Complementary Material for Relational Conditional Set Operations

Alexis I. Aspauza Lescano[1[0000−0001−8685−010X]] and Robson L. F. Cordeiro[1[0000−0002−6795−3004]]

University of São Paulo – USP, Av. Trab. São Carlense 400, São Carlos, SP, Brazil

This is a complementary document related to the paper: "Relational Conditional Set Operations" submitted to ADBIS 2021. Here, we divide this document into two parts: (1) the algorithm to convert an infix expression to postfix, and (2) the extended relational conditional set operations which support index structures either in left or right relation.

1 Infix To Postifx Expression

We present in this section the Algorithm 1, which converts an infix expression to a postfix expression. We can see some examples of these conversions in Table 1.

Table 1: Examples of Infix and Posfix equivalences.

Infix Expression	Postfix Expression
DP.Category = SP.Category	DP.Category SP.Category =
DP.Category = SP.Category AND DP.Product = SP.Product	DP.Category SP.Category = DP.Product SP.Product = AND
DP.Category = SP.Category AND DP.Product = SP.Product AND (DP.Units ≤ SP.Units AND ¬ (DP.Price < SP.Price) OR DP.Price / 2 ≥ SP.Price)	DP.Category SP.Category = DP.Product SP.Product = AND DP.Units SP.Units ≤ DP.Price SP.Price < ¬ AND DP.Price SP.Price 0.5 * ≥ OR AND

Starting the algorithm, in lines 1 and 2, we need to create a stack of pending operations and a vector to contain the postfix expression to be returned. Also, in the initialization, line 3 pushes an open bracket "(" onto the stack of pending operations, as well as line 4, adds the closing bracket ")" to the end of our input expression. Lines 5 to 23 are the main loop, which will get every single token extracted from the predicate (c). The current token being read will be processed according to its type. If the token is an operand (any constant or reference to a column table), it will be added directly to the postfix expression vector. In case the token is an opening bracket "(", it will be added to the stack of pending

Algorithm 1 ToPostfix(c)

```
    Input: c: Predicate in infix notation
    Output: postfixExp: Expression in postfix notation
create stack pendingOps
create vector postfixExp
push "(" onto pendingOps
add ")" to the end of c
for each token extracted from c do
    if token is an operand then
        add token to postfixExp
    else if token is "(" then
        add token to pendingOps
    else if token is ")" then
        lastToken = pop from pendingOps
        while lastToken != "(" do
            add lastToken to postfixExp
            lastToken = pop from pendingOps
        end while
    else if token is an operator then
        while top of pendingOps is an operator and it has equal or higher prece-
dence than token do
            lastToken = pop from pendingOps
            add lastToken to postfixExp
        end while
        add token to pendingOps
    end if
end for
return postfixExp
```

operations. But if the token is a closing bracket ")", we will pop tokens from the pending operations and add them to the postfix expression until an opening bracket is found, and the bracket will just be discarded. As the last option, if the token is an operator of any type (logical negation, arithmetic operator, logical operator, or logical connector), we will add to the postfix expression all top operators from the pending operations while the top is an operator with higher or equal precedence than the current token; and then, add the token to the stack of pending operations. The next operators are ordered by higher to lower precedence:

1. $\neg$
2. *, / both with same precedence
3. +, - both with same precedence
4. $<, \leq, >, \geq, =, \neq$ both with same precedence
5. $\wedge$
6. $\vee$

Finally, we will return our postfix expression vector.

2 Relational Conditional Set Operations

In the paper, we assumed the index structures are always in the right relation. Here, we present an algorithm that is capable of execute the RelCond Set Operations if there are indexes either for the left or for the right relation.

Algorithm 2 RelCondSetOp(${}_c\mathsf{T}_1$, ${}_c\mathsf{T}_2$, c, $SetOp$)

Input: ${}_c\mathsf{T}_1$: Left RelCond Set, ${}_c\mathsf{T}_2$: Right RelCond Set, c: Predicate, $SetOp$
Output: ${}_c\mathsf{T}_R$: All tuples that are in the result

Create a vector of bits R with size of $|{}_c\mathsf{T}_1|$
if $SetOp$ is $\cap_c$ **then**
    Initialize R with 0's
else if $SetOp$ is $-_c$ **then**
    Initialize R with 1's
end if
if ${}_c\mathsf{T}_2$ is the table with index structures **then**
    for each tuple t_i of ${}_c\mathsf{T}_1$ **do**
        if IsCondMember(${}_c\mathsf{T}_2$, t_i, c) **then**
            if $SetOp$ is $\cap_c$ **then**
                set $R[i]$ as True
            else if $SetOp$ is $-_c$ **then**
                set $R[i]$ as False
            end if
        end if
    end for
else if ${}_c\mathsf{T}_1$ is the table with index structures **then**
    for each tuple t_j in ${}_c\mathsf{T}_2$ **do**
        S = Index_TupleQuery(${}_c\mathsf{T}_1$, t_j, c)
        if $SetOp$ is $\cap_c$ **then**
            $R = R \vee S$
        else if $SetOp$ is $-_c$ **then**
            $R = R \wedge \neg S$
        end if
    end for
end if
${}_c\mathsf{T}_R$ = get tuples for all true bits of R
return ${}_c\mathsf{T}_R$