Puzzles and Backtracking

Why Backtracking for Games/Puzzles?

- Any game/puzzle is fundamentally search problems over possible moves
- Backtracking systematically explores choices while efficiently discarding bad paths
  - Games naturally form a decision tree
  - Early pruning/discarding bad choices saves huge time
  - Easy to use!
    - Since majority of them follows the following structure

General Strategy: Template

```
def backtrack(state):
    if goal reached:
        save solution
        return

    for choice in possible choices:
        if valid(choice):
            make move
            backtrack(new state)
            undo move    # backtrack
```

Let's Explore one example to understand it

General Strategy/ Algo Template

```
def backtrack(state):
    if goal reached:
        save solution
        return

    for choice in possible choices:
        if valid(choice):
            make move
            backtrack(new state)
            undo move    # backtrack
```

Challenge reduced to “Encoding” these 4 things

Let’s Explore one example to understand it

Farmer, Goat, Wolf and Cabbage Puzzle

Puzzle Statement

There is a farmer who wishes to cross a river, but he is not alone. He also has a goat, a wolf, and a cabbage along with him. There is only one boat available, which can support the farmer and either the goat, wolf, or the cabbage. So at a time, the boat can have only two objects (farmer and one other).

If the goat and wolf are left alone (either in the boat or onshore), the wolf will eat the goat.

Similarly, if the Goat and cabbage are left alone, then goat will eat the cabbage.

Encoding this puzzle

F=Farmer; G=Goat; W=Wolf; C=Cabbage
0= in the Left bank; 1=in the Right Bank
(F,G, W, C)
(0, 0, 0, 0) → all on left
(1, 1, 1, 1) → all on right ⇒ Goal

Identify Constraints (aka Invalid States)

Encoding this puzzle

F=Farmer; G=Goat; W=Wolf; C=Cabbage
0= in the Left bank; 1=in the Right Bank
(F,G, W, C)
(0, 0, 0, 0) → all on left
(1, 1, 1, 1) → all on right

Identify Constraints (aka Invalid States)

❌ Goat + Wolf alone ⇒ `(G == W AND F != G)`
❌ Goat + Cabbage alone ⇒ `(G == C AND F != G)`
So, Valid state is:

```
NOT((G == W AND F != G)OR(G == C AND F != G))
```

Encoding this puzzle

F=Farmer; G=Goat; W=Wolf; C=Cabbage
0= in the Left bank; 1=in the Right Bank
(F,G, W, C)
(0, 0, 0, 0) → all on left
(1, 1, 1, 1) → all on right

Identify Constraints (aka Invalid States)

```
NOT((G == W AND F != G)OR(G == C AND F != G))
```

