```
In[75]:= Get@FileNameJoin[{DirectoryName@NotebookFileName[], "Z3Interop.wl"}]
```

```
In[2]:= $ContextPath = DeleteDuplicates@Append[$ContextPath, "Z3Interop`Nonogram`"];
```

Parse out the structure

Here follows some ghastly, though at least pure, XLSX-parsing code.

```
In[3]:= Clear[fixColour];
        fixColour["FFFF0000"] = Red;
        fixColour["FFFFFF00"] = Yellow;
        fixColour["FFFFFFFF"] = Black;
        fixColour["FF0070C0"] = Blue;
        fixColour[x_] := Throw[x];
```

```
In[9]:= extractSi[XMLElement["si", {}, xs_List]] := Select[
          If[Length[#[[3]]] == 1, {fixColour@"FFFFFFFF", First@Cases[#,
                XMLElement["t", _, {x_}] ⧴ (FromDigits[#, 10] & /@ StringSplit[x]), All]},
             With[{style = Cases[#[[3, 1]], XMLElement["rPr", {},
                    {___, XMLElement["color", {"rgb" → r_}, ___], ___}] ⧴ r, All],
               text = Cases[#[[3, 2]], XMLElement["t", _, {t_}] ⧴ t, All]},
              {If[style === {}, fixColour@"FFFFFFFF", Assert[Length@style] == 1;
                fixColour@First@style], Assert[Length@text == 1];
               FromDigits[#, 10] & /@ StringSplit[First@text]}]] & /@ xs,
          Length@Last[#] > 0 &]
```

```
In[10]:= xlsx = FileNameJoin[{DirectoryName@NotebookFileName[], "nauseator.xlsx"}];
```

```
In[11]:= extracted = extractSi /@ Import[xlsx, {"ZIP", "xl/sharedStrings.xml"}][[2, 3]];
```

```
In[12]:= colStringIds = 1 + FromDigits[#, 10] &@* First@* Last@* First@* Last /@ Most@
            Rest[Import[xlsx, {"ZIP", "xl/worksheets/sheet1.xml"}][[2]][[3, 5, 3, 1, 3]]]

Out[12]= {1, 2, 3, 4, 92, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
          22, 23, 24, 25, 26, 27, 28, 29, 93, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
          40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58}
```

```
In[13]:= rowStringIds = 1 + FromDigits[#[[3, 1, 3, 1, 3, 1]], 10] & /@
           Import[xlsx, {"ZIP", "xl/worksheets/sheet1.xml"}][[2]][[3, 5, 3, 2 ;; -2]]

Out[13]= {59, 60, 94, 95, 96, 61, 62, 63, 97, 64, 65, 66, 98, 99, 67, 68, 100, 101, 102, 103, 69,
          104, 105, 106, 107, 108, 109, 110, 111, 70, 112, 113, 114, 115, 116, 117, 71, 72, 73,
          74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 118, 119, 89, 90, 91, 120}
```

```
In[14]:= rowsIn = Map[Reverse, Flatten[#, 1] & /@
             Map[Thread, Extract[extracted, List /@ rowStringIds], {2}], {2}];
```

```
In[15]:= colsIn = Map[Reverse, Flatten[#, 1] & /@
             Map[Thread, Extract[extracted, List /@ colStringIds], {2}], {2}];
```

For example:

```
In[16]:= colsIn[[1]]

Out[16]= {{4, 🟥}, {2, ⬛}, {2, 🟦}, {5, 🟦}, {1, 🟥}, {4, 🟥}, {4, 🟥}, {12, ⬛}}
```

Construct a system of constraints

For more detail, see the z3interop.nb example notebook.

Define an arbitrary mapping of colours to numbers, so that we can represent the problem in integers.

```
In[17]:= mapping = With[{colours = Union@Cases[rowsIn, _?ColorQ, All]},
           Assert[FreeQ[colours, White]];
           MapIndexed[#1 → First@#2 &, Append[colours, White]]]

Out[17]= {⬛ → 1, 🟦 → 2, 🟥 → 3, 🟨 → 4, ⬜ → 5}
```

```
In[18]:= constrainedColumns :=
           MapIndexed[gapsToConstraints[#1, First@#2, Length@rowsIn, colGap] &, colsIn];
```

```
In[19]:= constrainedRows :=
           MapIndexed[gapsToConstraints[#1, First@#2, Length@colsIn, rowGap] &, rowsIn];
```

```
In[20]:= additionalConstraints := constrainedCells[rowGap, colGap, cell,
            rowsIn, colsIn, constrainedRows, constrainedColumns, mapping];
```

Form the program:

```
In[21]:= vars := DeleteDuplicates@
            Flatten@{Cases[{constrainedCells, constrainedColumns, constrainedRows},
                colGap[_, _], Infinity], Cases[{additionalConstraints, constrainedColumns,
                 constrainedRows}, rowGap[_, _], Infinity], Cases[{additionalConstraints,
                 constrainedColumns, constrainedRows}, cell[_, _], Infinity]};
```

```
In[22]:= constraints := Assertion /@
           Flatten[{additionalConstraints, constrainedColumns, constrainedRows}]
```

```
In[23]:= declared := Declare[#, Integer] & /@ vars
```

```
In[24]:= symbols = {colGap → "colGap", rowGap → "rowGap", cell → "cell"};
```

```
In[25]:= program := Riffle[toString[symbols, #] & /@
               Flatten@{declared, constraints, CheckSat, GetModel}, "\n"] // StringJoin;
```

This is an example where the built-in Z3Interop `toString` does not know how to perform addition. Teach it:

```
In[26]:= toString[symbols_, a_ + b_] :=
          StringJoin["(+ ", toString[symbols, a], " ", toString[symbols, b], ")"]
```

Write and run the program:

```
In[27]:= s = OpenWrite[FormatType → OutputForm, PageWidth → Infinity];
         Write[s, program];
         outputLocation = Close[s]

Out[27]= /private/var/folders/hz/9prp92151cqgf8370qt8ngfw0000gn/T/m00000685231
```

```
In[28]:= output = RunProcess[{"z3", outputLocation},
              ProcessEnvironment → <|"PATH" → "/usr/local/bin"|>]["StandardOutput"];
```

```
In[29]:= StringCases[output, RegularExpression["(un)?sat"]]

Out[29]= {sat}
```

Parse out the solution (sorry about the rendered PDF being so blurry):

```
In[30]:= answer = getDefinitions[symbols, output];
```

```
In[77]:= Table[cell[i, j], {i, 1, Length@rowsIn}, {j, 1, Length@colsIn}] /. answer /.
            (Reverse /@ mapping) // Image

Out[77]=
```