This section contains exercises for using SAT and bit-vector encodings into Z3. You can use bit-vectors directly to save energy for encoding common idioms that can otherwise directly be encoded in SAT.

1 \( n \)-queens

The classical \( n \)-queens puzzle is to place \( n \) queens on an \( n \times n \) chess-board so that no two queens attack each other. The exercise asks you to encode an \( n \) queens placement problem and then have Z3 enumerate solutions. The following text contains a guided walk through of how this can be accomplished. You can try your own encoding ideas as well.

1.1 Placing queens on a chess-board

1. We will allocate an \( n \) bit bit-vector per row. So let \( r_1, \ldots, r_n \) be \( n \) \( n \)-bit bitvectors. In SMT-LIB the declaration for \( r_1, \ldots, r_8 \) looks as follows:

\[
\begin{align*}
(\text{declare-const } r1 (_\text{ BitVec 8})) \\
(\text{declare-const } r2 (_\text{ BitVec 8})) \\
(\text{declare-const } r3 (_\text{ BitVec 8})) \\
(\text{declare-const } r4 (_\text{ BitVec 8})) \\
(\text{declare-const } r5 (_\text{ BitVec 8})) \\
(\text{declare-const } r6 (_\text{ BitVec 8})) \\
(\text{declare-const } r7 (_\text{ BitVec 8})) \\
(\text{declare-const } r8 (_\text{ BitVec 8}))
\end{align*}
\]

2. Each vector \( r_i \) should have at most one bit set. There are various ways of encoding this. The simplest is to create \((n+12)\) axioms per row of the chessboard. For example, for row \( r_1 \) one can assert the axioms:

\[
\begin{align*}
r1[7] & \rightarrow \neg r1[6], r1[7] \rightarrow \neg r1[5], \ldots \\
r1[7] & \rightarrow \neg r1[0], \ldots \\
r1[6] & \rightarrow \neg r1[5], r1[6] \rightarrow \neg r1[4], \ldots \\
r1[6] & \rightarrow \neg r1[0], \ldots
\end{align*}
\]

We use the notation \( r1[7] \) for accessing the most significant bit of \( r_1 \). The other bits are at positions 0 to 6. This is not the most succinct way, however. Consider the formula

\[
\text{bv0}[8] = (r1&(_r1 - 1))
\]

where \( \text{bv0}[8] \) is a bit-vector of length 8 consisting of all zeros. It says that taking the bit-wise and of \( r_1 \) and \( r_1 - 1 \) results in 0. The arithmetical circuit for this formula is much smaller.

3. Now consider the columns; for each column \( k \) there should be exactly one row \( r_i \), such that \( r_i[k] \) is set. How would you encode this using as few constraints as possible?
4. Finally consider the diagonals. Also at most one bit should be set on diagonals. How can you express succinctly that in each diagonal there is at most one bit set?

5. Write a program that takes a number \( n \) as input and generates a problem in SMTLIB format for the \( n \)-queens placement.

6. Write a program that takes a number \( n \) as input, uses the Z3 API to enumerate placements of queens.

### 1.2 Enumerating solutions

This section applies if you are using the programmatic API. Suppose we wish to enumerate several solutions for \( n \)-queens. We would need to block previous solutions when resuming search. For this purpose Z3 exposes models that assigns values to variables. One can take the values and construct new formulas to block with. With the managed API, the relevant calls are:

- \( \text{LBool result} = \text{z3.CheckAndGetModel(ref model)}; \) If the current context is satisfiable, then the result is \( \text{LBool.True} \). The reference argument `model` is populated with an object containing the satisfiable assignment.

- \( \text{int r1 val} = \text{model.GetNumeralValueInt(model.Eval(r1))} \) retrieves the numeric value assigned to row \( r1 \).

- \( \text{Term eq1} = \text{z3.MkEq(z3.MkNumeral(r1 val, row type), r1);} \) is the equation stating that \( r1 \) has value \( r1 \text{ val} \).

- \( \text{z3.AssertCnstr(z3.MkNot(z3.MkAnd(assignment))}); \) If `assignment` is an array containing the current assignment to the variables as equalities, then the current assignment gets blocked by asserting the negation of these.

### 1.3 Symmetry reduction

We might not care about enumerating solutions that are symmetric. That is, we don’t need to enumerate solutions that can be obtained by turning the chess-board around. A symmetry breaking predicate is an additional constraint that restricts the search space by ruling out symmetric alternative solutions.