Possible Moves

1. Farmer alone
2. Farmer + Goat
3. Farmer + Wolf
4. Farmer + Cabbage

Farmer Always Moves

Encoding this puzzle

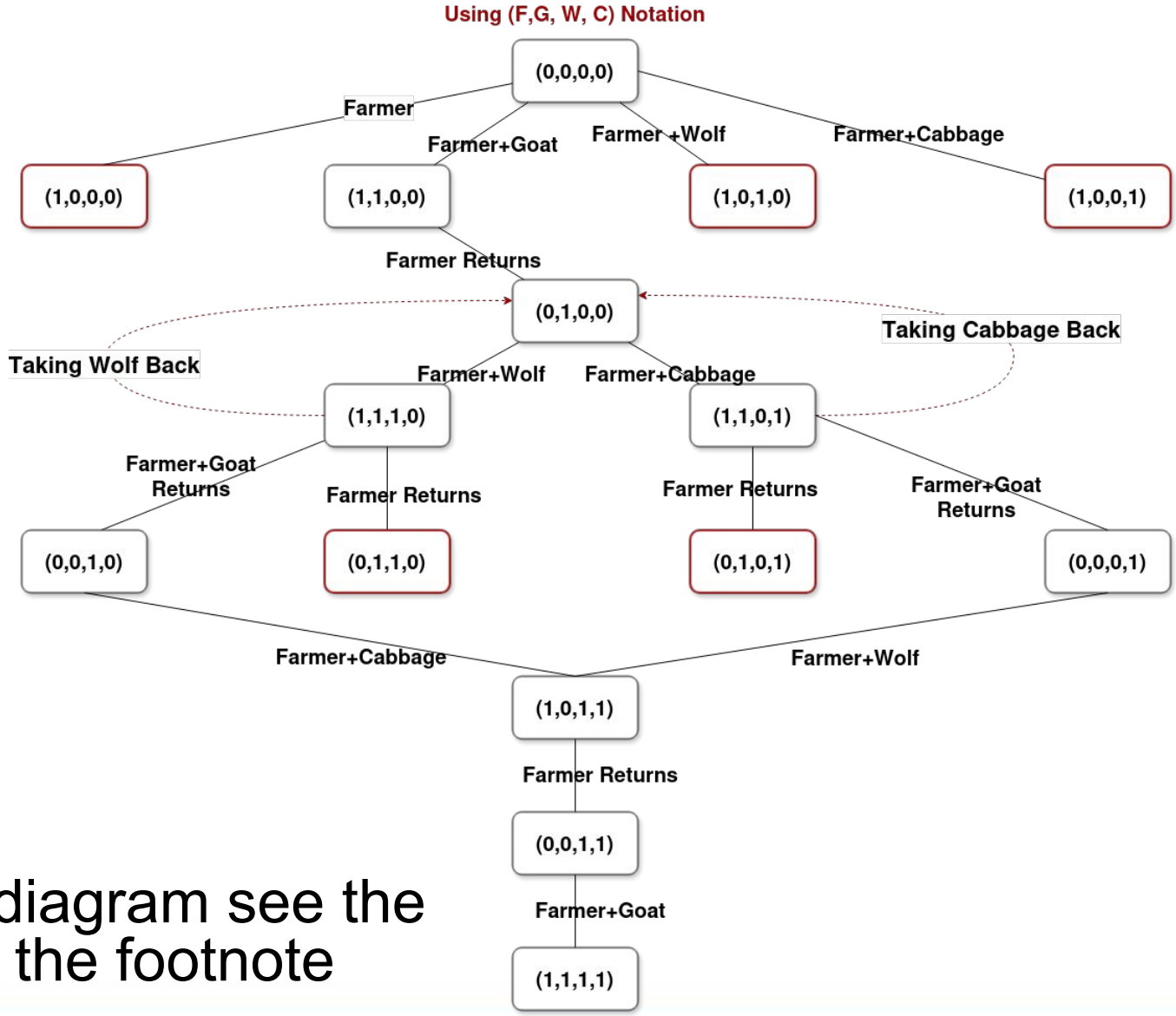
F=Farmer; G=Goat; W=Wolf; C=Cabbage
0= in the Left bank; 1=in the Right Bank

Possible Moves & State transition

Each move flips the location bit of the candidate/item
∴ use "Boolean Complement" or X□
if Farmer wants to take something with him/her
then item location and Farmer location should be same

1. Farmer alone
   - (F, G, W, C) → (F□, G, W, C)
2. Farmer + Goat (If F == G)
   - (F, G, W, C) → (F□, G□, W, C)
3. Farmer + Wolf (If F == W)
   - (F, G, W, C) → (F□, G, W□, C)
4. Farmer + Cabbage (If F == C)
   - (F, G, W, C) → (F□, G, W, C□)

State space / Solution Space



For clearer diagram see the link given in the footnote

(F,G, W, C); (0, 0, 0, 0) → all on left; (1, 1, 1, 1) → all on right ⇒ Goal

https://drive.google.com/file/d/1ed254123ztRjEOkFTu9XYazPUJ3u4TI8/

Math Crossword Solver

amazon

Problem Statement

- **You are given a 2D grid representing a math crossword puzzle.**
- **Each cell of the grid contains one of the following:**
  - **Symbol | Meaning**

Symbol	Meaning
.	Empty cell (Not to use)
B	Blank cell (to be filled with a no.)
No.	Fixed number (e.g., 23, 37)
+ - x /	Arithmetic operators
=	Equation separator

- <Token pool> : e.g <1 1 1 3 4 5 6 6 13 25 37>
  - The Blank cells are supposed to be filled using tokens.

