University at Albany, SUNY
College of Engineering and Applied Sciences
Computer Science Department

Implementation of a saturation procedure for propositional clauses

(Master's Project)

Raswitha Narla

Advisor: Paliath Narendran

Abstract

I implemented an algorithm for this project in which it performs saturation of an expression which is given in the string format.We will be doing two methods in the programming 1) Resolution method and 2) Subsumption method.In resolution method we check whether the individual strings can be reduced to one string and in the subsumption method we check whether the extracted result from both strings is subset of them or not. In this way we reduce an given combined string and this method is known as saturation. I implemented this algorithm using Python3 language .

Acknowledgements

I would like to thank my advisor and mentor Prof. Paliath Narendran for giving me an opportunity to take up this project under his guidance and providing immense help in every way possible.I would also like to thank all the people whose research work helped me immensely in understanding all the concepts. Last, but not the least, I would like to thank my friends and family for supporting me and strongly encouraging me in all of my dreams, especially academically.

Implementation of a saturation procedure for propositional clauses

1 Introduction

Resolution is logic form consisting of a function which takes premises, analyzes their syntax, and returns a conclusion.It is is applied to all possible pairs of clauses that contain complementary literals. After each application of the resolution rule, the resulting sentence is simplified by removing repeated literals. If the sentence contains complementary literals, it is not considered because of the concept tautology. If after applying a resolution rule the empty clause is derived, the original formula is contradictory and therefore it can be concluded that the initial conjecture follows from the axioms.

The subsumption theorem is an important theorem concerning resolution. The defination of sumbsumption can be written as: Let C and D be clauses. We say D subsumes (or θ-subsumes) C if there exists a substitution θ such that D$\theta \subseteq$C.
Let $\sum$ be a set of clauses and C a clause. We say there exists a deduction of C from $\sum$, written as $\sum \vdash_d$C, if C is a tautology, or if there exists a clause D such that $\sum \vdash_r$ D and D subsumes.

The combination of both the Resolution and the Subsumption property is known as the saturation part. Saturation is done by checking all the string combinations for the resolution property and then checking with the subsumption property to get the final output.

2 Definition and Problem Statement

A string S is defined by giving the set of individual strings which can also be converted to clauses. In the below sections an algorithm and the approach followed was proposed where a input string will go through it and will be saturated until the strings cannot be resolved or subsumed.Our goal is to practically implement this algorithm in python3 language.

3 Algorithm

An algorithm is the set of steps which describe our complete program. Initially let us write the input and output for our algorithm.
Input: String is our input in which we can also have the combination of the small strings.
Output: A string which is saturated by resolution and subsumption methods in our algorithm.

1. Initially add two strings and we will get the resultant string
   S_1 + S_2 = resultant string
2. Now check the resultant string with the other strings in the same format whether they can be

resolved or not.
3. The above step is repeated
4. After getting the final resultant combination of the strings then we need to send it to the subsumption function.
5. Subsumption function is called and will return the answer string.
6. The string returned will be sent to the main function and then we will remove the extra literals in that string and print the output val.

The entire procedure runs in the form as shown below:
Let U, P be individual of strings. If the given set of strings is S = $S_1, S_2, ..., S_n$ then set
U: = S_1 and P := S_2

The procedure goes like:

Resolve the string in U with the string in P if it is possible then add the resulting string to the end of the input, if not put them as it as without doing any changes.

In the similar fashion check all the possibility combination of the strings and add them to the input and repeat the process.

Stop the process when there is no combination of the strings to resolve and store and print the output.

4 Implementation

I have braken down the implementation in 4 parts:

I.A program that will read a string and create the initial level strings which will be splited using the method such as they are splited where we have ”+”. symbol.This level of strings will be stored in a $main_list$.

