# Problem #1

Given a set of *n*, 1-bit numbers, the *max* of those *n* numbers is equal to 0b1 if and only if **any** of those numbers are 0b1. In contrast, the *min* of the *n* 1-bit numbers is 0b1 if and only if **all** *n* numbers are 0b1. Hence, for *n* 1-bit numbers, the *max* and *min* can be written as:

|  |  |  |
| --- | --- | --- |
|  |  | ( 1 ) |

where is the *i*th 1-bit number in the set of *n* numbers.

As an extension of the equation above, consider the case of a set of *k*-bit numbers (where ). Since any integer is composed of a series of bits, determining the minimum and maximum of the *n* numbers can be done at the bit level. The most significant bit (MSB) of the max/min is calculated using equation ; this is because MSB of the maximum/minimum only depends on the MSBs of the set of *n* numbers. For all subsequent bits, *j* (where *j* is a bit index defined as ), extra caution must be shown because it is not enough to only look at the value of the *j*thbit of a number since bit value is only relevant to a min/max operation if and only if all of its previous, more significant bits (i.e. and greater than 0) are equivalent to the corresponding bits in the min/max number.

**Example: Determine the maximum of a set of three, two-digit numbers { 0b00, 0b10, 0b01 }[[1]](#footnote-1)**

To determine the MSB of the maximum, you consider the most significant bit of all three words in the set, which are {0b0, 0b1, 0b0} respectively. Hence, the most significant bit (MSB) of max is equal to the OR of the MSBs of the three numbers in the set. When determining the value of the second bit (i.e. least significant in this case) for the maximum, one can only consider those numbers who preceding bits are equal to (i.e. same as) the previous, more significant bits in *max*. To check for equivalence, use the operator, , which is defined as:

|  |  |
| --- | --- |
|  | ( 2 ) |

If one did not check equivalence and used the equation as is, the *max* would erroneously be: 0b11 (with the MSB coming from word 0b10 and the LSB coming from word 0b01).

When one combines the requirement of checking preceding bit equivalence with equation , the complete Boolean expression to find the *j*th bit of the minimum and maximum of a set of *n* numbers is:

|  |  |  |
| --- | --- | --- |
|  |  | ( 3 ) |

where is the value of the *j*thbit of the *i*th number in the set of *n* numbers.

**Minimax Tree Values**

Minimax is a logical extension of equation . In this problem, each node in the tree has three successors. Depending on whether the node is in a *max* or *min* level, it applies the corresponding equation from on its children to determine the respective value of each of its *k*-bits. For this problem, *k* (i.e. bits per word) is 3 since the numbers are 3-bits in length, and the size of each set of numbers, *n*, is 3 (i.e. the number of successors). The following are the bit equations for each of the values in the 4 ply minimax as defined in equation . For level 1, the bit equations are expressed in conjunctive normal form (CNF). However, since these CNF equations are not in terms of only intermediary values and leaf nodes (e.g. ), additional manipulation of the equations is required to complete the question as stated.

**Max – Level 0 (Root of the Tree)**

|  |  |
| --- | --- |
|  | ( 4 ) |
|  | ( 5 ) |
|  | ( 6 ) |

**Min – Level 1 – Specifically for move m1.**

|  |  |
| --- | --- |
| **This is already in CNF.** | ( 7 ) |
| **This is converted to CNF via:** | ( 8 ) |
| **This is converted to CNF via:** | ( 9 ) |

**Max – Level 2**

|  |  |
| --- | --- |
|  | ( 10 ) |
|  | ( 11 ) |
|  | ( 12 ) |

**Min – Level 3**

|  |  |
| --- | --- |
|  | ( 13 ) |
|  | ( 14 ) |
|  | ( 15 ) |

**Non-Tree Intermediary Variables**

To simplify the solutions for and , three intermediary relations that are not part of the tree will be used. They are shown in equations ( 16 ), ( 17 ), and ( 18 ). Non-tree intermediary relations are used to simplify Boolean expressions and reduce growth in the resulting Boolean expressions.

|  |  |
| --- | --- |
|  | ( 16 ) |

|  |  |
| --- | --- |
|  | ( 17 ) |

|  |  |
| --- | --- |
|  | ( 18 ) |

**Intermediary Variables in the Minimax Tree**

The previously expressed CNF expressions for were not in terms of the leaf nodes as required by the problem. To successively substitute successor expressions back into the equations for would cause the length of the expressions to go exponentially and become even more unmanageable. In this section, a set of intermediary CNF expressions are derived that will be used to express in terms of the tree’s leaf nodes. This final expression of in terms of leaf nodes is done in the next section.

