Sudoku 数独

https://github.com/rdi-berkeley/zkp-course-lecture3-code

> 为简便起见, 本次只实现了对 `Row` 的 `Distinct` 判断, 对于 9 宫格 和 Column 没有做限制

```
pragma circom 2.0.0;

template NonEqual(){
    signal input in0;
    signal input in1;
    signal inv;
    inv <-- 1/ (in0 - in1);
    inv*(in0 - in1) === 1;
}

// all elements are unique in the array.
template Distinct(n) {
    signal input in[n];
    component nonEqual[n][n];
    for(var i = 0; i < n; i++){
        for(var j = 0; j < i; j++){
            nonEqual[i][j] = NonEqual();
            nonEqual[i][j].in0 <== in[i];
            nonEqual[i][j].in1 <== in[j];
        }
    }
}

// Enforce that 0 <= in < 16
template Bits4(){
    signal input in;
    signal bits[4];
    var bitsum = 0;
    for (var i = 0; i < 4; i++) { // 逐位取出 in 的二进制位, 放在 bits 里
        bits[i] <-- (in >> i) & 1;
        bits[i] * (bits[i] - 1) === 0;
        bitsum = bitsum + 2 ** i * bits[i];
    }
    bitsum === in;
}

// Enforce that 1 <= in <= 9
template OneToNine() {
    signal input in;
    component lowerBound = Bits4();
    component upperBound = Bits4();
    lowerBound.in <== in - 1;
    upperBound.in <== in + 6;
}

template Sudoku(n) {
    // solution is a 2D array: indices are (row_i, col_i)
    signal input solution[n][n];
    // puzzle is the same, but a zero indicates a blank
    signal input puzzle[n][n];

    component distinct[n];   // 先声明组件数组, 之后在循环中具体定义
    component inRange[n][n]; // 先声明组件数组, 之后在循环中具体定义

    for (var row_i = 0; row_i < n; row_i++) {
        for (var col_i = 0; col_i < n; col_i++) {
            // we could make this a component
            // puzzle_cell * (puzzle_cell - solution_cell) = 0;
            puzzle[row_i][col_i] * (puzzle[row_i][col_i] - solution[row_i][col_i]) === 0;
        }
```

```
    }
    // ensure uniqueness in Rows.
    // ensure that each solution # is in-range.
    for (var row_i = 0; row_i < n; row_i++) {
        for (var col_i = 0; col_i < n; col_i++) {
            if (row_i == 0) {
                distinct[col_i] = Distinct(n);
            }
            inRange[row_i][col_i] = OneToNine();
            inRange[row_i][col_i].in <== solution[row_i][col_i];
            distinct[col_i].in[row_i] <== solution[row_i][col_i];
        }
    }
}

component main {public[puzzle]} = Sudoku(9);
```

Modules

1. 模板 NonEqual 定义了判断两个数是否不相等的模块。
2. 模板 Distinct 利用 NonEqual 模块检查一个数组中的元素是否都不相等。
3. 模板 Bits4 将一个数拆成 4 个 bit, 并检查其范围在 0 到 15 之间
4. 模板 OneToNine 利用 Bits4 检查一个数是否在 1 到 9 之间。
5. 模板 Sudoku 定义了数独解题的主要逻辑

NonEqual

```
template NonEqual(){
    signal input in0;
    signal input in1;
    signal inv;
    inv <-- 1/ (in0 - in1);
    inv*(in0 - in1) === 1;
}
```

当 `in0 = in1` 时，这个电路是如何工作的? $\frac{1}{(in0-in1)}$ ，为什么可以除 0？

GPT: `1 / (in0 - in1)` 确实会导致一个**数学上的不确定性**（通常情况下，除数不能为零）。然而，对于零知识证明的约束系统，这种情况可以得到处理。在这个案例中，如果 `in0 - in1` 等于零，那么 **`inv` 的值就会被定义为任意的（可以是任何值）**，而约束 `inv * (in0 - in1) === 1` 就不会被满足。这就实现了 "NonEqual" 的功能，也就是说，如果 `in0` 和 `in1` 相等，那么这个约束就会导致整个电路无效。

Distinct

```
// all elements are unique in the array.
template Distinct(n) {
    signal input in[n];
    component nonEqual[n][n];
    for(var i = 0; i < n; i++){
        for(var j = 0; j < i; j++){
            nonEqual[i][j] = NonEqual();
            nonEqual[i][j].in0 <== in[i];
            nonEqual[i][j].in1 <== in[j];
        }
    }
}
```

1. 实例化了 `n` 个 NonEqual 组件, 名称是 `nonEqual[i][j]`
2. 然后在两个 `for` 循环中连接所有 `nonEqual` 实例, 任意两数 `in[i]` 和 `in[j]` 都会传递给一个 `NonEqual` 实例
3. `NonEqual` 会判断两数是否相等
4. 如果任意两个数相等,都会导致 `NonEqual` 中的约束不满足
5. 所以所有 `NonEqual` 的约束都满足的情况下, 证明了 `in` 数组中的元素都是不相等的
6. 通过这种方式, `Distinct` 利用和 `NonEqual` 的组合,实现了判断一个数组是否所有元素都不相等