1.In this function main() we will be call both the func() and the subsumption function.
2.The choice of the data structure which I used in the program is a dictionary in which we have the key - value pair.In our case the keys are strings which is our input splited into the individual strings and values are how many strings are present.
3.As doing the saturation the initial step is to check whether there re any set of the strings that can be resolved.
4.func(ipt) - The parameter for the func() is the ipt which is the input string
5.Example ipt = anb+bc (which is considered as ab’ + bc).
6.We use a for loop to iterate each string in the given input and the we will create an dictionary to

calculate the number of strings and store how many times the string is repeated.
7.In the main next we will separate the strings and split them using the ”+” symbol.
8.Next after these program lines we will call the subsumption function the output from the that function will be the final one and then we need to remove the extra literals from the output and then print it.
The usage of the while loop in the main function we did is because we used an dynamic way of the input and after giving the one input we will be displayed by the statement in the output terminal as Continue(1/0) so the while loop is used to take the multiple inputs.So, if you give 1 then you can enter the other input and then the entire process runs again and the output will be displayed or if you enter 0 then the program will exit.

II.A program in which the strings are converted into the lists and emerge them so that to check the resolution property.

1.This is the stage where we will discuss what happens in the func(ipt).
2.In the program part we will call the canBeReduced(list,list).
3.In this we will return some value from the canBeReduced(list,list) and store in the result so before going to this reduced function we will split the string into the individual characters and store them in the list.
4.In this the conversation list will be if suppose we have ab+ncd the main list will be represented as [’a’,’b’ , ’nc’,’d’]. Basically when we have n in the input then the side character of n is considered to be combined with n since we will take it as the contradication of that character.
5.So in this function we will have mail importance to the list converstion and send the individual lists to the canBeReduced(list,list) as the parameters and check whether we can resolve both the lists or not to get the resulting list.

III.A program that resolves to set of lists which we will give as the parameters to the function, I will be using the function canBeReduced(list,list).

1.This is an important function in our entire program where we will be checking whether our both the lists which we give as an input to this function can be resolved or not.
2. Resolved meaning in our program = if we have na+a then we can reduce it to 1.
3.So, the main condition in our program is if we have two variables with the negation values in the other string then we should not perform the reduced operation.Let us consider an example and see how it looks:
If the input is ab + nanb (i.e ab + a’b’) since in this we have the contradiction property followed for both the variables a and b then we cannot reduce this combination of the strings.
In this way by using the for loop iteration we will check the resolution property for the combinations of the lists we will generate through the main list.
The function will return the value and the reduced form of the strings which will be added to the end of the input.

IV.A program with the final input generated by the canBeReduced() function will check for the

subsumption property and if subsumed by the new resultants then they are removed. I will return the new combination of the strings after the subsumption.

1.In this function the parameter will be the main list which we will be generating by the above step function and then the subsumption property is checked.
2. So let us discuss what is an subsumption property - suppose when we combined two strings we got ”cd” as the resultant string now we need to check whether ”cd” is the subset of all the individual strings in the input. If it is an subset of any of the strings then that string need to be removed from the input.
In this way we will check the subsumption property and then we will remove the strings and return the value.

5 Conclusion

We have implemented the algorithm, the usage of the dictionaries is part of the program efficiency. In this I have used different set of the data structures such as sets, lists, dictionaries. My approach is the recursive approach in which the function is called if the specific conditions satisfy which are given in the program. Implementing this program is challenging because we need to check for the every combination of the strings in the given entire big set of strings for the resolution and add to the end of the input and send that to the subsumption function which will return the final output string.

6 References

7 Appendix

7.1 Instructions

The below code starts the execution from the command __name__ ==’__main__’:

.
In our input string we can have alphabets(a to z) and negation of alphabets which are taken in the form(na , nb ..... nz)
When you run the program on the output screen we will have: ”Enter input = ” statement.
Example: Enter input = anb + bc (which is equal to ab’+bc)
After giving the input it will go through the entire functions in the program and the output will be printed as ”val = ”.
Next in the output terminal it ask ”Continue 0/1: ” that means if we want to give the another input

then we need to give 1 and if we want to exit the program we need to give 0.
Let us consider one example ad see how it executes and prints the output. Let the input be ”anb + bcd + nde”

