Laporan Final Project
Logika Formal

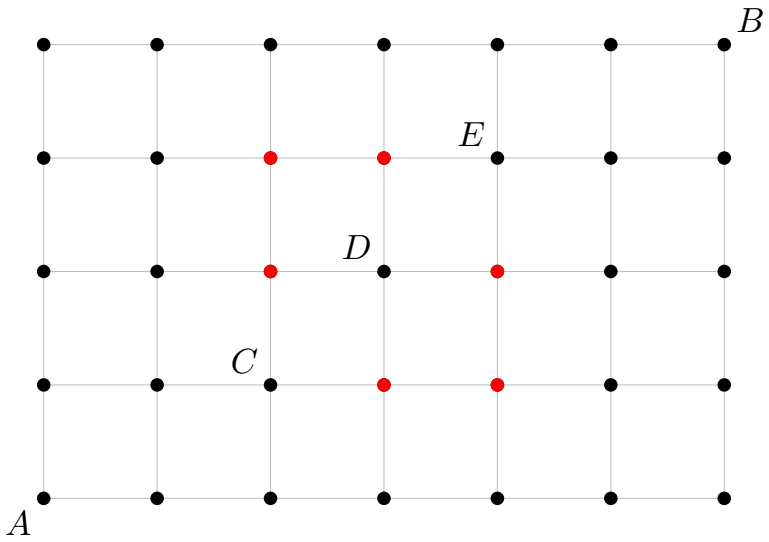
Disusun oleh:

Teosofi Hidayah Agung (5002221132)

Dosen Pengampu :
Muhammad Syifa'ul Mufid, S.Si., M.Si., D.Phil.
(19890911 201404 1 001)

Departemen Matematika
Institut Teknologi Sepuluh Nopember
Semester Genap 2024/2025
Surabaya

Problem Description



A robot is programmed to move across a grid board measuring 6 columns by 4 rows. The robot begins its journey from point $A(0, 0)$ and is assigned to reach the destination point $B(6, 4)$. However, the board is not entirely traversable. Certain points are marked in red, indicating that the robot is not allowed to pass through them. Additionally, there are points labeled C, D, and E which are located at strategic positions on the grid and play a critical role in determining the optimal path.

This problem involves designing and analyzing the robot's movements under two different motion modules, each with distinct movement rules:

- **Module (a)**
  The robot is only allowed to move one step either to the right or upward at a time. This means that from a position (x, y), the robot can move only to $(x + 1, y)$ or $(x, y + 1)$. Under this rule:
  - The task is to calculate the number of *possible paths* from point A to point B without passing through forbidden points.
  - Additionally, paths that go through point C must be excluded, so only paths that avoid C are counted.
- **Module (b)**
  In this second module, the robot's movements become more flexible. Besides moving right and up, the robot is also allowed to move diagonally upward-right, from (x, y) to $(x + 1, y + 1)$. With this additional rule:
  - The robot has three possible directions at each step.

– The problem not only involves counting the total number of valid paths but also identifying the *minimum-length paths* — those that require the fewest steps from A to B.
– Moreover, it must be proven that *every minimum-length path in this module passes through points C, D, and E*, signifying that these points are crucial in any optimal path.

Solution Flow

Solver Initialization and Variable Declaration

The solution begins by importing the `z3` library and defining the main parameters:

- `max_steps = 10`: This is set as the maximum number of steps considered for each path. This is based on the fact that the robot needs 6 steps on the X-axis and 4 steps on the Y-axis to reach from A to B.
- **Coordinate Variables**: The robot's path is represented by a series of coordinates (x_i, y_i), where i is the step index from 0 to `max_steps`. Hence, integer lists `X` and `Y` are declared with length `max_steps + 1`.
- **Solver Instance**: A `Solver()` object from Z3 is initialized to store all constraints.

