ASSIGNMENT 3: KACHUA MOVEMENT OPTIMIZATION

KACHUA VERSION : 2.0.1

EXECUTION FLOW OF THE TOOL:

- User inputs the turtle file path as command line argument to the ***Kachua.py***, file is then opened and parsed to make the ***Intermediate Representation (IR).***
- ***IR*** is passed to the ***optimize()*** in the ***Submission.py*** by ***Kachua.py*** to optimize Kachua movements.
- **optimize()** first optimizes each statement block of CFG independently using **blockoptimize()** .
- **blockoptimize()** builds a CFG using IR and optimizes each block of the CFG individually and generates a new optimized IR and returns it to **optimize()** as **ir2**.
- **optimize()** takes the new IR from **blockoptimize()**, generates a new CFG for it and then calculates nodes with more than **1 predecessor** or **successor.**
- Using above generated information about nodes, **facts** set for each required node is generated and used for optimization
- First node(s) with multiple successors are optimized followed by node(s) with multiple predecessor are optimized.
- At the end optimized IR is displayed on terminal and passed back to **kachua.py** to be stored in **optimized.kw** file.

STEPS TO RUN THE TOOL:

1. Check if python packages like ***graphviz, kturtle*** and other packages and dependencies mentioned in ***Readme.md*** are installed properly, if not you can install it by running command "***pip <space> install <space> <package name>***" in any terminal or appropriate commands from ***Readme.md***
2. Open any terminal (***Command Prompt, Windows PowerShell, Git Bash***, etc.) and move into the directory where **k*achua.py*** is present
3. Type " ***./kachua.py <space> -O <space> file path/turtle file name.tl***" to run optimizer
   For eg, to run one of the testcases provided type: "***./kachua.py <space> -O <space> ../Submission/testcases/test2.tl"***
4. Press Enter
5. ***Optimized IR*** *will be displayed in the terminal*
6. *Optimized IR is stored as* ***optimized.kw*** *in KachuaCore folder*
7. *To run optimized.kw type '****./kachua.py <space> -b <space> -r <space> ./ optimized.kw****'*

HARSH AGARWAL
21111030
MTECH CSE 2021-23
PROGRAM ANALYSIS, VERIFICATION AND TESTING

IMPLEMENTATION LOGIC:

- **blockoptimize():**
  - CFG for passed IR is generated and Termination node is removed
  - Dictionary is defined to store old index and new index as key, value pair
  - For each node in the CFG, an assignment instruction ‘:optidist = 0’ is appended to IR
  - For each statement in that node, statement type is checked, if

    - **Forward MoveCommand**,
      - checks whether previous optimization done was not of backward movement.
        - If not
          - and total rotation is not 0, then if required, a forward / backward command is appended and a rotation instruction is appended to IR, optidist is set to 0
          - move value is added to ‘optidist’ and appropriate variable values are updated,
        - If yes, then a backward command is appended to IR
          - if total rotation is not zero then rotation command is appended.
          - optidist is set to 0 and value of current forward is then stored and appropriate variables are updated.
      - New index and old index are stored in dictionary for later use to calculate jump of conditional statements.

    - **Backward MoveCommand**
      - Similar method is followed for this command as followed above for Forward MoveCommand

    - **Right MoveCommand**
      - Angle of rotation is added to total rotation and remainder after division by 360 is stored as total rotation.
      - If the command is target of a jump, then new index is taken as index of previous ‘:optidist = 0’ instruction

    - **Left MoveCommand**
      - Angle of rotation is subtracted from total rotation and remainder after division by 360 is stored as total rotation.
      - If the command is target of a jump, then new index is taken as index of previous ‘:optidist = 0’ instruction

        - **PenCommand**
            - If the total rotation angle is not zero then a rotation command is appended.
            - If a forward/backward instruction is pending then that is appended and variables are updated.
            - PenCommand is appended and old and new index are stored
            - ‘:optidist = 0’ is appended.

        - **ConditionCommand**
            - If the total rotation angle is not zero then a rotation command is appended.
            - If a forward/backward instruction is pending that that is appended and variables are updated.
            - ConditionCommand is appended with relative jump as 0 and old and new index are stored.

        - **Any other Command**
            - Append the command and old and new index are stored.

    - Before moving to next node:
        - If a forward/backward statement is left to be appended, then it is appended and old and new index are stored
        - If total rotation is not 0, then a rotation command is appended and old and new index are stored.
    - Jump value of all conditional instructions is updated using old and new index values
    - At the end, new IR generated is returned to optimize()

- **predecessor() / successor()**:
    - Predecessor(s) / Successor(s) of each node is identified, except for termination node
    - Information of only those nodes are stored which have more than one predecessor / successor
    - Information in form of dictionary is returned to optimize()

- **calcfacts():**
    - For the nodes passed as parameter to the function, following information is found out :
    -
        - Head :
            - Finds out where first forward/backward instruction is present in the node, that index and type (forward or backward) is stored.
            - Finds and stores index of all ‘:optidist = 0’ preceding the first forward/backward instructions.

        - Finds and stores rotation between first instruction in node and first forward/backward instruction in that node and index of the rotation instruction found.

    - Tail :
        - Finds out where last forward/backward instruction is present in the node , that index and type (forward or backward) is stored.
        - Finds and stores rotation between last forward/backward instruction in node and last instruction in that node and index of the rotation instruction found.
    - Information found is returned to optimize() as dictionary with key as node id and value as info found.

- **optimize():**
    - First blockoptimize() is called to perform block level optimization
    - Using the IR returned by blockoptimize(), a CFG is generated.
    - CFG is passed to predecessor() and successor() to calculate predecessor(s) and successor(s) of each nodes respectively.
    - In similar sense as forward analysis, first nodes with more than one successor are optimized.
    - Node and its list of successors are passed to calcfacts() to find out the movement of kachua at the end of predecessor block and at the beginning of each successor block. If they meet all conditions defined, then both predecessor block and successor blocks are optimized.
    - Then in similar sense as backward analysis, next nodes with more than one predecessor are optimized.
    - Node and list of its predecessor are passed to calcfacts() to find out the movement of kachua at the beginning of successor block and at the end of each predecessor block. If they meet all conditions, then both successor block and predecessor blocks are optimized.
    - Optimized IR at the end of all three optimization is printed in the terminal and returned to kachua.py

LIMITATIONS & ASSUMPTIONS:

- Input program does not contain ‘:optidist’ as an identifier
- Left and Right movement commands should have implicit numeric values. For example : right 90 or left -45 are allowed but right :vara or left 360/:varb are not allowed.
- Forward and Backward move commands should have positive move values only.
- Penup and Pendown statement blocks are optimized indifferently, that is there is no extra optimization done when penup command is found.
- Repeat statement block is optimized but loop in-variance is not evaluated.

HARSH AGARWAL
21111030
MTECH CSE 2021-23