

# Long Title

## ABSTRACT

### ACM Reference Format:

. 2020. Long Title. In *Proceedings of The 9th Computer Science Education Research Conference (CSERC'20)*. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

## 1 INTRODUCTION

## 2 EXAMPLES

### 2.1 Example: Propositional and Predicate Logic

#### 2.1.1 De Morgan's Laws.

Verifying De Morgan's laws with valid version (tautology) as well as invalid version (with counterexample):

##### Input file:

```
1 p: BOOLEAN
2 q: BOOLEAN
3
4 -- formula is tautology
5 verify not (p and q) <=> not p or not q
6 verify not (p or q) <=> not p and not q
7
8 -- formula is not tautology
9 verify not (p and q) <=> not p and not q
```

##### Output result:

```
1 ((not (p and q)) = ((not p) or (not q)))
2 Is a tautology.
3
4 ((not (p or q)) = ((not p) and (not q)))
5 Is a tautology.
6
7 ((not (p and q)) = ((not p) and (not q)))
8 Where:
9   p : BOOLEAN
10  q : BOOLEAN
11
12 Is not a tautology. Here is a counter example:
13   p : false
14   q : true
```

#### 2.1.2 Quantification: Single forall ( $\forall$ ).

For quantification verification, there is no need to declare variables separately. **Input file:**

```
1 -- formula is tautology
2 verify forall i: INTEGER | i <= i * i
3
4 -- formula is not tautology
5 verify forall j: INTEGER | j < j * j
```

##### Output result:

```
1 forall i | (i <= (i * i))
2 Is a tautology.
```

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

CSERC'20, October 2019, Campus The Hague, Netherlands

© 2020 Copyright held by the owner/author(s).

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

```
4 forall j | (j < (j * j))
5 Where:
6   j : INTEGER
7 Is not a tautology. Here is a counter example:
8   j : 1
```

#### 2.1.3 Quantification: Single exists ( $\exists$ ).

##### Input file:

```
1 -- formula is tautology
2 verify exists p,q,r : BOOLEAN | (p or not q) and (q or
3   not r) and (r or not p)
4
5 -- formula is not tautology
6 verify exists s,t,v : BOOLEAN | (s and not t) and (t and
7   not v) and (v and not s)
```

##### Output result:

```
1 exists p,q,r | (((p or (not q)) and (q or (not r))) and (
2   r or (not p)))
3 Is a tautology.
4
5 exists s,t,v | (((s and (not t)) and (t and (not v))) and
6   (v and (not s)))
7 Where:
8   s : BOOLEAN
9   t : BOOLEAN
10  v : BOOLEAN
11 Is not a tautology.
12 Counterexample is not available.
```

#### 2.1.4 Quantification: Nested quantification ( $\forall / \exists$ ).

##### Input file:

```
1 -- formula is tautology
2 verify forall i: INTEGER | exists j: INTEGER | i <= j and
3   j <= i => i = j
4
5 -- formula is not tautology
6 verify exists k: INTEGER | forall n: INTEGER | k <= n and
7   n <= k => k = n + 1
```

##### Output result:

```
1 forall i | exists j | (((i <= j) and (j <= i)) => (i = j))
2 Is a tautology.
3
4 exists k | forall n | (((k <= n) and (n <= k)) => (k = (n
5   + 1)))
6 Where:
7   k : INTEGER
8   n : INTEGER
9 Is not a tautology.
10 Counterexample is not available.
```

## 2.2 Example: Program Verification

### 2.2.1 Compute Tax.

##### Input file:

```
1 compute_tax(status: INTEGER ; income: INTEGER) : REAL
2 require
3   positive_income: income >= 0
4 local
5   part1: REAL
6   part2: REAL
7   part3: REAL
8 do
9   if status = 1 or status = 2 then
```

```

10  if status = 1 then
11    if income <= 8350 then
12      part1 := income * 0.1;
13      Result := part1;
14    elseif income <= 33950 then
15      part1 := 8350 * 0.1;
16      part2 := (income - 8350) * 0.15;
17      Result := part1 + part2;
18    else
19      part1 := 8350 * 0.1;
20      part2 := (33950 - 8350) * 0.15;
21      part3 := (income - 33950) * 0.25;
22      Result := part1 + part2 + part3;
23    end
24  else
25    if income <= 16700 then
26      part1 := income * 0.1;
27      Result := part1;
28    elseif income <= 67900 then
29      part1 := 16700 * 0.1;
30      part2 := (income - 16700) * 0.15;
31      Result := part1 + part2;
32    else
33      part1 := 16700 * 0.1;
34      part2 := (67900 - 16700) * 0.15;
35      part3 := (income - 67900) * 0.25;
36      Result := part1 + part2 + part3;
37    end
38  end
39  else
40    Result := -1;
41  end
42  ensure
43  -- Discharged postcondition example
44  discharged: (status = 1 and income = 34870) => (Result
    = part1 + part2 + part3)
45
46  -- Not discharged postcondition example
47  not_discharged: (status = 2 and income > 67900) => (
    Result = 16700 * 0.1 + (67900 - 16700) * 0.15)
48  end
49
50  verify compute_tax

