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:

$max_1 = \bigvee_{i=1}^{n} V_i$	$min_1 = \bigwedge_{i=1}^{n} V_i$	(1)

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

As an extension of the equation (1) above, consider the case of a set of *k*-bit numbers (where $k > 1$). 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 (1); 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 $1 < j \leq k$), extra caution must be shown because it is not enough to only look at the j^{th} bit of the *n* numbers since a number, V_i, is only relevant to a min/max operation for bit *j* if all of its previous, more significant bits (i.e. $< j$ 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]

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 the maximum (i.e. OR) of three numbers is 0b1. 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 (same) to the previous, more significant bits in *max*. To check for equivalence, use the operator, ⇔, which is defined as:

$(A \Leftrightarrow B) \equiv (A \wedge B) \vee (\bar{A} \wedge \bar{B})$	(2)

In simpler terms, both bits must be 0b0 or both bits must be 0b1. If one did not check equivalence and used the equation (1) 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 (1), the complete Boolean expression to find the j^{th} bit of the minimum and maximum of a set of *n* numbers is:

$max_j = \bigvee_{i=1}^{n} \left(V_{i,j} \wedge \left(\bigwedge_{k=1}^{j-1} (max_k \Leftrightarrow V_{i,k}) \right) \right)$, for $j > 1$	$min_j = \bigwedge_{i=1}^{n} \left(V_{i,j} \vee \left(\bigvee_{k=1}^{j-1} \overline{(min_k \Leftrightarrow V_{i,k})} \right) \right)$, for $j > 1$	(3)

where $V_{i,j}$ is j^{th} bit of the i^{th} number in the set of *n* numbers.

Minimax is a logical extension of equation (3). Each node in the tree has three children. Depending on whether the node is in a *max* or *min* level, it applies the corresponding equation from (3) 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 size of each set of numbers, *n*, is 3 (i.e. the number of children). The following are the bit equations for each of the values in the 4 ply minimax as defined in equation (3):

[1] The case of *min* is logical extension of the *max* example and is not shown. However, its equation is provided in (**Error! Main Document Only.**).

Max – Level 0 (Root of the Tree)

$$V_1 = \bigvee_{i=1}^{3} V_{i,1}$$	(4)
$$V_2 = \bigvee_{i=1}^{3} (V_1 \Leftrightarrow V_{i,1}) \wedge V_{i,2}$$	(5)
$$V_3 = \bigvee_{i=1}^{3} (V_1 \Leftrightarrow V_{i,1}) \wedge (V_2 \Leftrightarrow V_{i,2}) \wedge V_{i,3}$$	(6)

Min – Level 1

$$V_{i,1} = \bigwedge_{j=i}^{3} V_{i,j,1}$$	(7)
$$V_{i,2} = \bigwedge_{j=1}^{3} \overline{(V_{i,1} \Leftrightarrow V_{i,j,1})} \vee V_{i,j,2}$$	(8)
$$V_{i,3} = \bigwedge_{j=1}^{3} \overline{(V_{i,1} \Leftrightarrow V_{i,j,1})} \vee \overline{(V_{i,2} \Leftrightarrow V_{i,j,2})} \vee V_{i,j,3}$$	(9)

Max – Level 2 (Root of the Tree)

$$V_{i,j,1} = \bigvee_{i=1}^{3} V_{i,j,k,1}$$	(10)
$$V_{i,j,2} = \bigvee_{i=1}^{3} (V_{i,j,1} \Leftrightarrow V_{i,j,k,1}) \wedge V_{i,j,k,2}$$	(11)
$$V_{i,j,3} = \bigvee_{i=1}^{3} (V_{i,j,1} \Leftrightarrow V_{i,j,k,1}) \wedge (V_{i,j,2} \Leftrightarrow V_{i,j,k,2}) \wedge V_{i,j,k,3}$$	(12)

Min – Level 3

$$V_{i,j,k,1} = \bigwedge_{j=i}^{3} V_{i,j,k,m,k1}$$	(13)
$$V_{i,j,k,2} = \bigwedge_{j=1}^{3} \overline{(V_{i,j,k,1} \Leftrightarrow V_{i,j,k,m,1})} \vee V_{i,j,k,m,2}$$	(14)
$$V_{i,j,k,3} = \bigwedge_{j=1}^{3} \overline{(V_{i,j,k,1} \Leftrightarrow V_{i,j,k,m,1})} \vee \overline{(V_{i,j,k,2} \Leftrightarrow V_{i,j,k,m,2})} \vee V_{i,j,k,m,3}$$	(15)