Example 1: (Input = anb + bcd + nde)
The below are the execution steps:
anb + bcd $->$ acd (nb + b = 1 resolution is done)
anb + bcd + nde + acd (added to the input)
nde + acd $->$ eac (check for the resolution and done)
anb + bcd + nde + acd + eac (again add to the input)
subsumption(anb + bcd + nde + acd + eac) $->$ anb + bcd + nde + acd + eac (checking whether it can be subsumed or not and it cannot be subsumed)
anb + bcd + nde + acd + eac will be the final output.

Let us consider an another example where the output is subsumed:
Example 2: (Input = nabnc + nabc)
The below are the execution steps:
nabnc + nabc $->$ nab (nc + c = 1 resolution is done)
nabnc + nabc + nab (added to the input)
No more resolution can be done
subsumption(nabnc + nabc + nab) $->$ nab (checking whether it can be subsumed more or not and it can be subsumed because nab is subset of both nabnc and nabc so in the final output no need of those two individual strings)
nab will be the final output.

7.2 Python Code

```
''' This is subsumption function where we will check if the resultant output
is the subset of the input strings'''
def subsumption(main_list):
    length1 = len(main_list)
    #declare a z empty dictionary and u empty string
    z={}
    u = ""
    # store individual elements of the strings in the form of the list
    for i in range(len(main_list)):
        z[i]=[]
    # This loop iterates over the individual elements of the list
    for i in range(len(main_list)):

         for j in range(len(main_list)):
            # checkthe resultant input size less than individual string size
```

```
            if len(main_list[i])<=len(main_list[j]):
              k=0
              while(k+len(main_list[i])<=len(main_list[j])):
                 # if it is a subset change the main_list and append to z
                 if main_list[i]==main_list[j][0+k:len(main_list[i])+k] and main_list[i]
                   z[i].append(main_list[j])

                 k+=1
    # print(z)
    for i in z.keys():
        if z[i]!=[]:
           for j in z[i] :
            if j in main_list:
             main_list.remove(j)
    # Calculate length of the main_list
    length = len(main_list)
    for i in main_list:
        u += "".join(i) + "+"
    u = u[: -1]
    '''If the output string length is equal to the input string return 0,
        u else return 1,u'''
    if(length1 == length):
        return 0,u
    else:
        return 1,u

'''Sending 2 of the individual strings from the input and check whether they
 are resolvable or not'''
def canBeReduced(list1, list2):
    count = 0
    temp_val = ""
    judge_list = []
    ''' for suppose input to this function is (ipt = nab + ab) then
        list1 = ['na', 'b'] and list2 = ['a', 'b']'''
    for iter1 in list1:
        sub_list = []
        for iter2 in list2:
            d = dict()
            # this for loop is used to get the character count
            for iter3 in (iter1 + iter2):
                if(iter3 not in d.keys()):
                    d[iter3] = 1
                else:
```

```
                    d[iter3] += 1
            # to remove duplicate values we used set over here
            if(len(d) == 2 and set(d.values()) == set([1, 2])):
                for iter4 in d.keys():
                    if('n' != iter4):
                        judge_list.append(iter4)
                        count += 1
    # judge_list contains the resolvable strings as the list form
    judge_list = set(judge_list)
    if(count == 1):
        for iter1 in judge_list:
            val = ""
            for iter2 in (list1 + list2):
                if(iter1 not in iter2):
                    val += iter2
        return 1, val
    return 0, None

index_list = []

def func(ipt):
    global index_list
    unique = []
    d = dict()
    main_list, temp = [], ""
    for iter1 in ipt.split("+"):
        sub_list = []
        flag = 0
        ''' For suppose (ipt = anb+bc) iter1 = anb, bc,
                main_list = [['a', 'nb'], ['b', 'c']]'''
        #print(iter2) #1st time = a and 2nd time = n and 3rd time  = b
        for iter2 in iter1:
            if(iter2 == 'n'):
                flag = 1
            if(iter2 != 'n'):
                temp = ('n' + iter2) if(flag) else iter2
                flag = 0
                sub_list.append(temp)
        main_list.append(sub_list)

    # print(main_list)

