1

Elementos0.out\_sum = Elementos1.out\_sum

Elementos0.out\_sum\_flag = Elementos1.out\_sum\_flag

Elementos0.out\_result = Elementos1.out\_result

Elemento.in\_sum = Elementos0.in\_sum

Elemento.in\_sum\_flag = Elementos0.in\_sum\_flag

Elemento.in\_result = Elementos0.in\_result

Elementos1.in\_sum = Elemento.out\_sum

Elementos1.in\_sum\_flag = Elemento.out\_sum\_flag

Elementos1.in\_result = Elemento.out\_result

P3: Elemento -> int

Elemento.out\_result = Elemento.in\_result

Elemento.out\_sum = function refresh\_sum

Elemento.out\_sum\_flag = Elemento.in\_sum\_flag

$1 = Elemento.in\_sum, $2 = Elemento.in\_sum\_flag, $3 = str.value

int refresh\_sum($1,$2,$3){

if($2==1) return $1+$3; else return $1;

}

P4: Elemento -> str

Elemento.out\_result = function refresh\_result

Elemento.out\_sum = function refresh\_sum

Elemento.out\_sum\_flag = function refresh\_sum\_flag

$1 = Elemento.in\_result, $2 = Elemento.in\_sum,

$3 = Elemento.in\_sum\_flag, $4 = str.value

ArrayList<Integer> refresh\_result($1, $2, $3, $4){

if($4.equals(“soma”) && $3 == 1 && $2 > 0)

return $1.add($2); else return $2;

}

$1 = Elemento.in\_sum, $2 = str.value

int refresh\_sum($1,$2){

if($2.equals(“soma”)) return 0; else return $1;

}

$1 = Elemento.in\_sum\_flag, $2 = str.value

int refresh\_sum\_flag($1, $2){

if($2.equals(“soma”)) return 1; else return $1;

}

# Resolução utilizando o VisualLisa

Este problema foi também resolvido visualmente com a ajuda da ferramenta VisualLisa.

## Produções

As **Produções (P):**

**P0:** Lista -> ‘[‘ Elementos ‘]’

**P1:** Elementos -> Elemento

**P2:** | Elemento ‘,’ Elementos

**P3:** Elemento -> int

**P4:** | str

da gramática independente de contexto que já está definida, quando representada visualmente em VisualLisa fica como a Figura 1.

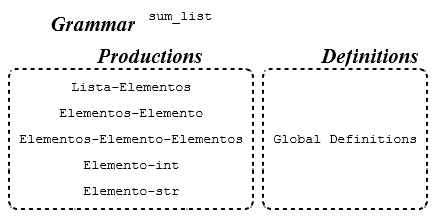


Figura Produções

### Lista -> Elementos

A produção Lista -> Elementos visualmente fica como mostra a Figura 2, em que também já aparecem os atributos de cada símbolo.