Basic Constraint Definition

Several fundamental constraints are added to define the problem space:

- **Start and End Coordinates**: The robot must start at $A(0, 0)$ and end at $B(6, 4)$ on the final step (`max_steps`).
  - $X[0] = 0, Y[0] = 0$
  - $X[10] = 6, Y[10] = 4$
- **Grid Boundaries**: The robot must remain within the grid. The x-coordinate must be between 0 and 6, and y-coordinate between 0 and 4 for each step.
  - $0 \leq X[i] \leq 6$, for $i \in \{1, \ldots, 10\}$
  - $0 \leq Y[i] \leq 4$, for $i \in \{1, \ldots, 10\}$
- **Obstacles**: The robot is not allowed to step on any of the forbidden coordinates:

  $$(3, 1), (4, 1), (4, 2), (2, 2), (2, 3), (3, 3).$$

  This constraint is applied for every step from $i = 1$ to 9, excluding start and end.

Module (a): Horizontal and Vertical Movement Only

In this module, the robot can only move one step to the right or upward.

- `right_step(i)`: Represents a rightward step $(x_{i+1} = x_i + 1, y_{i+1} = y_i)$.
- `up_step(i)`: Represents an upward step $(x_{i+1} = x_i, y_{i+1} = y_i + 1)$.
- `module_a(solver)`: Constructs a new solver that inherits the base constraints and adds the movement rules for Module (a) for each step $i \in \{0, \ldots, 9\}$. Each step must either be a right step or an up step.

- `print_all_paths(solver, c)`: A utility function to find and print all valid paths according to the solver's constraints. It iteratively finds a satisfying model and then adds a constraint to exclude that path in subsequent iterations.

1. (i) Total Number of Valid Paths from A to B

Using `module_a` and calling `print_all_paths`, the system enumerates all valid paths in Module (a).

1. (ii) Showing No Valid Path Passes Through Point C

To demonstrate this, a constraint is added to force the robot to pass through point $C(2, 1)$ at any step.

- $\text{Or}(X[i] = 2 \wedge Y[i] = 1)$ for some $i \in \{1, \ldots, 10\}$
- After adding this constraint, the solver returns `unsat`, confirming that no valid path passes through point C.

Module (b): Horizontal, Vertical, or Diagonal Movement

This module introduces an additional movement option: diagonal up-right.

- `upward_diagonal_step(i)`: Represents a diagonal move ($x_{i+1} = x_i + 1, y_{i+1} = y_i + 1$).
- **Step Length Analysis**: The minimum number of steps from A to B is 6 (e.g., 4 diagonal and 2 horizontal moves). Paths of length from 1 to 10 are tested for validity.

2. (i) Total Number of Paths and (ii) Number of Minimum-Length Paths

The function `module_b` iterates over all path lengths from 1 to 10:

- For each path length i, a new solver is initialized and constraints applied for start, end, and step behavior.
- Any unused steps beyond i are "turned off" by setting $X[k] = 0, Y[k] = 0$ for $k > i$.
- If model is unsatisfiable, it indicates no valid paths of that length.
- Else if satisfiable, the model is printed and counted.

2. (iii) Proof: All Minimum Paths Pass Through C, D, and E

This part proves that all minimum-length paths (6 steps) pass through points $C(2,1)$, $D(3,2)$, and $E(4,3)$.

- **Approach**: Add a constraint to forbid passing through any of those three points. If the solver returns `unsat`, it means all paths must go through at least one of them.
- The constraint added is:

$$\neg\left((X[i]=2 \wedge Y[i]=1) \vee (X[i]=3 \wedge Y[i]=2) \vee (X[i]=4 \wedge Y[i]=3)\right)$$

  for $i \in \{1,\ldots,9\}$.
- **Result**: The solver returns `unsat` for the proofing by contradiction.

Conclusion

Based on the output generated by the Z3 solver program's execution, all questions in the robot navigation problem were answered completely. The use of SMT proved effective not only for finding valid paths but also for enumerating and proving specific properties of these paths.

Module (a): Horizontal and Vertical Movement

The program successfully identified the complete set of possible paths under the limited movement rules.

