Overview

The aim for this practical was to implement a system capable of parsing Boolean formulae and printing truth tables of the aforementioned formulae.

Design and Implementation

The desired formula to be parsed would be passed into the program as an attribute, accompanied by the number of variables contained within the formula.

To process the formula, I believed a stack would be helpful, allowing quick and simple access to the most recent operands. As such, a simple implementation of a stack was implemented, providing functionality to push and pop elements, as well as reading the top of the stack, determining whether or not the stack is full or empty, and construction methods for allocating and freeing memory to and from the stack.

With the stack implemented, it would be relatively simple to process a formula. One would simply have to iterate through the string, parsing each character, and executing the relevant operation. The characters are parsed using a switch, with each valid character having a different case.

Variables found within the formula would merely have their corresponding Boolean values pushed onto the stack. The operators, however, would call a function respective to its operation, popping the top two elements on the stack and passing them to this function, pushing the returned result back onto the stack. If the inspected character isn't recognised by the switch, the default case is triggered, and the system exits displaying a malformed formula message.

As for checking all possible variable value combinations, an array is created with the size of the number of variables (as determined by the user upon execution of the program). This array is flooded with zeros and passed into a recursive method. This method will alter the pass the values array into the formula parser before altering a zero value to a one and then recursively calling itself. This continues until every possible value combination has been exhausted.

The value array itself is related to the variables within the formula by a simple linear system. The value within the array contains the variable value of the letter in the alphabet in the same position of the array. For example, the first value in the array contains the value of the variable ‘a’. Since we can assume all variables within formulae are in alphabetical order, no validation is required.

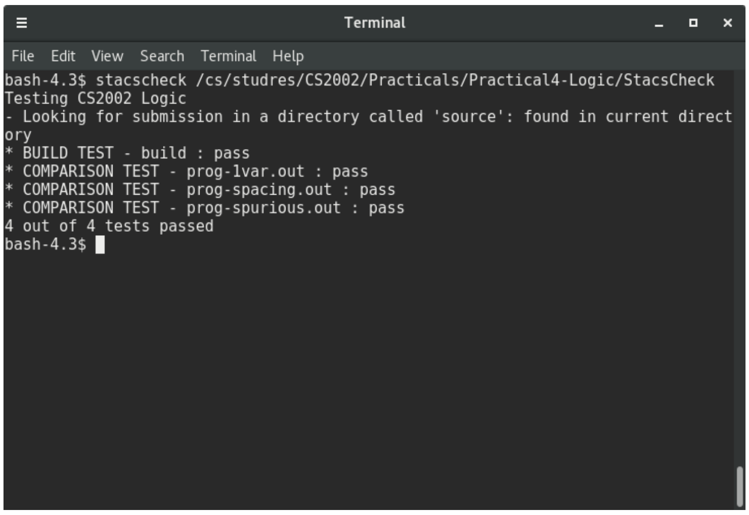
When printing the truth table, a function is called to print a correctly formatted table header, before the recursive parsing begins. With each new parsing, the current variable values are printed, followed by the operation values, and concluding with the final result. This ends with a correctly formatted truth table for the given formula.

Extension – Formulae Equality

If a second formula is also passed into the program upon execution, the system will process both formulae as normal, and then inspect whether the two are indeed equal.

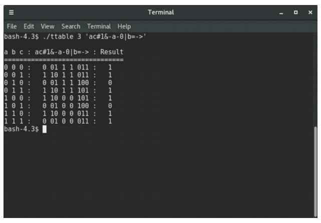
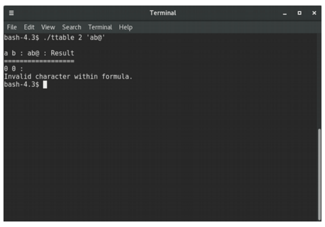
To achieve this, I simply implemented a function that took the two formula as parameters, concatenated them together and added an ‘=’ operator at the end. This would create a new formula designed to check the equality of the two separate ones. I then have the system parse this new formula, checking to see whether a final result is a zero. If so, an inequality has been found, and the system prints the truth table up to the failure point. If, however, all results are returned as true, the full truth table is displayed and the system recognises the equality.