Matrix like representation for Import/export

Blank cells; Use Drag-n-Drop to fill cells from Tokens

Game Board

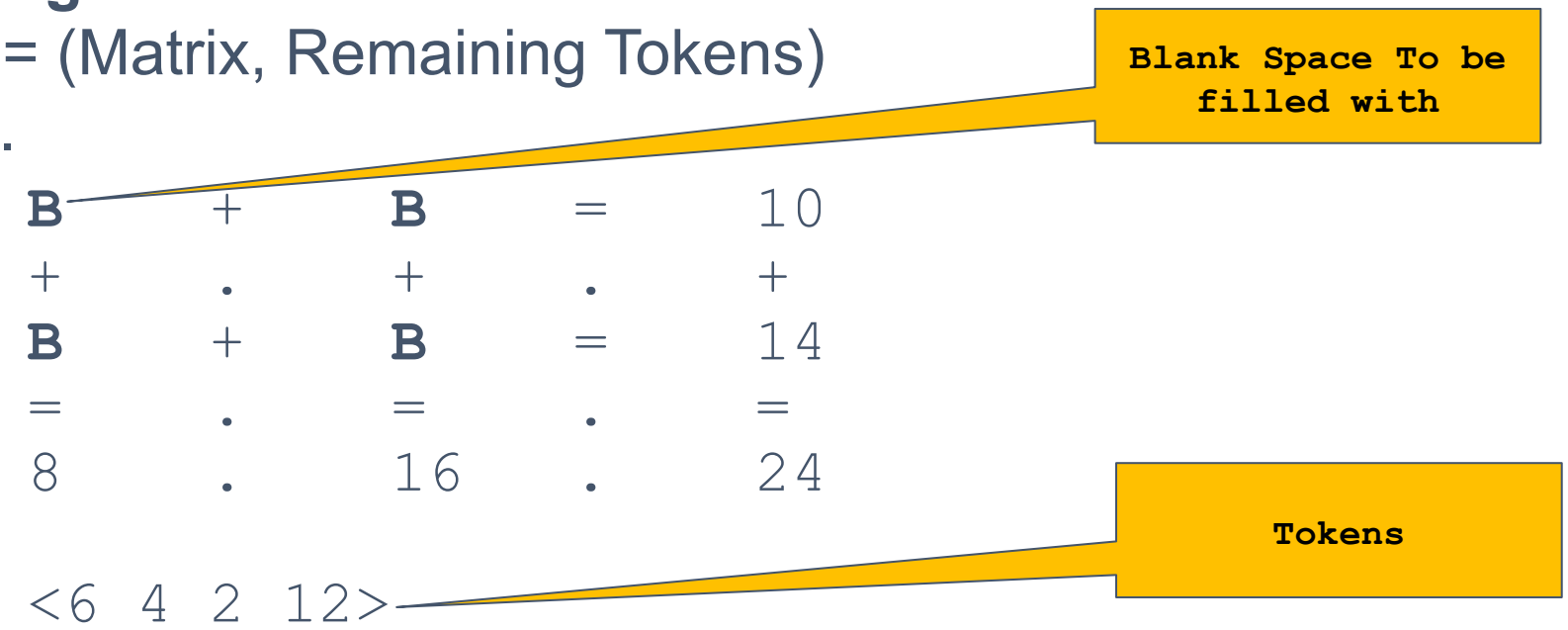
https://suchat-subho.github.io/webtools/MathCrossWordSolver/
https://play.google.com/store/apps/details?id=com.zm.crossmath&hl=en_IN