1. Find a symmetry breaking predicate that reduces the placements on the first row.

2. Can you think of other symmetry breaking predicates for \( n \)-queens?

### 2 Longest path

The shortest path between two nodes in a graph is a classical graph algorithm problem. Dijkstra’s algorithm uses a heap to solve the problem in \( O(n \log n) \); and when edge weights are integers, there are even more efficient solutions. To find a longest path between two nodes in a directed graph is on the other hand an NP complete problem. In this exercise we will convert longest path problems into a Boolean constraint satisfaction problem.
2.1 Primes and Grey codes

We first need a graph. Of course there are many graphs to pick from. The graph we use here is constructed in a peculiar way. Let $V$ be the set of primes between 1000 and 10000. We connect two nodes in $v, w \in V$, such that $v \rightarrow w$, iff the decimal representation of $v$ and $w$ has one digit differing by only one. A sample program for generating the graph, courtesy of Utkarsh Upadhyay, is provided below.

```ocaml
let rec sieve (arr:int array ref) step idx
  if idx >= (!arr).Length then () else
  ((!arr).[idx] <- 0; sieve arr step (step+idx))

let main from till =
  let numbers = ref (Array.init (till+1) (fun i -> i)) in
  (!numbers).[1] <- 0;
  for ii = 2 to int.ceil(sqrt(float(till)))) do
    if (!numbers).[ii]>0 then sieve numbers ii (2*ii);
  done;
  Array.filter
  (fun v -> v>0)
  (Array.sub (!numbers) from (till-from+1))

let get_pairs from till =
  let primes = main from till in
  let rec grey diff =
    if diff=0 then false // Same number.
    elsif diff=1 then true
    elsif diff%10 <> 0 then false
    else diff / 10 |> grey
    in
  let pairs = ref [] in
  for i = 0 to (primes.Length-1) do
    for j = 0 to (primes.Length-1) do
      if (grey (abs(primes.[j]-primes.[i]))) then
        pairs := (i, j)::(!pairs);
    done;
  done;
  (primes, !pairs)

let prime_index, prime_conn = get_pairs 1000 10000
```

2.2 Encoding the graph

The next task we have to face is how one can encode a graph and a path finding problem. So far we have a graph $G = (V,E)$, where $V$ is a set of $n$ vertices, and $E$ is a set of $m$ edges. We now sketch one possible encoding of the problem. Suppose the set of vertices $V$ is $\{v_1, ..., v_n\}$, then associate $n$ bit-vectors $ord_1, ..., ord_n$, each of $\lceil \log n \rceil$ bits (the nearest natural number that is greater than or equal to $\log n$). We will refer to these bit-vectors as ordinals. The purpose of the ordinals is to guess an ordering of each vertex, a prefix of the ordering will correspond to a path in the graph. This can be encoded by
Hands on Z3

![Sample Sudoku Board]

Figure 1: Sample Sudoku Board

asserting for each vertex \( v_i \), and for each number \( k = 1, \ldots, n \):

\[
((\text{ord}_i \simeq k) \land (k < \text{max}\_\text{path})) \rightarrow \bigvee_{v_j \in V} ((v_i, v_j) \in E \land \text{ord}_j = k + 1)
\]

The bit-vector max path is the length of the maximal path in \( G \). We also require that some vertex has ordinal 0:

\[
\bigvee_{v_j \in V} \text{ord}_j \simeq 0
\]

There is a problem with the encoding.

- What is the practical problem, assuming there are around 10^64 vertices?
- Find a compact encoding.
- Write a program that converts the graph prime index, prime conn into a constraint for Z3.
- Write a wrapper around CheckAndGetModel to find the maximal value of max path.

### 2.3 Sudoku

The popular Sudoku puzzle is to place numbers 1-9 on a \( 9 \times 9 \) board, such that each number occurs only once in every row and column. Furthermore, if you divide the board into 9 sub-grids each of size \( 3 \times 3 \), then each of these sub-grids should also be covered by different numbers. Sudoku puzzles come with boards that have been partially occupied with numbers. The puzzle is to occupy the remaining fields in such a way that the constraints on rows, columns, and sub-grids are satisfied. A sample Sudoku problem is given in Figure 1.

1. Assume that Sudoku problems are given in the form:

```
..1.2...  .6.93...  3...814.  .9...83.  ..4.7.6...  .37...2.  .159...2  .48.5...  ...1.3...
```

2. Write a program that converts Sudoku problems into:

   (a) an SMT formula using Bit-vectors,
   (b) an SMT formula using integers,
   (c) calls over the Z3 binary API, and use it to enumerate solutions.

3. Test the program on your favorite puzzle. Note: There is a shortcut on [http://modante.googlepages.com/sudokusolver](http://modante.googlepages.com/sudokusolver).

3 Puzzle solving using LINQ


4 An assignment problem using bit-vectors

We got the following question on the Z3 bug mailing list from a student in Singapore. We reformulate it to guide you through an encoding that uses bit-vectors.