Testing



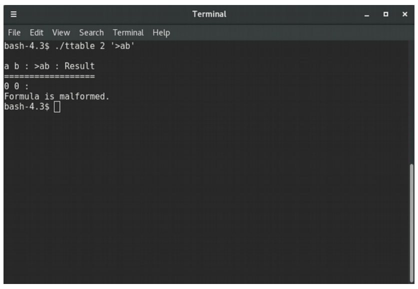
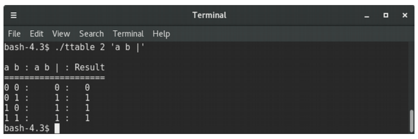
```
bash-4.3$ stacscheck /cs/studres/CS2002/Practicals/Practical4-Logic/StacsCheck
Testing CS2002 Logic
- Looking for submission in a directory called 'source': found in current direct
ory
* BUILD TEST - build : pass
* COMPARISON TEST - prog-1var.out : pass
* COMPARISON TEST - prog-spacing.out : pass
* COMPARISON TEST - prog-spurious.out : pass
4 out of 4 tests passed
bash-4.3$
```

My program passes all stacscheck tests.

Test No.	**Test Description**	**Expected Result**	**Actual Result**
1	Calling command given in specification. This tests all logic operator functionality.	Truth table within the specification will be printed.	Console display: 
2	Entering a formula containing an invalid character.	System recognises and quits graciously.	Console display: 

```
bash-4.3$ ./ttable 3 'ac#1&-a-0|b=->'

a b c : ac#1&-a-0|b=-> : Result
================================
0 0 0 :   0 01 1 1 011 :   1
0 0 1 :   1 10 1 1 011 :   1
0 1 0 :   0 01 1 1 100 :   0
0 1 1 :   1 10 1 1 101 :   1
1 0 0 :   1 10 0 0 101 :   1
1 0 1 :   0 01 0 0 100 :   0
1 1 0 :   1 10 0 0 011 :   1
1 1 1 :   0 01 0 0 011 :   1
bash-4.3$
```

```
bash-4.3$ ./ttable 2 'ab@'

a b : ab@ : Result
==================
0 0 :
Invalid character within formula.
bash-4.3$
```

3	Entering malformed formula.	System recognises error and quits.	Console display: 
4	Inputting formula with whitespace.	Whitespace ignored.	Console display: 

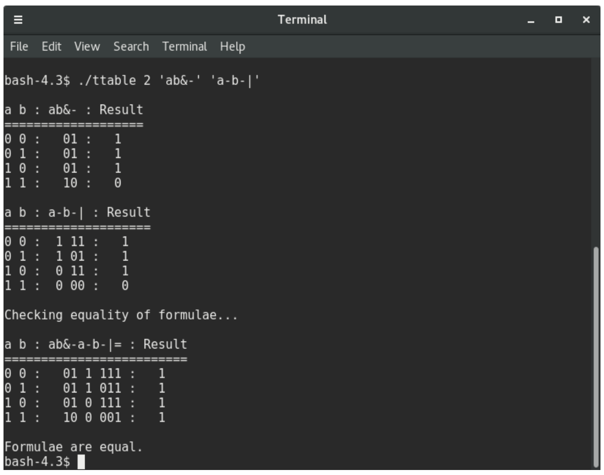
```
bash-4.3$ ./ttable 2 '>ab'

a b : >ab : Result
==================
0 0 :
Formula is malformed.
bash-4.3$ 
```

```
bash-4.3$ ./ttable 2 'a b |'

a b : a b | : Result
====================
0 0 :     0 :   0
0 1 :     1 :   1
1 0 :     1 :   1
1 1 :     1 :   1
bash-4.3$ 
```