Problem Statement

- You are given a 2D grid representing a math crossword puzzle.
- Each cell of the grid contains one of the following:
- Objectives:
  - Fill all the Blank cells
    - i.e. Each B must be replaced by exactly one number from the token pool
  - Every horizontal and vertical sequence forms
    - <operand> <operator> <operand> = <result>
      - and must evaluate correctly
  - A number placed in a cell participates in:
    - one horizontal equation
    - one vertical equation
      - and both must be valid

Identifying the Puzzle Encoding

- **State and goal:**
  - State = (Matrix, Remaining Tokens)
    - i.e.



    - **Goal State:** No Blank space, Token list =∅ and valid
- Validity:
- Move:

Identifying the Puzzle Encoding

- **State and goal:**
  - State = (Matrix, Remaining Tokens)
- **Validity as per Game rules:**
  - Filled cells contains items from the token list
  - All equations are of the form

    `<operand> <operator> <operand> = <result>`
  - For all rows, and cols, each completed equation must evaluate correctly
- Move:

Identifying the Puzzle Encoding

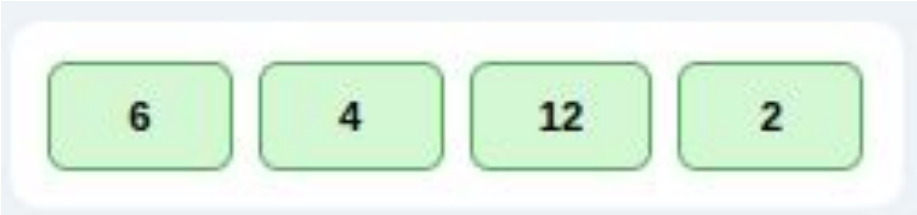
- **State and goal:**
  - State = (Matrix, Remaining Tokens)
- **Validity:**
  - Filled cells contains items from the token list
  - All equations are of the form

    `<operand> <operator> <operand> = <result>`
  - For all rows, and cols, each completed equation must evaluate correctly
- **Move:**
  - Identify all blank cells like (0,2) , and fill it using values from token pool/list
    - e.g. (0,2) = 12 , etc.

