Evidencia Final Compiladores: Desarrollo de herramienta de soporte al proceso de análisis de imágenes

A01422874 Salvador Salgado Normandia

Implementación de flujos que sean agregaciones: b = img -> blur(12) -> mean() como último paso. Estas funciones son parte de numpy

Para este caso fue necesario retomar el concepto de agregadores vistos en clases. De manera simple, el termino agregación se puede definir como la acumulación de operaciones de un flujo, las cuales son recibidas por las siguientes funciones. Se puede decir que en casos donde cuentas con varias funciones dentro de otras como parámetros, con el uso de agregaciones solo se está modificando inicialmente el elemento del flujo incial, y consecuentemente modifica al resto de funciones. De igual manera, al momento de querer visualizar el proceso de como una variable esta siendo modificada, el uso de agregaciones permite identificar todos los cambios aplicados.

En este caso, al contar con procesamiento de imágenes en formato matrices, siempre el primer parámetro de cualquiera de las funciones establecidas por opencv reciben como primer parámetro a la matriz.

Para el procesamiento del flujo, siempre se inicializará identificando el último elemento del flujo, en donde contará con una lista de sus parámetros “nodos”. En esta lista de parámetros se considera un parámetro pending, el cual contiene el resultado o salida del elemento anterior en el flujo. Este proceso sigue hasta llegar a la primera variable del flujo.

En el ejemplo dado para el ejercicio *b = img -> blur(12) -> mean(),* primero se estaría procesando la función mean, la cual a pesar de no parecer que cuente con parámetros, este cuenta con un pending node la cual una vez que se haya alcanzado a procesar img, esta regresara a mean como el parámetro pending que equivaldría a una matriz de imagen.

En este caso se tuvo que considera que en nuestra implementación de flujo, existiera la posibilidad de identificar en nuestra gramatrica el llamado de funciones sin parámetros, ya que en caso de no estar establecida marca un error al momento de llegar a mean() y no identificar ningún parámetro.

```
def p_flow_function_call_no_params(p):
    '''
    flow_function_call : VARIABLE LPAREN RPAREN
    '''
    node = add_node({"type": "FLOW_FUNCTION_CALL",
                     "label": f"ff_{p[1]}", "value": p[1]})
    pending_node = add_node(
        {"type": "PENDING_NODE", "label": "pending", "value": ""})
    parseGraph.add_edge(node["counter"], pending_node["counter"])
    p[0] = node
```

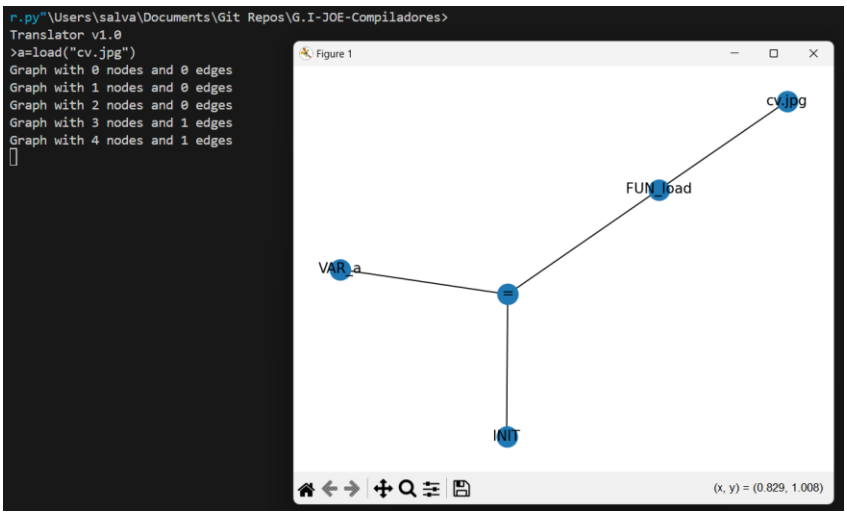
Una vez realizada esta configuración, el resto de flujo previamente realizado durante la clase permite ir identificando todas las funciones “uso de counter” y sus parámetros (dentro de listas), para que una vez alcancemos a la variable al inicio, los parámetros pending se irán resolviendo hasta volver al mean y entregarle el pending node como parámetro.

