### From Checker to Solver

**Tactics to Pseudocode to Code**

and: Short Circuit Evaluation
(at no extra charge)

---

<table>
<thead>
<tr>
<th>2</th>
<th>6</th>
<th>7</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>8</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>_</td>
<td>4</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>_</td>
<td>5</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>_</td>
<td>9</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>_</td>
<td>4</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>_</td>
<td>_</td>
</tr>
</tbody>
</table>

---

\{1, 2, 3, 4, 5, 6, 7, 8, 9\}

---

<table>
<thead>
<tr>
<th>2</th>
<th>6</th>
<th>7</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>8</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>_</td>
<td>4</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>_</td>
<td>5</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>_</td>
<td>9</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>_</td>
<td>4</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>_</td>
<td>_</td>
</tr>
</tbody>
</table>

---

\{1, 2, 3, 4, 5, 6, 7, 8, 9\}
\[
\begin{array}{cccc}
_ & _ & _ & 2 \\
6 & 8 & _ & 7 \\
1 & 9 & _ & _ \\
8 & 2 & 1 & _ \\
_ & 4 & 6 & 2 \\
_ & 5 & _ & 3 \\
_ & 9 & 3 & _ \\
_ & 4 & 5 & _ \\
7 & 3 & 1 & 8 \\
\end{array}
\]

\[
\begin{array}{cccc}
_ & _ & _ & 2 \\
6 & 8 & _ & 7 \\
1 & 9 & _ & _ \\
8 & 2 & 1 & _ \\
_ & 4 & 6 & 2 \\
_ & 5 & _ & 3 \\
_ & 9 & 3 & _ \\
_ & 4 & 5 & _ \\
7 & 3 & 1 & 8 \\
\end{array}
\]
We can find standard solution tactics on the web ...

SadMan Software
Sudoku Technique - Naked Single

Naked Single (Singleton, Sole Candidate)
Whilst solving Sudoku puzzles, it is often the case that a cell can only possibly take a single value, when the contents of the other cells in the same row, column and block are considered. This is where, between them, the row, column and block use eight different digits, leaving only a single digit available for the cell.

For example, in the partial Sudoku puzzle below, the marked cell can only be a 5. All other digits are excluded by the contents of the other cells in the row, column and block.

... and translate to design, then pseudocode, then code.

“Naked single” (aka “sole candidate”)

Human tactic:
Start with 9 candidates in each cell
Cross off candidates used in the row, column, and block
If there is only one candidate left in a cell, choose it

Program tactic:
Each cell has a “possible” attribute, a Python set
For each row, column, and block
Loop through the group to find used symbols
Loop again to remove used symbols from “possible”
If len(possible) is 1, choose that symbol

Keeping an invariant

It helps to establish an object invariant:

At all times, the tile.possible includes all possible symbols for the tile
Tile.possible may become smaller (lose elements) when we discover constraints, but it never becomes larger
If tile.possible has only one element, then tile.symbol is that element

Every method and function maintains the invariant

Pseudocode

Program tactic: “naked single” or “sole candidate”
Each cell has a “possible” attribute, a Python set
For each row, column, and block
Loop through the group to find used symbols
Loop again to remove used symbols from “possible”
If len(possible) is 1, choose that symbol

Pseudocode for a single group (row, column, or block):
used = empty
For tile in group:
if tile is not OPEN, add it to used
For tile in group:
remove used from tile.possible
if tile.possible is a singleton, set that choice
My “naked single” code

def group_tactic_remove_taken(self, group):
    ...
    for tile in group:
        if tile.symbol != OPEN:
            used.add(tile.symbol)
    for tile in group:
        if tile.symbol == OPEN:
            tile.possible = tile.possible - used
        if len(tile.possible) == 1:
            tile.determine(tile.possible.copy().pop())
    ...

Strategy: Applying tactics cheapest first

First check that board is valid

(maintain validity as an invariant)

Then loop

Apply cheapest tactic

Apply more expensive tactic

...

(to the first tactic that gives progress)

Until no tactic succeeded in progress

From cheap to expensive

progress = True
while progress:
    progress = some_tactic()
    progress = progress or another_tactic()
    progress = progress or another_tactic()

# At end of loop, board may or may not be complete,
# but it’s still valid.

From cheap to expensive

progress = True
while progress:
    progress = some_tactic()
    progress = progress or another_tactic()
    progress = progress or another_tactic()

Short circuit “or”: if progress is True, don’t execute another_tactic()

# At end of loop, board may or may not be complete,
# but it’s still valid.

Executes only as many tactics as necessary to achieve some progress. Order from cheap to expensive (in efficiency).
Solver requirements

Required: naked single, hidden single
(aka sole candidate, unique candidate)
Print solution or partial solution

Optional: naked pair, hidden pair, etc
Your choice. Choose tactics that are fairly
straightforward to program.

Really really optional: Trial and error (dfs)
(requires reorganization of strategy)

Sudoku robot tournament

Entry qualification:
Robot must successfully complete each of a set of
puzzles that can be solved using only “naked
single” (“sole candidate”) and “hidden
single” (“unique candidate”) tactics

Tournament rules:
Submit robot and one puzzle, in standard format.
Winner is robot that fully solves the most student-
contributed puzzles within 5 minutes, without
crashing.

Prize?