Boolean Laws

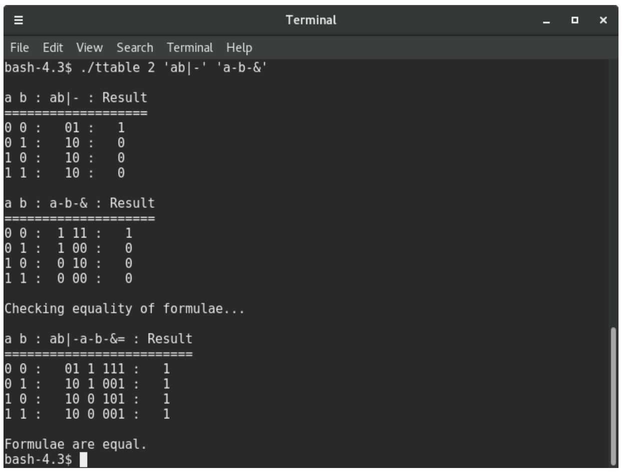
De Morgan's Laws

(a&b)' = a'|b'



```
bash-4.3$ ./ttable 2 'ab&-' 'a-b-|'

a b : ab&- : Result
===================
0 0 :   01 :   1
0 1 :   01 :   1
1 0 :   01 :   1
1 1 :   10 :   0

a b : a-b-| : Result
====================
0 0 :  1 11 :   1
0 1 :  1 01 :   1
1 0 :  0 11 :   1
1 1 :  0 00 :   0

Checking equality of formulae...

a b : ab&-a-b-|= : Result
=========================
0 0 :   01 1 111 :   1
0 1 :   01 1 011 :   1
1 0 :   01 0 111 :   1
1 1 :   10 0 001 :   1

Formulae are equal.