Ademas de esta modificación se agregó la función de mean y blur. Para el caos de blur este recibe tanto imagen como una tupla, pero en nuestro caso se decidió que solo recibiera una variable n y dentro de la función crear la tupla necesaria. Esto tomando en cuenta que la imagen será obtenida por medio del parámetro pending.

```
def mean(res):
    return np.mean(res)


def blur(image, num):
    size = (num, num)
    return cv2.blur(image, size)
```

Finalmente, se ejecuta el programa y se ejecutan las pruebas necesarias:

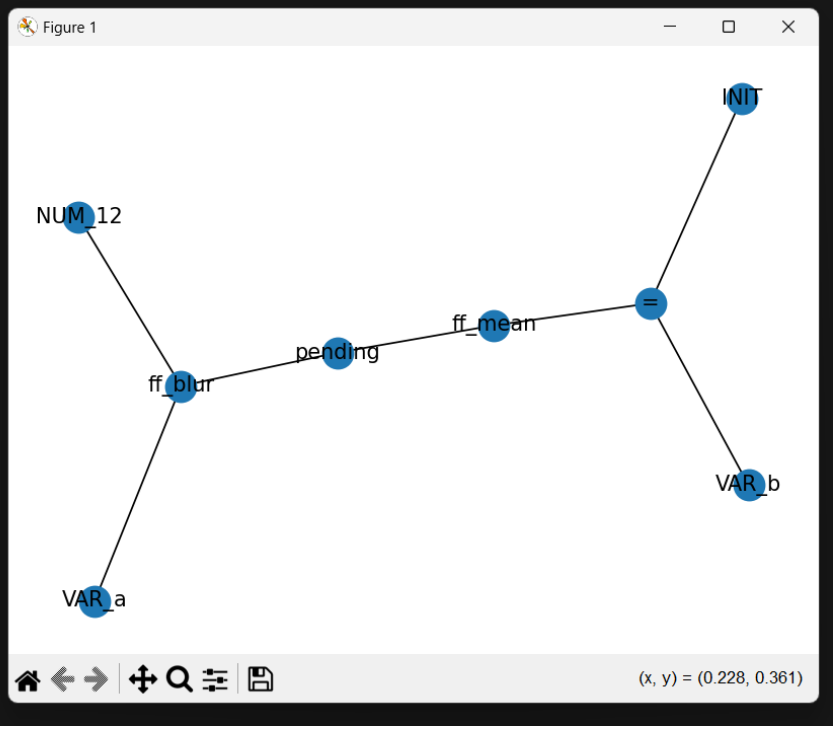
1) Crear variable “a” en donde se llama a función load() para almacenar una imagen.



```
Graph with 4 nodes and 1 edges
Result [[[ 6 14 44]
  [ 7 11 39]
  [11 12 32]
  ...
  [21 42 63]
  [21 40 61]
  [19 38 59]]

 [[ 9 18 45]
  [11 17 40]
  [14 17 32]
  ...
  [31 53 78]
  [30 50 75]
  [29 49 74]]
```

2) Ahora que ya contamos con la variable que ira al inicio del flujo, podemos ejecutar el flujo completo, en donde pasará la variable a como parámetro “pending/oculto” a blur(12) y este resultado/imagen se pasará a la función de mean como parámetro oculto. Debido a la modificación hecha para el llamado de función sin parámetros, mean() va a ser identificado de manera correcta e iniciara el análisis de funciones y parámetros hasta llegar a variable y obtener su resultado.

```
  [21 28 47]
  [19 26 45]
  [17 24 43]]

 [[25 38 60]
  [26 39 61]
  [27 40 62]
  ...
  [22 29 48]
  [20 27 46]
  [18 25 44]]

 [[25 38 60]
  [25 38 60]
  [24 37 59]
  ...
  [23 30 49]
  [21 28 47]
  [19 26 45]]]
>b=a->blur(12)->mean()
Graph with 0 nodes and 0 edges
Graph with 1 nodes and 0 edges
Graph with 2 nodes and 0 edges
Graph with 3 nodes and 0 edges
Graph with 4 nodes and 2 edges
Graph with 5 nodes and 2 edges
Graph with 6 nodes and 4 edges
Graph with 7 nodes and 4 edges
```



```
>b=a->blur(12)->mean()
Graph with 0 nodes and 0 edges
Graph with 1 nodes and 0 edges
Graph with 2 nodes and 0 edges
Graph with 3 nodes and 0 edges
Graph with 4 nodes and 2 edges
Graph with 5 nodes and 2 edges
Graph with 6 nodes and 4 edges
Graph with 7 nodes and 4 edges
Result 113.89563093256133
```