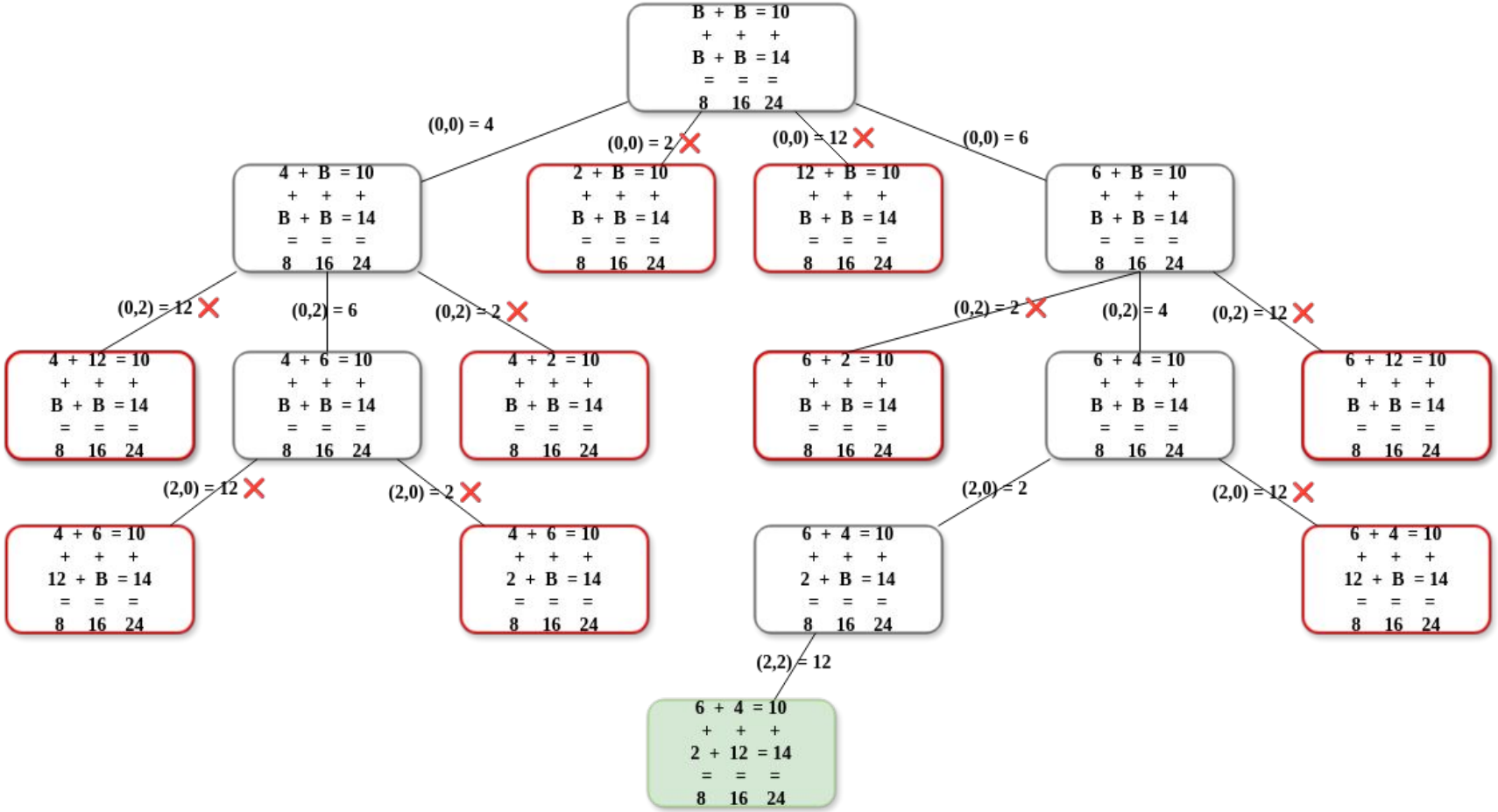
Game: Instance (G2)

```
B     +     B     =     10
+     .     +     .     +
B     +     B     =     14
=     .     =     .     =
8     .     16    .     24

<6 4 2 12>
```



Game: Instance (G2): State Space



For clearer diagram see the link given in the footnote

Solution Approach

- Procedure SOLVE(grid, blanks, tokens)
  - if all blanks are filled:
    - if VALIDATE(grid):
      - print grid // solution found
      - return TRUE
    - else:
      - return FALSE
  - (r, c) ← next blank cell
  - for each value in tokens:
    - if value not used:
      - if IS_SAFE(grid, r, c, value):
        - place value in grid[r][c] & mark value as used
        - if SOLVE(grid, blanks, tokens) = TRUE:
          - return TRUE
        - remove value from grid[r][c] & unmark value // backtrack
  - return FALSE

Solution Approach

- Function VALIDATE(grid)
  - for each row:
    - if equation in row is not correct:
      - return FALSE
  - for each column:
    - if equation in column is not correct:
      - return FALSE
  - return TRUE

- Function IS_SAFE(grid, r, c, value)
  - temporarily place value at grid[r][c]
  - if partial row is invalid:
    - undo placement
      - return FALSE
  - if partial column is invalid:
    - undo placement
      - return FALSE
  - undo placement
  - return TRUE

Solution Approach

- Function EVALUATE(expression)
  - replace 'x' with '*'
  - compute left-side
  - return TRUE if left-side == right-side else FALSE

Thank You