bash-4.3$ 
```

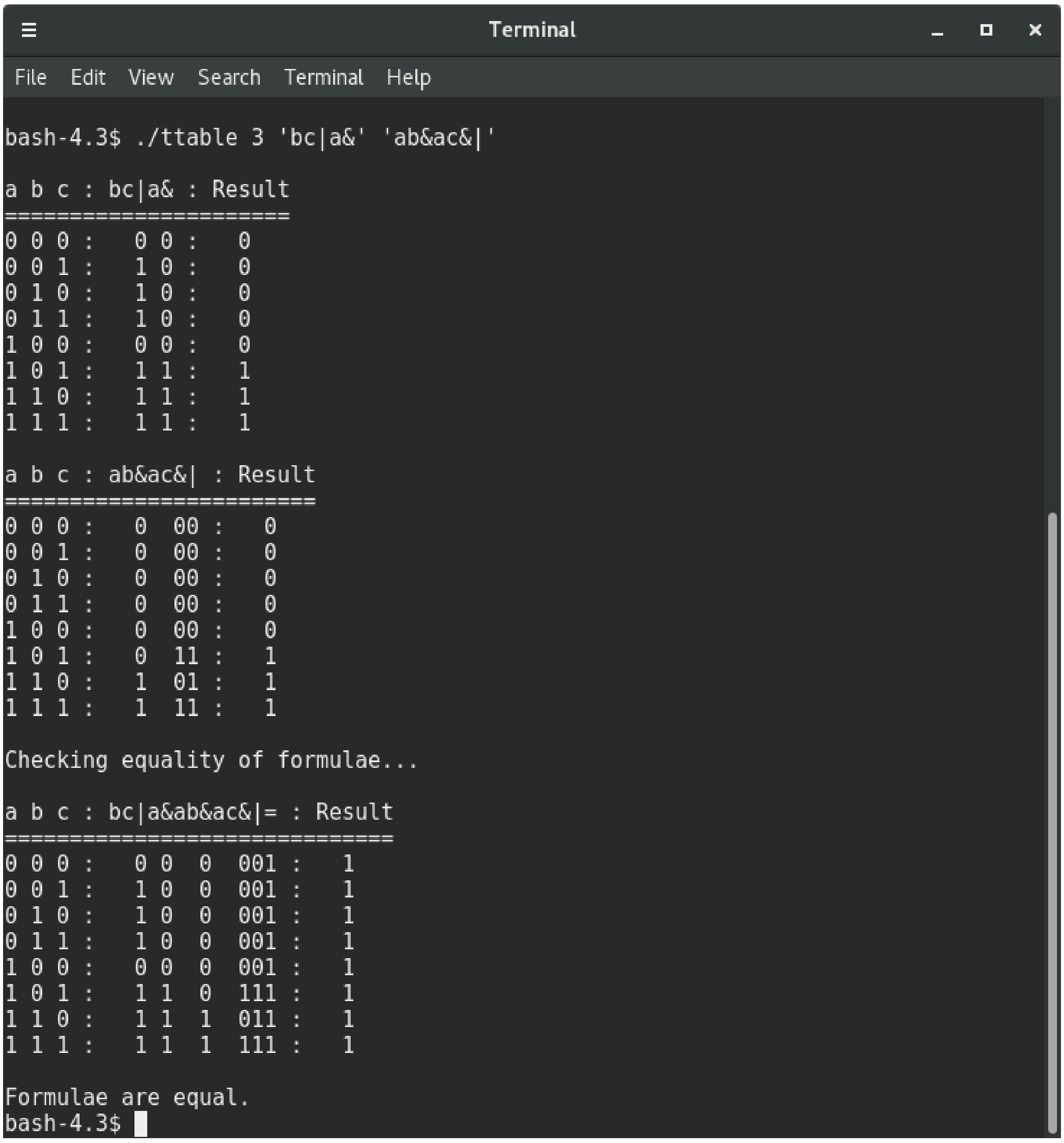
(a|b)' = a'&b'



```
bash-4.3$ ./ttable 2 'ab|-' 'a-b-&'

a b : ab|- : Result
===================
0 0 :   01 :   1
0 1 :   10 :   0
1 0 :   10 :   0
1 1 :   10 :   0

a b : a-b-& : Result
====================
0 0 :  1 11 :   1
0 1 :  1 00 :   0
1 0 :  0 10 :   0
1 1 :  0 00 :   0

Checking equality of formulae...

a b : ab|-a-b-&= : Result
=========================
0 0 :   01 1 111 :   1
0 1 :   10 1 001 :   1
1 0 :   10 0 101 :   1
1 1 :   10 0 001 :   1

Formulae are equal.
bash-4.3$ 
```