    l = len(main_list)
```

```
iter1 = 0
while(iter1 <= l-2):
    iter2 = iter1+1
    while(iter2 <= l-1):
        if([iter1, iter2] not in index_list):
            index_list.append([iter1, iter2])
            val = ""
            # we need to call canBeReduced function
            res, val = canBeReduced(main_list[iter1], main_list[iter2])
            if(res):
                sub_list, temp = [], ""
                flag = 0
                for iter3 in val:
                    if(iter3 == 'n'):
                        flag = 1
                    if(iter3 != 'n'):
 ''' in  this statement if(flag) is true then tempm will be
          ('n' + iter3) else iter3(it is the conditional statement)'''
                        temp = ('n' + iter3) if(flag) else iter3
                        flag = 0
                        sub_list.append(temp)
                count_main = 0
               #in this for loop the set comparison is done with main_list strings

                for iter3 in main_list:
                    if(set(sub_list) != set(iter3)):
                        count_main += 1
                if(count_main == l):
                    main_list.append(sub_list)
                    l += 1
                val3 = ""
                for iter3 in main_list:
                    # convert the mail_list into the string using join
                    val3 += "".join(iter3) + '+'
                val3 = val3[: -1]
                temp_var = []
                for iter3 in val3.split('+'):
                    if(iter3 not in temp_var):
                        temp_var.append(iter3)
                val3 = "+".join(temp_var)
                ipt = val3
                return func(ipt)
```

```
            iter2 += 1
        iter1 += 1
    return ipt

def main():
    global index_list
    a = 1
    while(a):
        val = ""
        # Enter the input string for the evaluation
        ipt = input("Enter input = ")
        ''' Split the given combined string into individual string which are
                seperated by "+"'''
        for iter1 in func(ipt).split('+'):
            temp, flag = "", 0
            d = dict()
            # Creating a dictionary and getting the key, value pairs
            for iter2 in iter1 :
                if(iter2 not in d.keys()):
                    d[iter2] = 1
                else:
                    d[iter2] += 1
            for k, v in d.items():
   '''After going through the dictionary keys if key vaalue does not
        contain n aand value is greater than 1 then set d[k] = 1'''
                if(k != 'n' and v > 1):
                    d[k] = 1
            for iter2 in iter1:
                if(d[iter2]):
                    #set flag = 1 when the iter2 string has n
                    if(iter2 == 'n'):
                        flag = 1
                    else:
                        temp += ("n" + iter2) if(flag) else iter2
                        flag = 0
                    d[iter2] -= 1
            val += temp + '+'
        ''' Since we are adding + to the temp string we used :-1 to print since
                we get rid of + in the val string'''
        val = val[: -1]

        main_list, temp = [], ""
        for iter1 in val.split("+"):
```

```
            sub_list = []
            flag = 0
            ''' For suppose we have (ipt = anb+bc) then iter1 = anb, bc
                        main_list = [['a', 'nb'], ['b', 'c']]'''
            for iter2 in iter1:
                if(iter2 == 'n'):
                    flag = 1
                if(iter2 != 'n'):
                    temp = ('n' + iter2) if(flag) else iter2
                    flag = 0
                    sub_list.append(temp)
            main_list.append(sub_list)
        ''' subsumption function will return a number and a string which will
                be assigned to flag and val'''
        flag, val = subsumption(main_list)
        if(flag):
            print("val =", val)
            a = int(input("Continue 1/0:"))
            if(a):
                index_list = []
            continue
        # print the final output
        print("val =", val)
        ''' Give input 1 if we want to enter another input string or give 0
                if you want to exit from the program'''
        a = int(input("Continue 1/0:"))
        if(a):
            index_list = []


if(__name__ == '__main__'):
    main()
```