**Max – Level 2 (Root of the Tree)**

|  |  |
| --- | --- |
| To convert this to CNF: | ( 19 ) |
| **Use an Intermediary Variable :**  Using this simplified form, the equation can easily be converted to CNF.  This is then combined with the solution of ( 16 ) to get a sufficiently expressed expression for use in the CNF: | ( 20 ) |
| **Use an Intermediary Variable :**  This reduces the expression to:  The equation above is now in the same form as the revised equation in ( 20 ). Using that equation’s solution, substitute this statement’s symbols.  Since was used as a simplification, it must be plugged back into the equations using the solution from ( 17 ). | ( 21 ) |

**Min – Level 3**

|  |  |
| --- | --- |
|  | ( 22 ) |
| **Use an Intermediary Variable :**  This reduces the expression to:  Using this simplified form, the equation can easily be converted to CNF.  This solution is then combined with the solution of ( 17 ): | ( 23 ) |
| **Define:**  The equation above is now in the same form as the revised equation in ( 23 ). Using that equation’s solution, substitute these symbols.  This solution is then combined with the solution of ( 18 ): | ( 24 ) |

**Final CNFs**

The CNF for is in equation ; note it is the conjunction for the final derived equations in , , and . Equation is the full CNF for

|  |  |
| --- | --- |
| **Final CNF:** | ( 25 ) |

The CNF for is in equation ( 26 ). It is the conjunction final derived equations in ( 8 ), ( 20 ), ( 23 ), and ( 25 ).

|  |  |
| --- | --- |
| **Final CNF:** | ( 26 ) |

The CNF for is in equation ( 27 ). It is the conjunction final derived equations in ( 9 ), ( 21 ), ( 24 ), and ( 26 ).

|  |  |
| --- | --- |
| **Final CNF:** | ( 27 ) |

# Problem #2

**Question: The pigeonhole principle says any function from pigeons into holes must result in two pigeons in the same hole. Let be a variable expressing that pigeon gets mapped to hole . Consider the case.**

**Express the following as propositional formulas:**

1. **Every gets mapped to some .**
2. **Some  is mapped to by  and  where .**

**The conjunction of these two statements is a propositional formula for . Convert  to clausal form and give a resolution refutation for this statement. Finally, trace the execution of DPLL on this formula.**

**Part A:**

Each pigeon can be in only one hole. As such, for a given pigeon, there is a disjunction of conjunctions (i.e. OR of ANDs). Each conjunction explicitly limits the pigeon to a single hole. The subsequent derivation in equation ( 28 ) converts the equation to CNF.

|  |  |
| --- | --- |
|  | ( 28 ) |

**Part B:**

The pigeonhole principle is satisfied whenever any two pigeons are in any one hole. This translates to a large disjunction of conjunctions (i.e. disjunctive normal form – DNF) where each conjunction represents pigeons and being simultaneously in hole . The inverse of a DNF is a CNF; the DNF equation will become important when it comes to performing the resolution refutation.

|  |  |
| --- | --- |
|  | ( 29 ) |

**:**

The pigeonhole principles states that part implies . Hence:

|  |  |
| --- | --- |
|  | ( 30 ) |

**:**

To show the pigeonhole principle is valid, take its negation and show that the negation is unsatisfiable. The negation is done in equation . Note that the implication is removed via implication elimination.