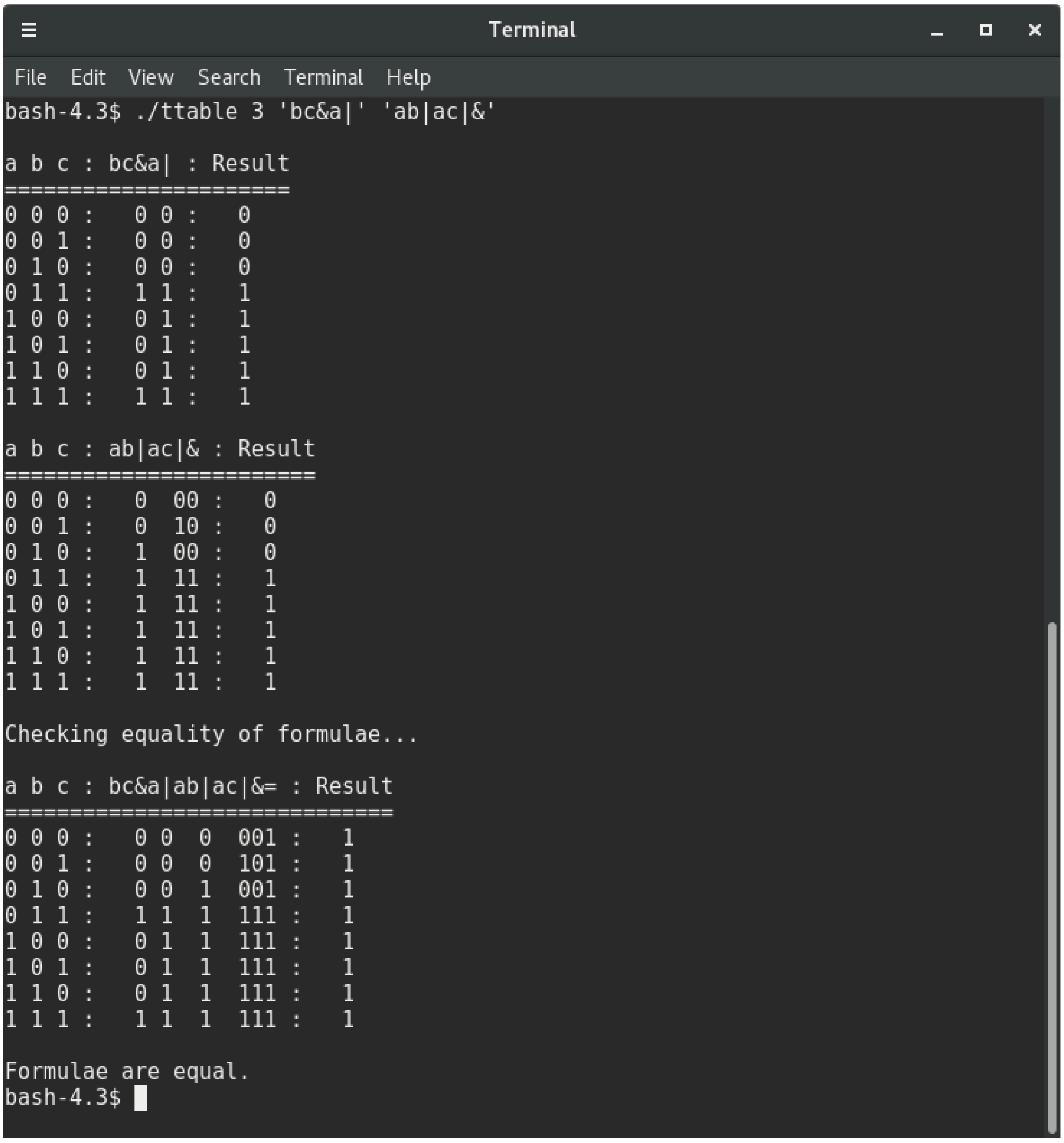
Distributive Laws

a&(b|c) = a&b|a&c



```
bash-4.3$ ./ttable 3 'bc|a&' 'ab&ac&|'

a b c : bc|a& : Result
======================
0 0 0 :   0 0 :   0
0 0 1 :   1 0 :   0
0 1 0 :   1 0 :   0
0 1 1 :   1 0 :   0
1 0 0 :   0 0 :   0
1 0 1 :   1 1 :   1
1 1 0 :   1 1 :   1
1 1 1 :   1 1 :   1

a b c : ab&ac&| : Result
========================
0 0 0 :   0  00 :   0
0 0 1 :   0  00 :   0
0 1 0 :   0  00 :   0
0 1 1 :   0  00 :   0
1 0 0 :   0  00 :   0
1 0 1 :   0  11 :   1
1 1 0 :   1  01 :   1
1 1 1 :   1  11 :   1

Checking equality of formulae...

a b c : bc|a&ab&ac&|= : Result
==============================
0 0 0 :   0 0  0  001 :   1
0 0 1 :   1 0  0  001 :   1
0 1 0 :   1 0  0  001 :   1
0 1 1 :   1 0  0  001 :   1
1 0 0 :   0 0  0  001 :   1
1 0 1 :   1 1  0  111 :   1
1 1 0 :   1 1  1  011 :   1
1 1 1 :   1 1  1  111 :   1

Formulae are equal.
bash-4.3$
```

a|(b&c) = (a|b)&(a|c)