```

### Output result:

```

1  compute_tax(status : INTEGER ; income : INTEGER) : REAL
2  ... (Omitted)
3
4  Where:
5  Precondition(Q) :
6    positive_income : (income >= 0)
7  Postcondition(R) :
8    case1_3_specific : (((status = 1) and (income =
9    34870)) => (Result = ((part1 + part2) + part3)))
10   not_discharged : (((status = 2) and (income > 67900))
11   => (Result = ((16700 * 0.1) + ((67900 - 16700)
12   * 0.15))))
13  Implementation(S) :
14  ... (Omitted)
15
16  wp(S, discharged)

```

```

14  (((((status = 1) or (status = 2)) => (((status = 1) =>
    (((income <= 8350) => (((status = 1) and (income
    = 34870)) => ((income * 0.1) = ((income * 0.1) +
    part2) + part3)))) and ((not (income <= 8350))
    and (income <= 33950)) => (((status = 1) and (
    income = 34870)) => (((8350 * 0.1) + ((income -
    8350) * 0.15)) = ((8350 * 0.1) + ((income - 8350)
    * 0.15)) + part3)))) and ((not (income <= 8350)
    ) and (not (income <= 33950))) => (((status = 1)
    and (income = 34870)) => (((8350 * 0.1) + ((33950
    - 8350) * 0.15)) + ((income - 33950) * 0.25)) =
    (((8350 * 0.1) + ((33950 - 8350) * 0.15)) + ((
    income - 33950) * 0.25)))))) and ((not (status =
    1)) => (((income <= 16700) => (((status = 1) and
    (income = 34870)) => ((income * 0.1) = ((income *
    0.1) + part2) + part3)))) and ((not (income <=
    16700)) and (income <= 67900)) => (((status = 1)
    and (income = 34870)) => (((16700 * 0.1) + ((
    income - 16700) * 0.15)) = ((16700 * 0.1) + ((
    income - 16700) * 0.15)) + part3)))) and ((not (
    income <= 16700)) and (not (income <= 67900))) =>
    (((status = 1) and (income = 34870)) => (((16700
    * 0.1) + ((67900 - 16700) * 0.15)) + ((income -
    67900) * 0.25)) = (((16700 * 0.1) + ((67900 -
    16700) * 0.15)) + ((income - 67900) * 0.25))))))
    ) and ((not ((status = 1) or (status = 2))) => (((
    status = 1) and (income = 34870)) => (-1 = ((part1
    + part2) + part3))))))
15  wp(S, not_discharged)
16  (((((status = 1) or (status = 2)) => (((status = 1) =>
    (((income <= 8350) => (((status = 2) and (income
    > 67900)) => ((income * 0.1) = ((16700 * 0.1) +
    ((67900 - 16700) * 0.15)))) and ((not (income <=
    8350)) and (income <= 33950)) => (((status = 2)
    and (income > 67900)) => (((8350 * 0.1) + ((income
    - 8350) * 0.15)) = ((16700 * 0.1) + ((67900 -
    16700) * 0.15)))))) and ((not (income <= 8350))
    and (not (income <= 33950))) => (((status = 2) and
    (income > 67900)) => (((8350 * 0.1) + ((33950 -
    8350) * 0.15)) + ((income - 33950) * 0.25)) =
    (((16700 * 0.1) + ((67900 - 16700) * 0.15))))))
    ) and ((not (status = 1)) => (((income <= 16700) =>
    (((status = 2) and (income > 67900)) => ((income
    * 0.1) = ((16700 * 0.1) + ((67900 - 16700) * 0.15)
    )))) and ((not (income <= 16700)) and (income <=
    67900)) => (((status = 2) and (income > 67900)) =>
    (((16700 * 0.1) + ((income - 16700) * 0.15)) =
    ((16700 * 0.1) + ((67900 - 16700) * 0.15))))))
    ) and ((not (income <= 16700)) and (not (income <=
    67900))) => (((status = 2) and (income > 67900))
    => (((16700 * 0.1) + ((67900 - 16700) * 0.15)) +
    ((income - 67900) * 0.25)) = ((16700 * 0.1) +
    ((67900 - 16700) * 0.15)))))) and ((not ((
    status = 1) or (status = 2))) => (((status = 2)
    and (income > 67900)) => (-1 = ((16700 * 0.1) +
    ((67900 - 16700) * 0.15))))))
17
18  Proof Obligation:
19  (positive_income) => wp(S, discharged)
20  Discharged.
21  (positive_income) => wp(S, not_discharged)
22  Not discharged.
23  Counterexample:
24    income : 67901
25    status : 2