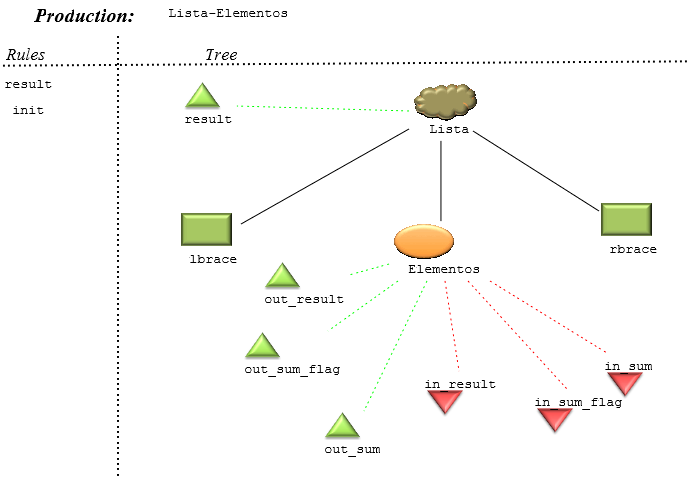


Figura Produção P0

### Elementos -> Elemento

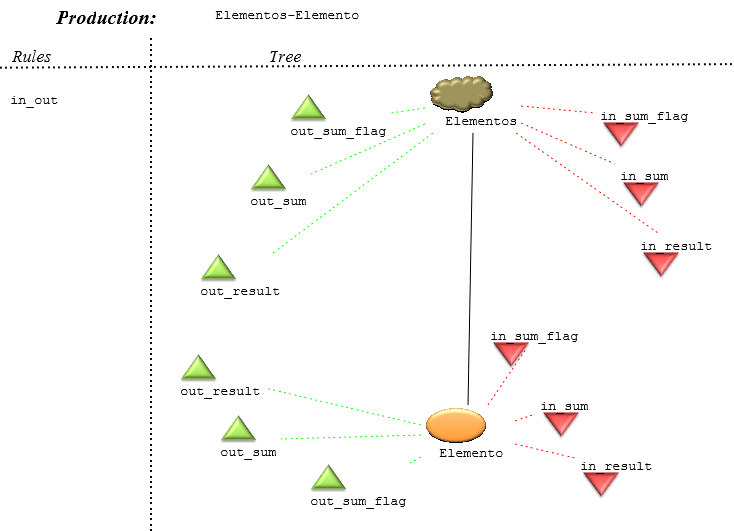


Figura Produção P1

### Elementos -> Elemento ‘,’ Elementos

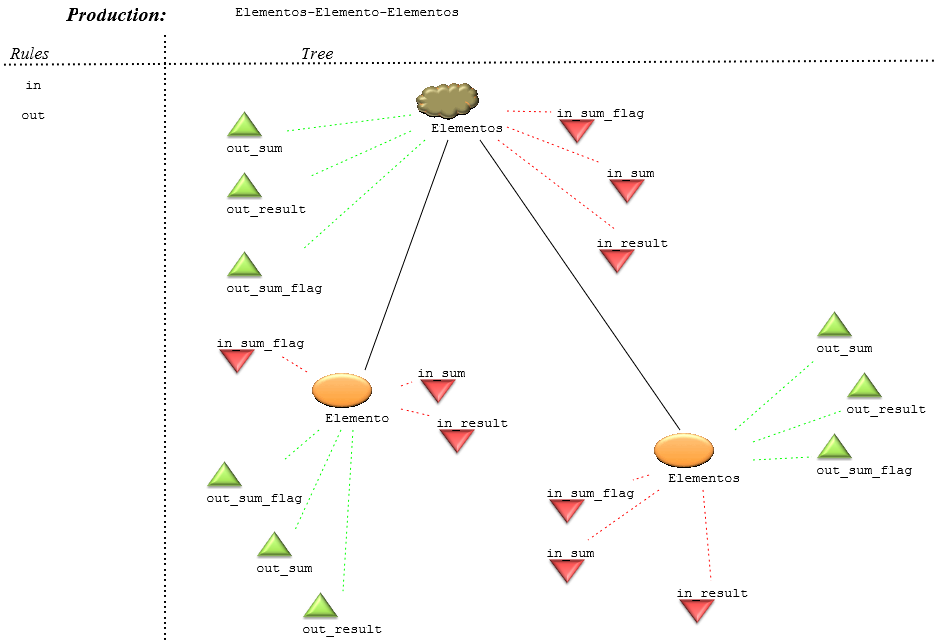


Figura Produção P2

### Elemento -> int

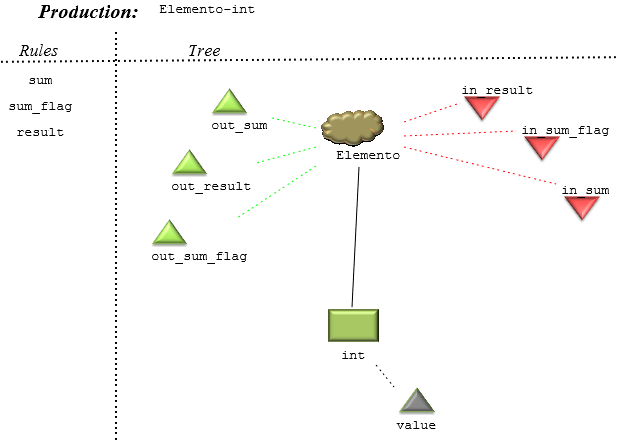


Figura Produção P3

### Elemento -> str

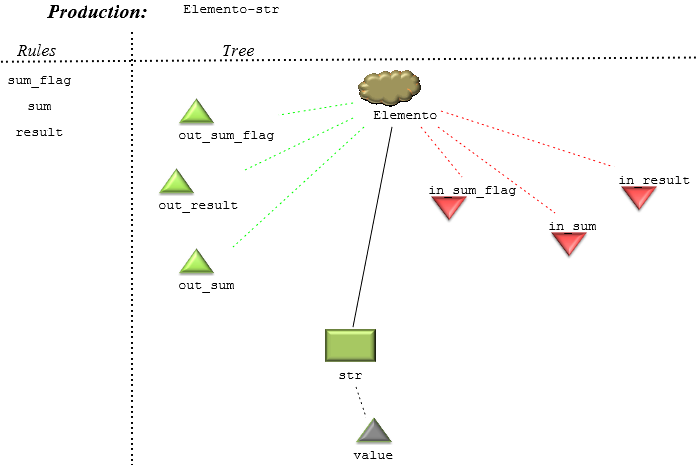