Bits4 & OneToNine

`Bits4` 可以将一个 数 转为 4 位二进制, 4 为二进制能表达的数字范围是 `0 ~ 15`
`Bits4` 的主要作用是服务于 `template OneToNine` , 来将限制数字范围在 `1 ~ 9`

- `lowerBound.in <== in - 1;` 如果 in <= 0 就越界了
- `upperBound.in <== in + 6;` 如果 in >= 10 就越界了

```
// Enforce that 0 <= in < 16
template Bits4(){
    signal input in;
    signal bits[4];
    var bitsum = 0;
    for (var i = 0; i < 4; i++) {  // 逐位取出 in 的二进制位, 放在 bits 里
        bits[i] <-- (in >> i) & 1;
        bits[i] * (bits[i] - 1) === 0;
        bitsum = bitsum + 2 ** i * bits[i];
    }
    bitsum === in;
}

// Enforce that 1 <= in <= 9
template OneToNine() {
    signal input in;
    component lowerBound = Bits4();
    component upperBound = Bits4();
    lowerBound.in <== in - 1; // 赋值后约束
    upperBound.in <== in + 6;
}
```

1. Bits4 有一个输入信号 `in`
2. 定义了一个 4 位的 `bits` 数组
3. 在 `for` 循环中, 提取 `in` 的每一位到 `bits` 数组中

关于 `bits[i] <-- (in >> i) & 1;`

1. `in >> i` 对 `in` 进行右移 `i` 位的运算, 如果 `in` 是 `0111` (即 7), 那么 `in >> 1` 即右移一位 会得到 `0011` , 右移 2 位得到 `0001`
2. 最后与 `1` 进行按位与运算(&),可以检查最后一位是否为 1
3. 如果最后一位是 1 , 则 `(in >> i) & 1` 的结果为 `1` , 如果最后一位是 0 , 则结果为 0
4. 由于右移后最后一位就是原本的第 `i` 位, 所以 `(in >> i) & 1` 可以提取出 `in` 的第 `i` 位 bit
5. 重复该过程,可以获取 `in` 的每一位比特, 存入 `bits` 数组。
6. 对每一位添加约束: `bits[i] * (bits[i] - 1) === 0;` 这强制 `bits[i]` 只能是 0 或 1

Sudoku (主逻辑)

```
template Sudoku(n) {
    signal input solution[n][n];
    signal input puzzle[n][n];

    component distinct[n];   // 先声明组件数组, 之后在循环中具体定义
    component inRange[n][n]; // 先声明组件数组, 之后在循环中具体定义

    for (var row_i = 0; row_i < n; row_i++) {
        for (var col_i = 0; col_i < n; col_i++) {
            puzzle[row_i][col_i] * (puzzle[row_i][col_i] - solution[row_i][col_i]) === 0;  // 约束 ①
        }
    }
    // ensure uniqueness in Rows.
    // ensure that each solution # is in-range.
    for (var row_i = 0; row_i < n; row_i++) {
        for (var col_i = 0; col_i < n; col_i++) {
            if (row_i == 0) {
                distinct[col_i] = Distinct(n);  // 组件 ②
            }
            inRange[row_i][col_i] = OneToNine();
            inRange[row_i][col_i].in <== solution[row_i][col_i]; //约束 ③
            distinct[col_i].in[row_i] <== solution[row_i][col_i];//约束 ④
        }
    }
}
```

约束 ① ∵ puzzle 中 的 0 表示空白格, 所以可首先添加约束 : 原本的 puzzle 和 solution 对应位置必须相等 (限制必须依照 **原题** 进行求解)

组件 ② 遍历第 0 行, 对每列加一个 `Distinct(n)` , 去 Check 这一列里都是 `1~9` 的不同数字

约束 ③ 约束 solution 里的数字必须在 `1~9`

约束 ④ 约束 solution 里(本列)数字必须各个不同

input

```
{
  "solution":
    [
      ["1","9","4","8","6","5","2","3","7"],
      ["7","3","5","4","1","2","9","6","8"],
      ["8","6","2","3","9","7","1","4","5"],
      ["9","2","1","7","4","8","3","5","6"],
      ["6","7","8","5","3","1","4","2","9"],
      ["4","5","3","9","2","6","8","7","1"],
      ["3","8","9","6","5","4","7","1","2"],
      ["2","4","6","1","7","9","5","8","3"],
      ["5","1","7","2","8","3","6","9","4"]
    ],
  "puzzle":
    [
      ["0","0","0","8","6","0","2","3","0"],
      ["7","0","5","0","0","0","9","0","8"],
      ["0","6","0","3","0","7","0","4","0"],
      ["0","2","0","7","0","8","0","5","0"],
      ["0","7","8","5","0","0","0","0","0"],
      ["4","0","0","9","0","6","0","7","0"],
      ["3","0","9","0","5","0","7","0","2"],
      ["0","4","0","1","0","9","0","8","0"],
      ["5","0","7","0","8","0","0","9","4"]
    ]
}
```

Rust version