|  |  |
| --- | --- |
|  | ( 31 ) |

When equation is populated with the resolved equations in and , the negation of the pigeonhole principle with three holes and four pigeons, , is in CNF. This is shown in equation .

|  |  |
| --- | --- |
|  | ( 32 ) |

**Resolution Refutation:**

Resolution refutation necessitates combining clauses which have literal(s) whose sign(s) (i.e. positive or negative) is/are complementary. When combining these clauses, the goal for resolution refutation is to find an empty clause. The four step proof of resolution refutation is shown in equations ( 33 ) to ( 36 ).

|  |  |
| --- | --- |
| From : | ( 33 ) |
| From : | ( 34 ) |
| From and : | ( 35 ) |
| From and :  Completing the Resolution Refutation since the empty clause was found. | ( 36 ) |

**DPLL Algorithm**

In the DPLL algorithm, there are four distinct steps per iteration; the steps and their sequential ordering are:

1. If a clause has been assigned to false, terminate. Similarly, if the assignment satisfies the set of clauses, return the assignment as the expression has been satisfied.
2. Check for any pure symbols (i.e. symbols that have the same sign in all clauses).
3. Check for any unit clauses (i.e. any clause with only one symbol)
4. Choose the first symbol from the list of unassigned symbols. Test assigning both true and false to that symbol and see if either assignment satisfies the expression.

I wrote a program (**HW3\_Q2\_DPLL.py**) to execute the DPLL algorithm on this CNF. Below is the output from my program. Each step the algorithm took before reaching an empty clause is listed as well as the initial conditions (e.g. clauses and model). This step by step description of the algorithm’s operations serves as the trace.

The clauses are below. A plus sign ("+") before a symbol name indicates a positive literal.

A minus sign ("-") before a symbol name indicates a negated literal.

[['-P1,1', '-P1,2'], ['-P1,1', '-P1,3'], ['-P1,2', '-P1,3'], ['+P1,1', '+P1,2', '+P1,3'], ['-P2,1', '-P2,2'], ['-P2,1', '-P2,3'], ['-P2,2', '-P2,3'], ['+P2,1', '+P2,2', '+P2,3'], ['-P3,1', '-P3,2'], ['-P3,1', '-P3,3'], ['-P3,2', '-P3,3'], ['+P3,1', '+P3,2', '+P3,3'], ['-P4,1', '-P4,2'], ['-P4,1', '-P4,3'], ['-P4,2', '-P4,3'], ['+P4,1', '+P4,2', '+P4,3'], ['-P1,1', '-P2,1'], ['-P1,1', '-P3,1'], ['-P1,1', '-P4,1'], ['-P2,1', '-P3,1'], ['-P2,1', '-P4,1'], ['-P3,1', '-P4,1'], ['-P1,2', '-P2,2'], ['-P1,2', '-P3,2'], ['-P1,2', '-P4,2'], ['-P2,2', '-P3,2'], ['-P2,2', '-P4,2'], ['-P3,2', '-P4,2'], ['-P1,3', '-P2,3'], ['-P1,3', '-P3,3'], ['-P1,3', '-P4,3'], ['-P2,3', '-P3,3'], ['-P2,3', '-P4,3'], ['-P3,3', '-P4,3']]

The model is: ['P1,1', 'P1,2', 'P1,3', 'P2,1', 'P2,2', 'P2,3', 'P3,1', 'P3,2', 'P3,3', 'P4,1', 'P4,2', 'P4,3']

Step #1: Try assigning symbol "P1,1" to "True".

Step #2: Symbol "P1,2" is a pure symbol. It was assigned to "False".

Step #3: Symbol "P1,3" is a pure symbol. It was assigned to "False".

Step #4: Unit clause found for symbol "P2,1". It was assigned to "False".

Step #5: Unit clause found for symbol "P3,1". It was assigned to "False".

Step #6: Unit clause found for symbol "P4,1". It was assigned to "False".

Step #7: Try assigning symbol "P2,2" to "True".

Step #8: Symbol "P2,3" is a pure symbol. It was assigned to "False".

Step #9: Unit clause found for symbol "P3,2". It was assigned to "False".