Figura Produção P4

## Regras

### Lista -> Elementos

#### result

Esta é a regra que devolve o resultado da frase que for dada para calcular e é calculada por: Lista.result = Elementos.result

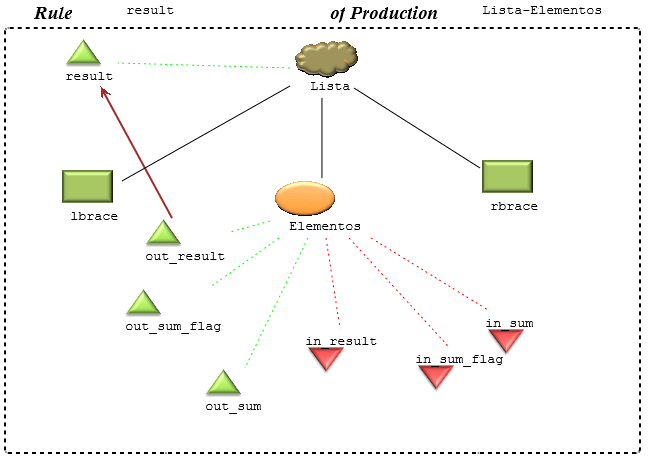


Figura Regra para Lista.result

#### init

O que é feito nesta regra é inicializar as variáveis *in\_sum* e *in\_sum\_flag* a zero.