```

In this example, the output of program details in the beginning and the content of Implementation(S) are omitted because they are the same as the content of input file.

### 2.2.2 Loop: indices\_of.

#### Input file:

```

1  indices_of(a: ARRAY[INTEGER]; value: INTEGER): ARRAY[
    INTEGER]
2  require

```

```

3  not_empty: a.count > 0
4  local
5    i: INTEGER
6    j: INTEGER
7  do
8    from
9      i := 1;
10     j := 1;
11    invariant
12      j <= i
13    until
14      i > a.upper
15    loop
16      if a[i] = value then
17        Result[j] := i;
18        j := j + 1;
19      end
20      i := i + 1;
21    variant
22      loop_variant: a.upper - i + 1
23    end
24  ensure
25    -- discharged postcondition
26    case1: exists k1: INTEGER | a[k1] = value => exists s1:
      INTEGER | Result[s1] = k1
27  end
28
29  verify indices_of

```

### Output result:

```

1  indices_of(a : ARRAY[INTEGER];value : INTEGER) : ARRAY[
2    INTEGER]
3    ... (Omitted)
4
5  Where:
6  Precondition(Q) :
7    not_empty : (a.count > 0)
8  Postcondition(R) :
9    case1 : exists k1 | ((a[k1] = value) => exists s1 | (
10      Result[s1] = k1))
11
12  Implementation(S) :
13    ... (Omitted)
14
15  Correctness conditions :
16  1. Given precondition Q, the initialization step Sinit
17    establishes LI I : {Q} Sinit {I}
18    ((a.count > 0) => (1 <= 1))
19
20  2. At the end of Sbody, if not yet to exit, LI I is
21    maintained : {I and (not B)} Sbody {I}
22    (((j <= i) and (not (i > a.upper))) => (((a[i] = value)
23      => ((j + 1) <= (i + 1))) and ((not (a[i] = value)
24        ) => (j <= (i + 1)))))
25
26  3. If ready to exit and LI I maintained, postcondition R
27    is established : I and B => R
28    (((j <= i) and (i > a.upper)) => exists k1 | ((a[k1] =
29      value) => exists s1 | (Result[s1] = k1)))
30
31  4. Given LI I, and not yet to exit, Sbody maintains LV V
32    as non-negative : {I and (not B)} Sbody {V >= 0}
33    (((j <= i) and (not (i > a.upper))) => (((a[i] = value)
34      => (((a.upper - (i + 1)) + 1) >= 0)) and ((not (a
35        [i] = value)) => (((a.upper - (i + 1)) + 1) >= 0))
36      ))
37
38  5. Given LI I, and not yet to exit, Sbody decrements LV V
39    : {I and (not B)} Sbody {V < V0}
40    (((j <= i) and (not (i > a.upper))) => (((a[i] = value)
41      => (((a.upper - (i + 1)) + 1) < ((a.upper - i) +
42        1))) and ((not (a[i] = value)) => (((a.upper - (i
43          + 1)) + 1) < ((a.upper - i) + 1)))))
44
45  Condition 1 is discharged.
46  Condition 2 is discharged.
47  Condition 3 is discharged.
48  Condition 4 is discharged.

```

32 Condition 5 is discharged.

Also in this example, the output of program details in the beginning and the content of Implementation(S) are omitted because they are the same as the content of input file.

## 3 RELATED WORKS

Related Works here...