Step #10: Unit clause found for symbol "P3,3". It was assigned to "True".

Step #11: Unit clause found for symbol "P4,2". It was assigned to "False".

Step #12: Unit clause found for symbol "P4,3". It was assigned to "True".

Step #13: Empty clause found. Recursing...

Assigning symbol "P2,2" to "True" failed.

Step #14: Try assigning symbol "P2,2" to "False".

Step #15: Unit clause found for symbol "P2,3". It was assigned to "True".

Step #16: Unit clause found for symbol "P3,3". It was assigned to "False".

Step #17: Unit clause found for symbol "P3,2". It was assigned to "True".

Step #18: Unit clause found for symbol "P4,2". It was assigned to "False".

Step #19: Unit clause found for symbol "P4,3". It was assigned to "True".

Step #20: Empty clause found. Recursing...

Assigning symbol "P1,1" to "True" failed.

Step #21: Try assigning symbol "P1,1" to "False".

Step #22: Try assigning symbol "P1,2" to "True".

Step #23: Symbol "P1,3" is a pure symbol. It was assigned to "False".

Step #24: Unit clause found for symbol "P2,2". It was assigned to "False".

Step #25: Unit clause found for symbol "P3,2". It was assigned to "False".

Step #26: Unit clause found for symbol "P4,2". It was assigned to "False".

Step #27: Try assigning symbol "P2,1" to "True".

Step #28: Symbol "P2,3" is a pure symbol. It was assigned to "False".

Step #29: Unit clause found for symbol "P3,1". It was assigned to "False".

Step #30: Unit clause found for symbol "P3,3". It was assigned to "True".

Step #31: Unit clause found for symbol "P4,1". It was assigned to "False".

Step #32: Unit clause found for symbol "P4,3". It was assigned to "True".

Step #33: Empty clause found. Recursing...

Assigning symbol "P2,1" to "True" failed.

Step #34: Try assigning symbol "P2,1" to "False".

Step #35: Unit clause found for symbol "P2,3". It was assigned to "True".

Step #36: Unit clause found for symbol "P3,3". It was assigned to "False".

Step #37: Unit clause found for symbol "P3,1". It was assigned to "True".

Step #38: Unit clause found for symbol "P4,1". It was assigned to "False".

Step #39: Unit clause found for symbol "P4,3". It was assigned to "True".

Step #40: Empty clause found. Recursing...

Assigning symbol "P1,2" to "True" failed.

Step #41: Try assigning symbol "P1,2" to "False".

Step #42: Unit clause found for symbol "P1,3". It was assigned to "True".

Step #43: Unit clause found for symbol "P2,3". It was assigned to "False".

Step #44: Unit clause found for symbol "P3,3". It was assigned to "False".

Step #45: Unit clause found for symbol "P4,3". It was assigned to "False".

Step #46: Try assigning symbol "P2,1" to "True".

Step #47: Symbol "P2,2" is a pure symbol. It was assigned to "False".

Step #48: Unit clause found for symbol "P3,1". It was assigned to "False".

Step #49: Unit clause found for symbol "P3,2". It was assigned to "True".

Step #50: Unit clause found for symbol "P4,1". It was assigned to "False".

Step #51: Unit clause found for symbol "P4,2". It was assigned to "True".

Step #52: Empty clause found. Recursing...

Assigning symbol "P2,1" to "True" failed.

Step #53: Try assigning symbol "P2,1" to "False".

Step #54: Unit clause found for symbol "P2,2". It was assigned to "True".

Step #55: Unit clause found for symbol "P3,2". It was assigned to "False".

Step #56: Unit clause found for symbol "P3,1". It was assigned to "True".

Step #57: Unit clause found for symbol "P4,1". It was assigned to "False".

Step #58: Unit clause found for symbol "P4,2". It was assigned to "True".

Step #59: Empty clause found. Recursing...

These clauses are unsatisfiable.

1. The case of *min* is logical extension of the *max* example and is not shown. However, its equation is provided in . [↑](#footnote-ref-1)