Elementos.in\_sum = 0

Elementos.in\_sum\_flag = 0

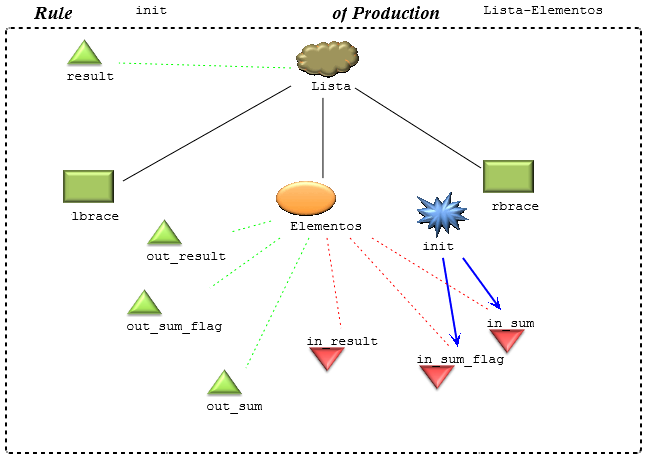


Figura Regra para inicializar variáveis

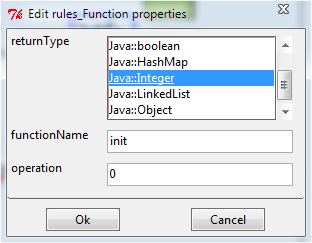


Figura Função init

### Elementos -> Elemento

#### in\_out

Aqui estão as regras:

Elemento.in\_result = Elementos.in\_result

Elemento.in\_sum = Elementos.in\_sum

Elemento.in\_sum\_flag = Elementos.in\_sum\_flag

Elementos.out\_result = Elemento.out\_result

Elementos.out\_sum = Elemento.out\_sum

Elementos.out\_sum\_flag = Elemento.out\_sum\_flag

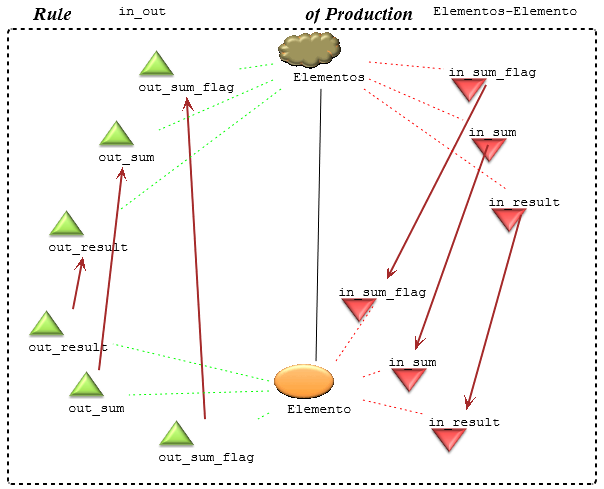


Figura Regras in\_out

### Elementos -> Elemento ‘,’ Elementos

#### In

Regras:

Elemento.in\_sum = Elementos0.in\_sum

Elemento.in\_sum\_flag = Elementos0.in\_sum\_flag

Elemento.in\_result = Elementos0.in\_result

Elementos1.in\_sum = Elemento.out\_sum

Elementos1.in\_sum\_flag = Elemento.out\_sum\_flag

Elementos1.in\_result = Elemento.out\_result

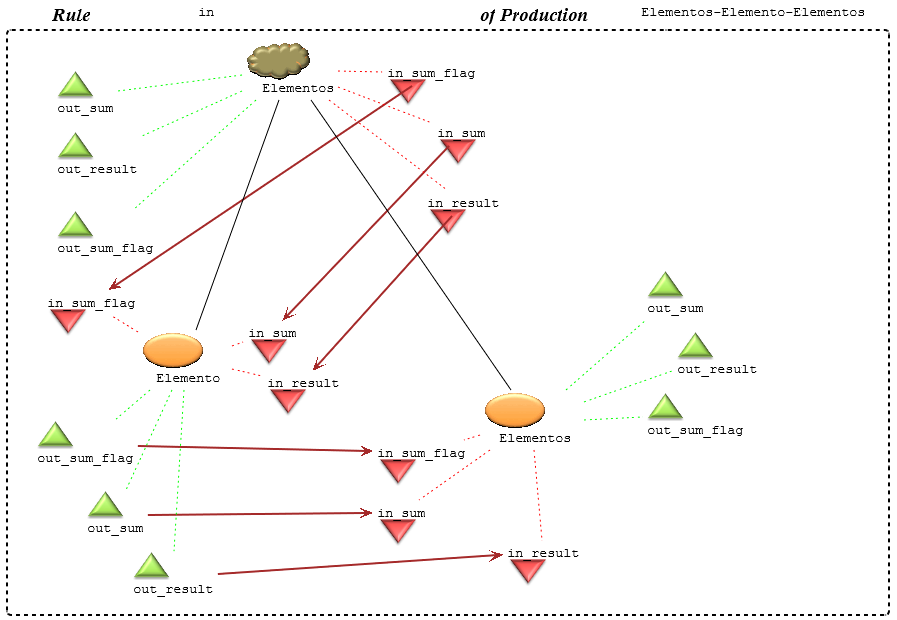


Figura Regras in

#### out

Elementos0.out\_sum = Elementos1.out\_sum

Elementos0.out\_sum\_flag = Elementos1.out\_sum\_flag

Elementos0.out\_result = Elementos1.out\_result

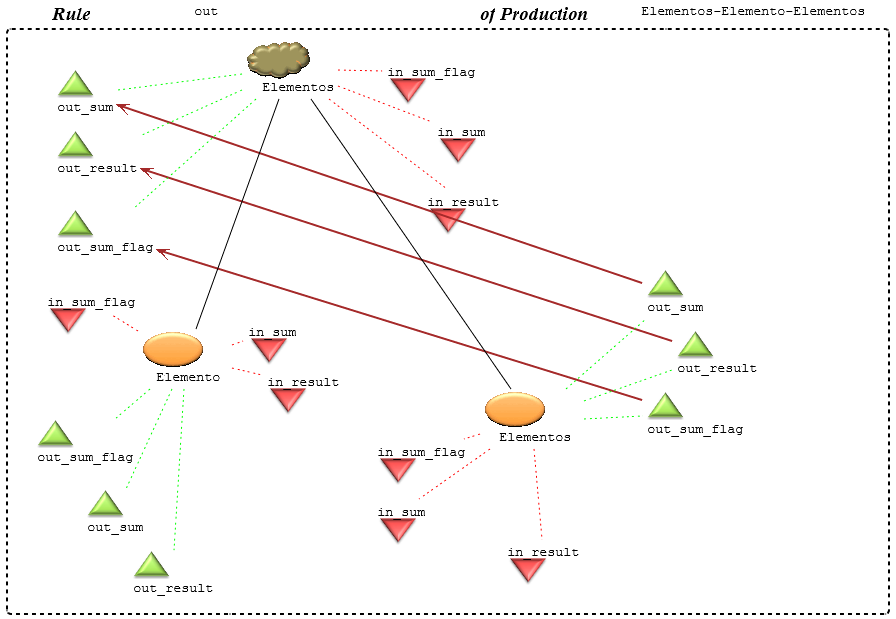


Figura Regras out

### Elemento -> int

#### sum

Elemento.out\_sum = function refresh\_sum

Em que a função é definida por:

$1 = Elemento.in\_sum, $2 = Elemento.in\_sum\_flag, $3 = str.value

int refresh\_sum($1,$2,$3){

if($2==1) return $1+$3; else return $1;