Equations (4), (7), (10), and (13) (i.e. the equations for the most significant bits of each level's word) are already in CNF. Equations (8) and (14) can be converted to CNF through equations (16) to (20).

$$V_{i,2} = \bigwedge_{j=1}^{3} \overline{(V_{i,1} \Leftrightarrow V_{i,j,1})} \vee V_{i,j,2}$$	(16)

$$V_{i,2} = \bigwedge_{j=1}^{3} \left(\overline{\left((V_{i,1} \wedge V_{i,J,1}) \vee (\overline{V_{i,1}} \wedge \overline{V_{i,J,1}}) \right)} \vee V_{i,j,2} \right)$$	(17)
$$V_{i,2} = \bigwedge_{j=1}^{3} \left(\left(\overline{(V_{i,1} \wedge V_{i,J,1})} \wedge \overline{(\overline{V_{i,1}} \wedge \overline{V_{i,J,1}})} \right) \vee V_{i,j,2} \right)$$	(18)
$$V_{i,2} = \bigwedge_{j=1}^{3} \left(\left((\overline{V_{i,1}} \vee \overline{V_{i,J,1}}) \wedge (V_{i,1} \vee V_{i,j,1}) \right) \vee V_{i,j,2} \right)$$	(19)
$$V_{i,2} = \bigwedge_{j=1}^{3} \left((\overline{V_{i,1}} \vee \overline{V_{i,J,1}} \vee V_{i,j,2}) \wedge (V_{i,1} \vee V_{i,j,1} \vee V_{i,j,2}) \right)$$	(20)

$V_{1,b}$ can be modified to use only symbols $V_{i,j,k,m,b}$ by doing successive substitution using the equations above. This can be done either by hand, which has limited scalability or though automation. I chose to automate this process by writing a program to do the substitution and then convert to CNF. The program is written in Python (HW3_Q1_Sympy_Solver.py) and uses the Python SymPy library. SymPy is open source and can be downloaded from https://pypi.python.org/pypi/sympy. Included with this homework is a text file "**HW3_Q1_Results.txt**" which is the output of my SymPy program. It contains $V_{i,b}$ both in non-CNF form as well as in CNF form; these Boolean expressions are in terms of the base proposition symbols (i.e. $V_{i,j,k,m,b}$) as required. Most of the outputs of the equations in both CNF and non-CNF formats are too long to reasonably appear in this document. Kindly refer to that as a reference.

Appendix A contains the CNF for $V_{1,1}$. This is given as an example of the tool's output. "`Not`" operations are represented with a exclamation point ("`!`"); "`Or`" operations are denoted with a plus sign ("`+`") while "`And`" operations are denoted with an ampersand ("`&`"). The canonical form for $V_{1,1}$ as shown in Appendix A is:

$$\bigwedge_{j=1}^{3} \left(\bigwedge_{r=1}^{3} \left(\bigwedge_{s=1}^{3} \left(\bigwedge_{t=1}^{3} (V_{1,j,1,r,i} \vee V_{1,j,1,s,i} \vee V_{1,j,1,t,i}) \right) \right) \right)$$

Problem #2

Question: The pigeonhole principle PHP_n^{n+1} says any function from $n+1$ pigeons into n holes must result in two pigeons in the same hole. Let $P_{i,j}$ be a variable expressing that pigeon i gets mapped to hole j. Consider the $n=3$ case.

Express the following as propositional formulas:

a. **Every i gets mapped to some j.**
b. **Some j is mapped to by i and i' where $i \neq i'$.**

The conjunction of these two statements is a propositional formula for PHP_3^4. Convert $\neg PHP$ 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 one hole and 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 (21) converts the equation in CNF.