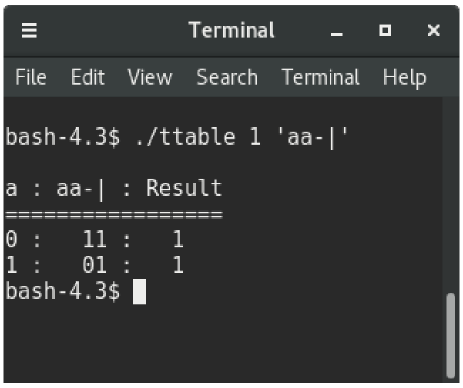


```
bash-4.3$ ./ttable 3 'bc&a|' 'ab|ac|&'

a b c : bc&a| : Result
======================
0 0 0 :   0 0 :   0
0 0 1 :   0 0 :   0
0 1 0 :   0 0 :   0
0 1 1 :   1 1 :   1
1 0 0 :   0 1 :   1
1 0 1 :   0 1 :   1
1 1 0 :   0 1 :   1
1 1 1 :   1 1 :   1

a b c : ab|ac|& : Result
========================
0 0 0 :   0  00 :   0
0 0 1 :   0  10 :   0
0 1 0 :   1  00 :   0
0 1 1 :   1  11 :   1
1 0 0 :   1  11 :   1
1 0 1 :   1  11 :   1
1 1 0 :   1  11 :   1
1 1 1 :   1  11 :   1

Checking equality of formulae...

a b c : bc&a|ab|ac|&= : Result
==============================
0 0 0 :   0 0  0  001 :   1
0 0 1 :   0 0  0  101 :   1
0 1 0 :   0 0  1  001 :   1
0 1 1 :   1 1  1  111 :   1
1 0 0 :   0 1  1  111 :   1
1 0 1 :   0 1  1  111 :   1
1 1 0 :   0 1  1  111 :   1
1 1 1 :   1 1  1  111 :   1

Formulae are equal.
bash-4.3$
```

Logic Puzzles

Coin toss

The variable is the value of the coin toss (either heads or tails). However, Chris will always win, so the final result must always be one. Therefore, the boolean formula for Chris winning is **a|a'**, with **a** being the value of the coin.



```
bash-4.3$ ./ttable 1 'aa-|'

a : aa-| : Result
=================
0 :   11 :   1
1 :   01 :   1
bash-4.3$ 
```

Ages

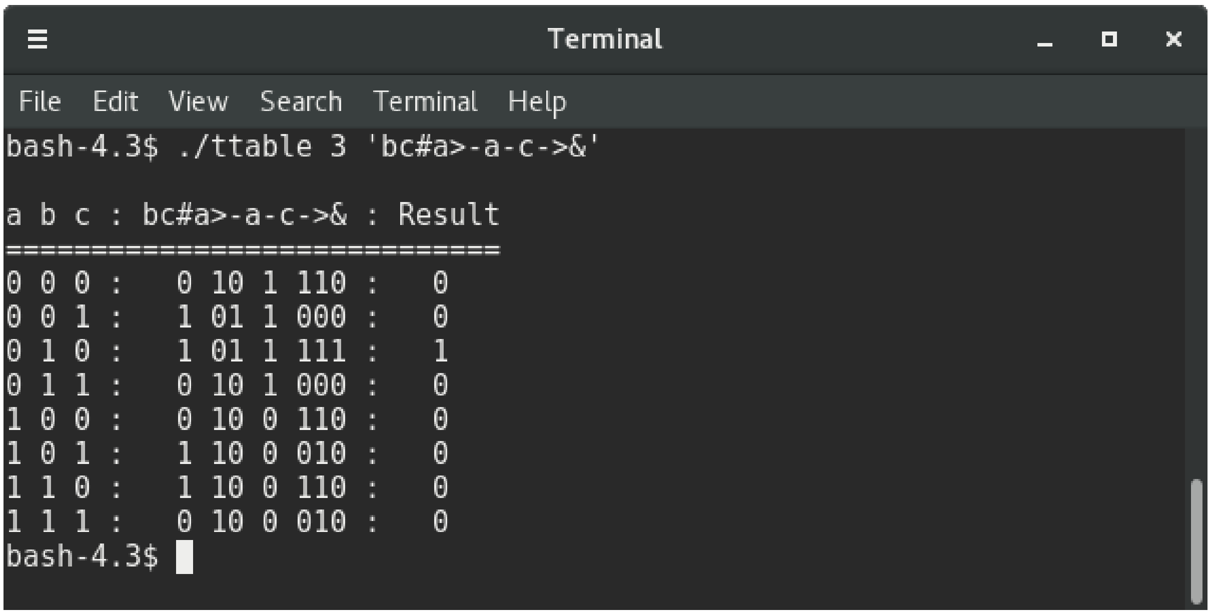
If Chris is the eldest, it means he cannot be the youngest, which implies that Ian is the eldest, but this cannot be true as you cannot have two eldest people. Therefore, Steve has to be the eldest. Since the eldest is now certain, Ian cannot be the eldest, so Chris must be the youngest, making Ian the middle.

