Team Members: Aden Tessman, Jonah Whites

Calca.rkt Technical Manual

**Overall Description:**

The Calca program works by calling a single function called “evaluate-expression” from this function the other functions are called in order to properly derive, simplify, and evaluate the given expressions. The two inline operators are calculated using match case statements, which are made recursive in a stand alone function.

**Match Case Statements:**

Match Case statements are used to efficiently determine and evaluate when “and” and “or” operators are present. These work by checking if the corresponding symbols ‘&’ or ‘V’ are the second element in the list. If they are the first and second elements are evaluated. The value found after evaluation is then attached to the head and tail of the list, with the other elements left in the middle, this is done so that the evaluated value is directly accessible and so that iter can check if an evaluation has occurred. If nothing is evaluated the same list is simply returned unchanged.

**Functions:**

**iter:** This function is used to make the match case statements recursive. It does this by calling a given function on each element of the list, iterating through the entire list unless an operator is found. If one is found and an evaluation happens the list will then be returned.

**de-list:** This function is used to remove sublists from the initial expression. It searches through element by element looking for sublists, if one is found Operate is called to evaluate the sublist, and the iteration continues. Additionally if a single element sublist is found it is converted to an and expression to “vocalize it” or in other words ensure that it is present in the output of evaluated elements.

**operate:** This function used to evaluate expressions. It uses member functions to determine if ‘&’ and ‘V’ are present if one is the corresponding operator is called with iter to evaluate this instance. Operate calls ‘&’ first and does this until all ‘&’s are gone creating operator precedence; it then does this for ‘V’ as well. Not’s evaluation is also done in operate, this is done by checking if a list exists beginning with ‘~’.

**extra:**  This function exists to do an additional check for single element sublists. Even though a similar check is done in de-list this function is also necessary to ensure any single element sublists that might have been made during the evaluation process are also properly evaluated without any other effects that would be brought on by a de-list call.

**Function Order:**

In evaluate-expression de-list, extra, and operate are called in that order. De-list is initially called to deal with any initial sublists. Next Extra is called to ensure the evaluated sublists are converted from single elements to “vocalized” expressions. Additionally before a final operate call the built in function flatten is applied to ensure all sublists are removed and any improper lists are corrected. Finally, now that the given expression has been converted entirely to a flat expression operate is called and the output is made.

**Output Example:**

(evaluate-expression '((~ #t) V (~ #f) & #t))

1. (de-list '((~ #t) V (~ #f) & #t))

'((#f) V (#t) & #t)

1. (extra '((#f) V (#t) & #t))

'((#f) & (#f) V (#t) & (#t) & #t)

1. (flatten '((#f) & (#f) V (#t) & (#t) & #t))

'(#f & #f V #t & #t & #t)

1. (operate '(#f & #f V #t & #t & #t))

'(#f #t #t #t)