$$
\begin{aligned}
R_1 &: \bigwedge_{i=1}^{4} \left(\left(P_{i,1} \wedge \overline{P_{i,2}} \wedge \overline{P_{i,3}}\right) \vee \left(\overline{P_{i,1}} \wedge P_{i,2} \wedge \overline{P_{i,3}}\right) \vee \left(\overline{P_{i,1}} \wedge \overline{P_{i,2}} \wedge P_{i,3}\right) \right) \\
R_1 &: \bigwedge_{i=1}^{4} \left(\neg\neg \left(\left(P_{i,1} \wedge \overline{P_{i,2}} \wedge \overline{P_{i,3}}\right) \vee \left(\overline{P_{i,1}} \wedge P_{i,2} \wedge \overline{P_{i,3}}\right) \vee \left(\overline{P_{i,1}} \wedge \overline{P_{i,2}} \wedge P_{i,3}\right) \right) \right) \\
R_1 &: \bigwedge_{i=1}^{4} \left(\neg \overline{\left(\left(P_{i,1} \wedge \overline{P_{i,2}} \wedge \overline{P_{i,3}}\right) \vee \left(\overline{P_{i,1}} \wedge P_{i,2} \wedge \overline{P_{i,3}}\right) \vee \left(\overline{P_{i,1}} \wedge \overline{P_{i,2}} \wedge P_{i,3}\right) \right)} \right) \\
R_1 &: \bigwedge_{i=1}^{4} \left(\neg \left(\overline{\left(P_{i,1} \wedge \overline{P_{i,2}} \wedge \overline{P_{i,3}}\right)} \wedge \overline{\left(\overline{P_{i,1}} \wedge P_{i,2} \wedge \overline{P_{i,3}}\right)} \wedge \overline{\left(\overline{P_{i,1}} \wedge \overline{P_{i,2}} \wedge P_{i,3}\right)} \right) \right) \\
R_1 &: \bigwedge_{i=1}^{4} \left(\neg \left(\left(\overline{P_{i,1}} \vee P_{i,2} \vee P_{i,3}\right) \wedge \left(P_{i,1} \vee \overline{P_{i,2}} \vee P_{i,3}\right) \vee \left(P_{i,1} \vee P_{i,2} \vee \overline{P_{i,3}}\right) \right) \right) \\
R_1 &: \bigwedge_{i=1}^{4} \left(\neg \left(\left(P_{i,1} \wedge P_{i,2}\right) \vee \left(P_{i,1} \wedge P_{i,3}\right) \vee \left(P_{i,2} \wedge P_{i,3}\right) \vee \left(\overline{P_{i,1}} \wedge \overline{P_{i,2}} \wedge \overline{P_{i,3}}\right) \right) \right) \\
R_1 &: \bigwedge_{i=1}^{4} \left(\left(\overline{P_{i,1}} \vee \overline{P_{i,2}}\right) \wedge \left(\overline{P_{i,1}} \vee \overline{P_{i,3}}\right) \wedge \left(\overline{P_{i,2}} \vee \overline{P_{i,3}}\right) \wedge \left(P_{i,1} \vee P_{i,2} \vee P_{i,3}\right) \right)
\end{aligned}
\qquad (21)
$$

Part B:

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

$R_2: \bigvee_{j=1}^{3} (P_{1,j} \wedge P_{2,j}) \vee (P_{1,j} \wedge P_{3,j}) \vee (P_{1,j} \wedge P_{4,j}) \vee (P_{2,j} \wedge P_{3,j}) \vee (P_{2,j} \wedge P_{4,j}) \vee (P_{3,j} \wedge P_{4,j})$ $\overline{R_2}: \bigwedge_{j=1}^{3} \left((\overline{P_{1,J}} \vee \overline{P_{2,J}}) \wedge (\overline{P_{1,J}} \vee \overline{P_{3,J}}) \wedge (\overline{P_{1,J}} \vee \overline{P_{4,J}}) \wedge (\overline{P_{2,J}} \vee \overline{P_{3,J}}) \wedge (\overline{P_{2,J}} \vee \overline{P_{4,J}}) \wedge (\overline{P_{3,J}} \vee \overline{P_{4,J}}) \right)$	(22)

PHP_3^4:

The pigeonhole principles states that part R_1 implies R_2. Hence:

$(R_1 \Rightarrow R_2)$	(23)

$\rightarrow PHP_3^4$:

To show the pigeonhole principle is valid, take its negation and show it is unsatisfiable. This is done in equation (24). Note that the implication is removed via implication elimination.

$\rightarrow (R_1 \Rightarrow R_2)$ $\rightarrow (\overline{R_1} \vee R_2)$ $R_1 \wedge \overline{R_2}$	(24)

When equation (24) is combined with the resolved equations in (21) and (22), the negation of the pigeonhole principle with three holes and four pigeons, $\rightarrow PHP_3^4$, is in CNF. This is shown in equation (25).

$\bigwedge_{i=1}^{4} \left((\overline{P_{i,1}} \vee \overline{P_{i,2}}) \wedge (\overline{P_{i,1}} \vee \overline{P_{i,3}}) \wedge (\overline{P_{i,2}} \vee \overline{P_{i,3}}) \wedge (P_{i,1} \vee P_{i,2} \vee P_{i,3}) \right)$ $\wedge \quad \bigwedge_{j=1}^{3} \left((\overline{P_{1,J}} \vee \overline{P_{2,J}}) \wedge (\overline{P_{1,J}} \vee \overline{P_{3,J}}) \wedge (\overline{P_{1,J}} \vee \overline{P_{4,J}}) \wedge (\overline{P_{2,J}} \vee \overline{P_{3,J}}) \wedge (\overline{P_{2,J}} \vee \overline{P_{4,J}}) \wedge (\overline{P_{3,J}} \vee \overline{P_{4,J}}) \right)$	(25)

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.