Finding the eldest:

- Steve or Chris is the eldest – **b#c**

- Ian is the eldest or Chris is the youngest – **a' → c'**

This means the eldest is represented by **((b#c) → a)' & (a' → c')**



```
bash-4.3$ ./ttable 3 'bc#a>-a-c->&'

a b c : bc#a>-a-c->& : Result
=============================
0 0 0 :   0 10 1 110 :   0
0 0 1 :   1 01 1 000 :   0
0 1 0 :   1 01 1 111 :   1
0 1 1 :   0 10 1 000 :   0
1 0 0 :   0 10 0 110 :   0
1 0 1 :   1 10 0 010 :   0
1 1 0 :   1 10 0 110 :   0
1 1 1 :   0 10 0 010 :   0
bash-4.3$ 
```

This proves Steve is the eldest. From here, you can determine that Chris must be the youngest as Ian cannot be the eldest, therefore Ian is in the middle.

Dinner Party

- Either D or C or both attend – **d|c**

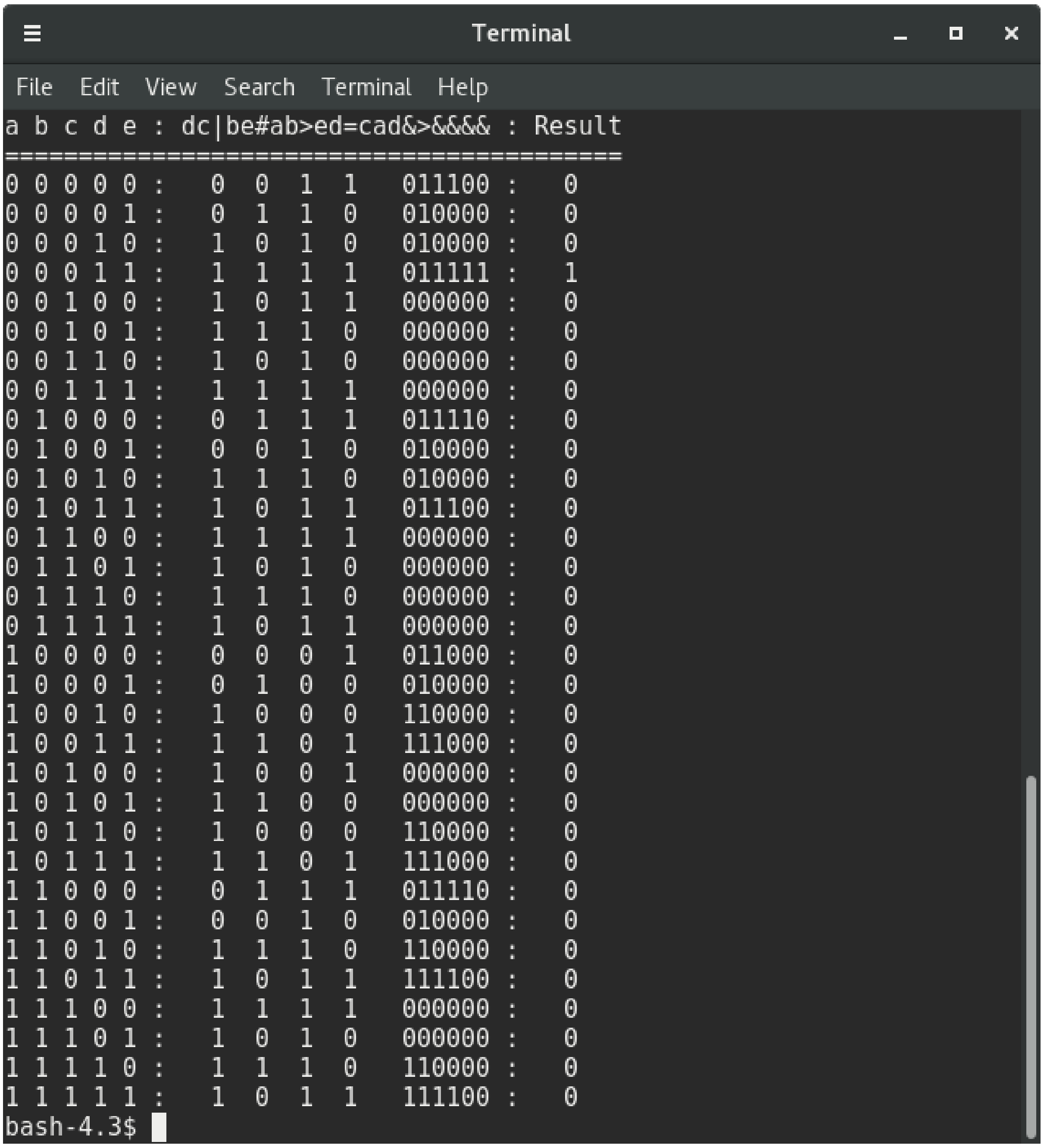
- Either B or E but not both attended – **b#e**