}

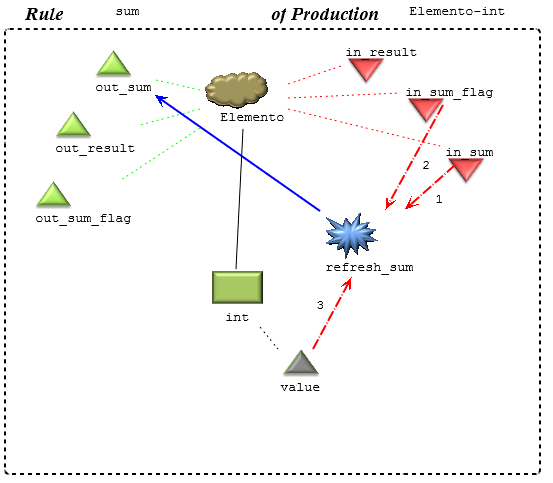


Figura Regra sum

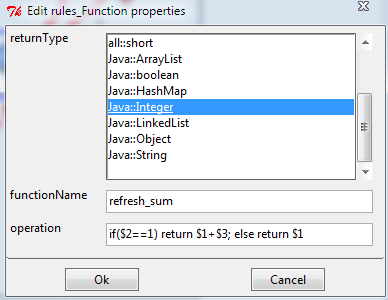


Figura Função refresh\_sum

#### sum\_flag

Elemento.out\_sum\_flag = Elemento.in\_sum\_flag

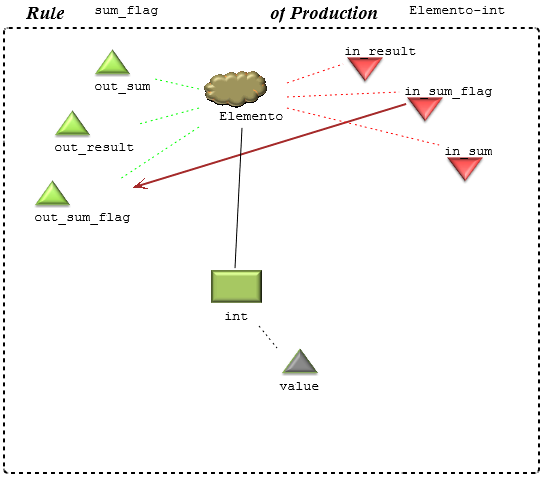


Figura Regra sum\_flag

#### result

Elemento.out\_result = Elemento.in\_result

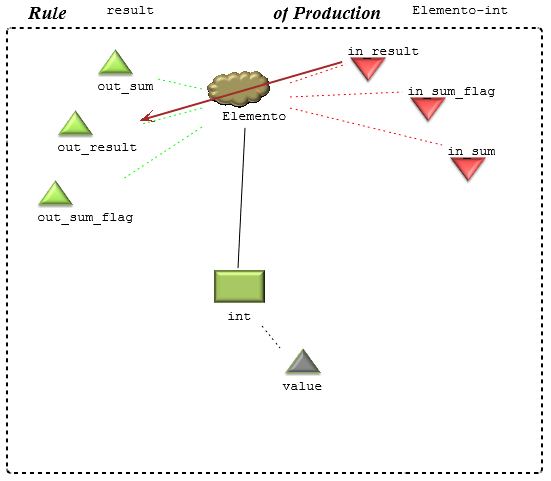


Figura Regra result

### Elemento -> str

#### sum\_flag

Elemento.out\_sum\_flag = function refresh\_sum\_flag

Em que a função é definida por:

$1 = Elemento.in\_sum\_flag, $2 = str.value

int refresh\_sum\_flag($1, $2){

if($2.equals(“soma”)) return 1; else return $1;

}

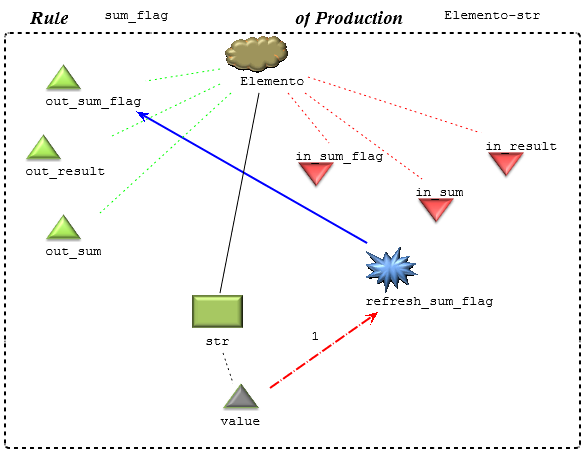


Figura Regra sum\_flag

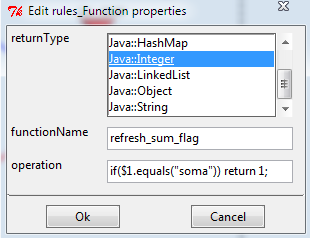


Figura Função refresh\_sum\_flag

#### sum

Elemento.out\_sum = function refresh\_sum

Em que a função é definida por:

$1 = Elemento.in\_sum, $2 = str.value

int refresh\_sum($1,$2){

if($2.equals(“soma”)) return 0; else return $1;

}

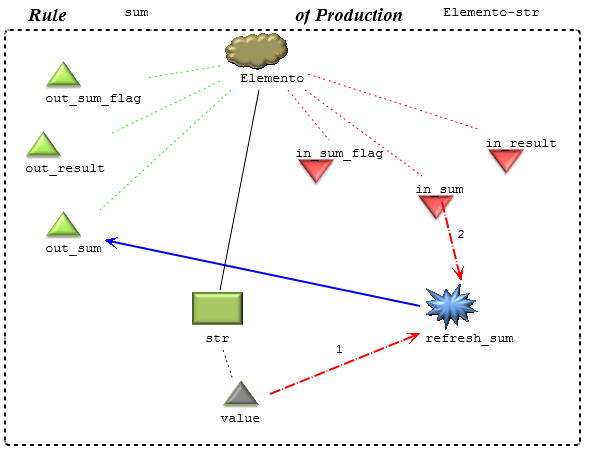


Figura Regra sum

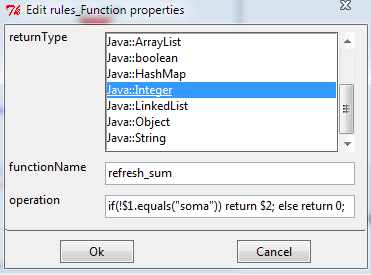


Figura Função refresh\_sum

#### result

Elemento.out\_result = function refresh\_result

Em que a função é definida por:

$1 = Elemento.in\_result, $2 = Elemento.in\_sum,

$3 = Elemento.in\_sum\_flag, $4 = str.value

ArrayList<Integer> refresh\_result($1, $2, $3, $4){

if($4.equals(“soma”) && $3 == 1 && $2 > 0)

return $1.add($2); else return $2;

}

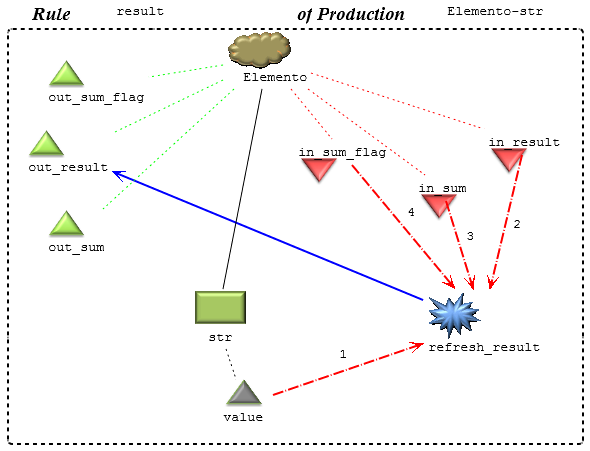


Figura Regra result

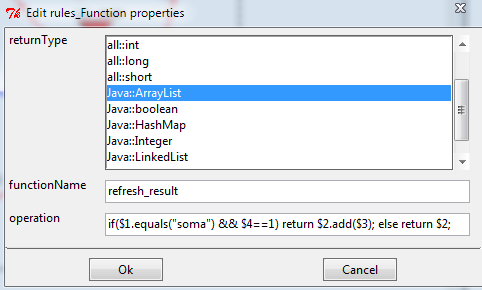


Figura Função refresh\_result