From R_1: $R_3: \frac{(\overline{P_{1,2}} \vee \overline{P_{1,3}}),\ (P_{1,1} \vee P_{1,2} \vee P_{1,3})}{P_{1,1}}$	(26)
From R_1: $R_4: \frac{(\overline{P_{2,2}} \vee \overline{P_{2,3}}),\ (P_{2,1} \vee P_{2,2} \vee P_{2,3})}{P_{2,1}}$	(27)
From $\overline{R_2}$ and R_3: $R_5: \frac{(\overline{P_{1,1}} \vee \overline{P_{2,1}}),\quad P_{1,1}}{\overline{P_{2,1}}}$	(28)
From R_4 and R_5:	(29)

$R_6: \frac{P_{2,1},\ \overline{P_{2,1}}}{\{\}}$ Completing the Resolution Refutation since the empty clause was found.	

DPLL Algorithm

In the DPLL algorithm, there are four distinct steps per iteration. They are in sequential order:

1. If a clause has been assigned to false or the assignment is true, 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 algorithm. Each step the algorithm took before reaching an empty clause is listed as well as the initial conditions (e.g. clauses and model).

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

Appendix A – $V_{1,b}$ in Conjunctive Normal Form

Note: The CNF equations for problem #1 were solved using my program "HW3_Q1_Sumpy_Solver.py". I included the $V_{1,1}$ in CNF since it is length is manageable as an example of the tool's output.