- If A attended, then so did B – **a → b**

- E attended if and only if D attended – **e=d**

- If C attended, then both A and D attended – **c → (a&d)**

To find the only solution, all of these must be true, therefore each singular formula must be anded together.



```
a b c d e : dc|be#ab>ed=cad&>&&&& : Result
==========================================
0 0 0 0 0 :   0  0  1  1   011100 :   0
0 0 0 0 1 :   0  1  1  0   010000 :   0
0 0 0 1 0 :   1  0  1  0   010000 :   0
0 0 0 1 1 :   1  1  1  1   011111 :   1
0 0 1 0 0 :   1  0  1  1   000000 :   0
0 0 1 0 1 :   1  1  1  0   000000 :   0
0 0 1 1 0 :   1  0  1  0   000000 :   0
0 0 1 1 1 :   1  1  1  1   000000 :   0
0 1 0 0 0 :   0  1  1  1   011110 :   0
0 1 0 0 1 :   0  0  1  0   010000 :   0
0 1 0 1 0 :   1  1  1  0   010000 :   0
0 1 0 1 1 :   1  0  1  1   011100 :   0
0 1 1 0 0 :   1  1  1  1   000000 :   0
0 1 1 0 1 :   1  0  1  0   000000 :   0
0 1 1 1 0 :   1  1  1  0   000000 :   0
0 1 1 1 1 :   1  0  1  1   000000 :   0
1 0 0 0 0 :   0  0  0  1   011000 :   0
1 0 0 0 1 :   0  1  0  0   010000 :   0
1 0 0 1 0 :   1  0  0  0   110000 :   0
1 0 0 1 1 :   1  1  0  1   111000 :   0
1 0 1 0 0 :   1  0  0  1   000000 :   0
1 0 1 0 1 :   1  1  0  0   000000 :   0
1 0 1 1 0 :   1  0  0  0   110000 :   0
1 0 1 1 1 :   1  1  0  1   111000 :   0
1 1 0 0 0 :   0  1  1  1   011110 :   0
1 1 0 0 1 :   0  0  1  0   010000 :   0
1 1 0 1 0 :   1  1  1  0   110000 :   0
1 1 0 1 1 :   1  0  1  1   111100 :   0
1 1 1 0 0 :   1  1  1  1   000000 :   0
1 1 1 0 1 :   1  0  1  0   000000 :   0
1 1 1 1 0 :   1  1  1  0   110000 :   0
1 1 1 1 1 :   1  0  1  1   111100 :   0
bash-4.3$
```

The solution is that both D and E attended the dinner party.