Assume there \( n \) persons and \( m \) locations. Each person is associated with a number of locations. You are now provided a number \( k < \min(n,m) \), and your problem is to formulate an encoding that selects \( k \) persons and \( k \) locations, such that all locations associated with a selected person is among the selected locations. Also each of the selected locations is associated with one of the selected persons.

First let us assume that there are 4 persons and 5 locations, with the original assignment:

Person1: location1, location3, location5
Person2: location2, location3
Person3: location1, location2
Person4: location2, location3

4.1 A high-level encoding

To warm up with, we will encode the assignment problem in first-order logic. For this purpose, here is a suggested approach:

- Introduce a binary predicate \( P(person, location) \) that encodes when persons are associated with locations.
- Introduce another binary predicate \( S \) that encodes a selection.
- Introduce logical constraints on \( S \) and \( P \) to enforce that
  1. All locations associated with a selected person is among the selected locations
  2. Each of the selected locations is associated with one of the selected persons.
  3. The number of selected persons is \( k \).
  4. The number of selected locations is \( k \).
4.2 An encoding using bit-vectors

We will assume that there are \( n \) bit-vectors each of length \( m \) that specifies if person \( i \) is associated with location \( j \).

Let us write the assignment using bit-vectors:

\[
\begin{align*}
(\text{define-const } & p1 \; (_\text{BitVec} \; 5) \; \#b10101) \\
(\text{define-const } & p2 \; (_\text{BitVec} \; 5) \; \#b00110) \\
(\text{define-const } & p3 \; (_\text{BitVec} \; 5) \; \#b00011) \\
(\text{define-const } & p4 \; (_\text{BitVec} \; 5) \; \#b00110)
\end{align*}
\]

If \( k \) is 3, then a valid selection is \( \{ \text{Person2, Person3, Person4}\} \).

Let us in the following assume that \( n = 10, m = 12 \) and \( k = 8 \), and consider the (randomly generated) sample data to test your solution:

\[
\begin{align*}
(\text{define-fun } & p1 \; () \; (_\text{BitVec} \; 12) \; \#b010100011010) \\
(\text{define-fun } & p2 \; () \; (_\text{BitVec} \; 12) \; \#b011010000010) \\
(\text{define-fun } & p3 \; () \; (_\text{BitVec} \; 12) \; \#b011000100000) \\
(\text{define-fun } & p4 \; () \; (_\text{BitVec} \; 12) \; \#b010001000000) \\
(\text{define-fun } & p5 \; () \; (_\text{BitVec} \; 12) \; \#b010010000000) \\
(\text{define-fun } & p6 \; () \; (_\text{BitVec} \; 12) \; \#b000010110000) \\
(\text{define-fun } & p7 \; () \; (_\text{BitVec} \; 12) \; \#b000110001010) \\
(\text{define-fun } & p8 \; () \; (_\text{BitVec} \; 12) \; \#b000001101010) \\
(\text{define-fun } & p9 \; () \; (_\text{BitVec} \; 12) \; \#b010001000000) \\
(\text{define-fun } & p10 \; () \; (_\text{BitVec} \; 12) \; \#b001111101010)
\end{align*}
\]

Here is one possible approach:

1. Introduce a bit-vector \( \text{selected} \) of length \( n \) representing the selected persons.

2. Write a macro (\text{define-fun}) that computes the bit-wise or of bit-vectors \( p1 \ldots p10 \) based on the persons that are selected.

3. Write a macro that computes the sum of selected bits in a bit-vector of length 10 and 12 respectively. You are advised to use bit-vector arithmetic and guard against overflow. Bit-vector addition can overflow if the most significant carry bit is 1. You can guard against overflow by zero-extending bit-vectors.

Some of the following bit-vector functions may become handy:

- \( \text{bvor} \) bit-wise or

- \( \text{bvadd} \) bit-vector addition

- \( (_\text{bv0} \; 12) \) create a zero bit-vector of length 12.

- \( (_\text{extract} \; i \; j) \) extract sub-range \( i \geq j \) from a bit-vector.

- \( \#b1 \) the bit-vector 1 of length 1.

- \( \#b0 \) the bit-vector 0 of length 1.

- \( (_\text{zero_extend} \; 5) \) zero extend a given bit-vector with 5 bits.
4.3 It is also a pseudo-Boolean problem

The use of bit-vectors fixes a certain solution method. A somewhat higher-level encoding method is to use integer arithmetic. Since we only add 0-1 variables, the resulting problem is a pseudo-Boolean problem.

1. Encode the problem using integer linear arithmetic.