$V_{1,1} =$

```
(V1,1,1,1,1+V1,1,2,1,1+V1,1,3,1,1)
&(V1,1,1,1,1+V1,1,2,1,1+V1,1,3,2,1)
&(V1,1,1,1,1+V1,1,2,1,1+V1,1,3,3,1)
&(V1,1,1,1,1+V1,1,2,2,1+V1,1,3,1,1)
&(V1,1,1,1,1+V1,1,2,2,1+V1,1,3,2,1)
&(V1,1,1,1,1+V1,1,2,2,1+V1,1,3,3,1)
&(V1,1,1,1,1+V1,1,2,3,1+V1,1,3,1,1)
&(V1,1,1,1,1+V1,1,2,3,1+V1,1,3,2,1)
&(V1,1,1,1,1+V1,1,2,3,1+V1,1,3,3,1)
&(V1,1,1,2,1+V1,1,2,1,1+V1,1,3,1,1)
&(V1,1,1,2,1+V1,1,2,1,1+V1,1,3,2,1)
&(V1,1,1,2,1+V1,1,2,1,1+V1,1,3,3,1)
&(V1,1,1,2,1+V1,1,2,2,1+V1,1,3,1,1)
&(V1,1,1,2,1+V1,1,2,2,1+V1,1,3,2,1)
&(V1,1,1,2,1+V1,1,2,2,1+V1,1,3,3,1)
&(V1,1,1,2,1+V1,1,2,3,1+V1,1,3,1,1)
&(V1,1,1,2,1+V1,1,2,3,1+V1,1,3,2,1)
&(V1,1,1,2,1+V1,1,2,3,1+V1,1,3,3,1)
&(V1,1,1,3,1+V1,1,2,1,1+V1,1,3,1,1)
&(V1,1,1,3,1+V1,1,2,1,1+V1,1,3,2,1)
&(V1,1,1,3,1+V1,1,2,1,1+V1,1,3,3,1)
&(V1,1,1,3,1+V1,1,2,2,1+V1,1,3,1,1)
&(V1,1,1,3,1+V1,1,2,2,1+V1,1,3,2,1)
&(V1,1,1,3,1+V1,1,2,2,1+V1,1,3,3,1)
&(V1,1,1,3,1+V1,1,2,3,1+V1,1,3,1,1)
&(V1,1,1,3,1+V1,1,2,3,1+V1,1,3,2,1)
&(V1,1,1,3,1+V1,1,2,3,1+V1,1,3,3,1)
&(V1,2,1,1,1+V1,2,2,1,1+V1,2,3,1,1)
&(V1,2,1,1,1+V1,2,2,1,1+V1,2,3,2,1)
&(V1,2,1,1,1+V1,2,2,1,1+V1,2,3,3,1)
&(V1,2,1,1,1+V1,2,2,2,1+V1,2,3,1,1)
&(V1,2,1,1,1+V1,2,2,2,1+V1,2,3,2,1)
&(V1,2,1,1,1+V1,2,2,2,1+V1,2,3,3,1)
&(V1,2,1,1,1+V1,2,2,3,1+V1,2,3,1,1)
&(V1,2,1,1,1+V1,2,2,3,1+V1,2,3,2,1)
&(V1,2,1,1,1+V1,2,2,3,1+V1,2,3,3,1)
&(V1,2,1,2,1+V1,2,2,1,1+V1,2,3,1,1)
&(V1,2,1,2,1+V1,2,2,1,1+V1,2,3,2,1)
&(V1,2,1,2,1+V1,2,2,1,1+V1,2,3,3,1)
&(V1,2,1,2,1+V1,2,2,2,1+V1,2,3,1,1)
&(V1,2,1,2,1+V1,2,2,2,1+V1,2,3,2,1)
&(V1,2,1,2,1+V1,2,2,2,1+V1,2,3,3,1)
&(V1,2,1,2,1+V1,2,2,3,1+V1,2,3,1,1)
&(V1,2,1,2,1+V1,2,2,3,1+V1,2,3,2,1)
&(V1,2,1,2,1+V1,2,2,3,1+V1,2,3,3,1)
&(V1,2,1,3,1+V1,2,2,1,1+V1,2,3,1,1)
&(V1,2,1,3,1+V1,2,2,1,1+V1,2,3,2,1)
&(V1,2,1,3,1+V1,2,2,1,1+V1,2,3,3,1)
&(V1,2,1,3,1+V1,2,2,2,1+V1,2,3,1,1)
```

```
&(V1,2,1,3,1+V1,2,2,2,1+V1,2,3,2,1)
&(V1,2,1,3,1+V1,2,2,2,1+V1,2,3,3,1)
&(V1,2,1,3,1+V1,2,2,3,1+V1,2,3,1,1)
&(V1,2,1,3,1+V1,2,2,3,1+V1,2,3,2,1)
&(V1,2,1,3,1+V1,2,2,3,1+V1,2,3,3,1)
&(V1,3,1,1,1+V1,3,2,1,1+V1,3,3,1,1)
&(V1,3,1,1,1+V1,3,2,1,1+V1,3,3,2,1)
&(V1,3,1,1,1+V1,3,2,1,1+V1,3,3,3,1)
&(V1,3,1,1,1+V1,3,2,2,1+V1,3,3,1,1)
&(V1,3,1,1,1+V1,3,2,2,1+V1,3,3,2,1)
&(V1,3,1,1,1+V1,3,2,2,1+V1,3,3,3,1)
&(V1,3,1,1,1+V1,3,2,3,1+V1,3,3,1,1)
&(V1,3,1,1,1+V1,3,2,3,1+V1,3,3,2,1)
&(V1,3,1,1,1+V1,3,2,3,1+V1,3,3,3,1)
&(V1,3,1,2,1+V1,3,2,1,1+V1,3,3,1,1)
&(V1,3,1,2,1+V1,3,2,1,1+V1,3,3,2,1)
&(V1,3,1,2,1+V1,3,2,1,1+V1,3,3,3,1)
&(V1,3,1,2,1+V1,3,2,2,1+V1,3,3,1,1)
&(V1,3,1,2,1+V1,3,2,2,1+V1,3,3,2,1)
&(V1,3,1,2,1+V1,3,2,2,1+V1,3,3,3,1)
&(V1,3,1,2,1+V1,3,2,3,1+V1,3,3,1,1)
&(V1,3,1,2,1+V1,3,2,3,1+V1,3,3,2,1)
&(V1,3,1,2,1+V1,3,2,3,1+V1,3,3,3,1)
&(V1,3,1,3,1+V1,3,2,1,1+V1,3,3,1,1)
&(V1,3,1,3,1+V1,3,2,1,1+V1,3,3,2,1)
&(V1,3,1,3,1+V1,3,2,1,1+V1,3,3,3,1)
&(V1,3,1,3,1+V1,3,2,2,1+V1,3,3,1,1)
&(V1,3,1,3,1+V1,3,2,2,1+V1,3,3,2,1)
&(V1,3,1,3,1+V1,3,2,2,1+V1,3,3,3,1)
&(V1,3,1,3,1+V1,3,2,3,1+V1,3,3,1,1)
&(V1,3,1,3,1+V1,3,2,3,1+V1,3,3,2,1)
&(V1,3,1,3,1+V1,3,2,3,1+V1,3,3,3,1)
```