- **Path Count**: A total of **10 valid paths** were found from point A(0,0) to B(6,4).
- **Logical Proof**: It was proven that **no valid path** passes through the coordinate C(2,1), as the program's output was `unsat` for that condition.

Output of the 10 Valid Paths for Module (a)

The following is the complete list of the 10 paths found by the solver:

```
Path 1: (0,0)->(1,0)->(2,0)->(3,0)->(4,0)->(5,0)->(6,0)->(6,1)->(6,2)->(6,3)->(6,4)
Path 2: (0,0)->(0,1)->(0,2)->(1,2)->(1,3)->(1,4)->(2,4)->(3,4)->(4,4)->(5,4)->(6,4)
Path 3: (0,0)->(0,1)->(0,2)->(0,3)->(1,3)->(1,4)->(2,4)->(3,4)->(4,4)->(5,4)->(6,4)
Path 4: (0,0)->(0,1)->(0,2)->(0,3)->(0,4)->(1,4)->(2,4)->(3,4)->(4,4)->(5,4)->(6,4)
Path 5: (0,0)->(1,0)->(1,1)->(1,2)->(1,3)->(1,4)->(2,4)->(3,4)->(4,4)->(5,4)->(6,4)
Path 6: (0,0)->(0,1)->(1,1)->(1,2)->(1,3)->(1,4)->(2,4)->(3,4)->(4,4)->(5,4)->(6,4)
Path 7: (0,0)->(1,0)->(2,0)->(3,0)->(4,0)->(5,0)->(5,1)->(6,1)->(6,2)->(6,3)->(6,4)
Path 8: (0,0)->(1,0)->(2,0)->(3,0)->(4,0)->(5,0)->(5,1)->(5,2)->(6,2)->(6,3)->(6,4)
Path 9: (0,0)->(1,0)->(2,0)->(3,0)->(4,0)->(5,0)->(5,1)->(5,2)->(5,3)->(5,4)->(6,4)
Path 10: (0,0)->(1,0)->(2,0)->(3,0)->(4,0)->(5,0)->(5,1)->(5,2)->(5,3)->(6,3)->(6,4)
```

Module (b): Horizontal, Vertical, and Diagonal Movement

With the addition of diagonal movement, the program analyzed paths of varying lengths and provided comprehensive results.

- **Total Path Count**: A total of **57 valid paths** were found.
- **Path Length Distribution**: The output shows the path distribution as follows:
  - Length 1-5: 0 paths (no valid paths)
  - Length 6: 4 paths
  - Length 7: 12 paths
  - Length 8: 15 paths
  - Length 9: 16 paths

  - Length 10: 10 paths

- **Minimal Path**: The shortest possible path length is **6 steps**, with a total of 4 such paths.
- **Minimal Path Property Proof**: It was proven that all four minimal paths **must pass** through at least one of the points C(2,1), D(3,2), or E(4,3).

Output of the 4 Minimal Paths (Length 6) for Module (b)

The following is the list of the 4 shortest paths found by the solver (after cleaning up the redundant output):

```
Path 1: (0,0)->(1,0)->(2,1)->(3,2)->(4,3)->(5,4)->(6,4)
Path 2: (0,0)->(1,1)->(2,1)->(3,2)->(4,3)->(5,4)->(6,4)
Path 3: (0,0)->(1,1)->(2,1)->(3,2)->(4,3)->(5,3)->(6,4)
Path 4: (0,0)->(1,0)->(2,1)->(3,2)->(4,3)->(5,3)->(6,4)
```

Source Code

```
from z3 import *

max_steps = 10

X = [Int(f'x_{i}') for i in range(max_steps+1)]
Y = [Int(f'y_{i}') for i in range(max_steps+1)]

print(", ".join([f"({x}, {y})" for x, y in zip(X, Y)]))

s = Solver()

s.add(X[0] == 0, Y[0] == 0)
s.add(X[max_steps] == 6, Y[max_steps] == 4)

s.add([X[i+1] >= 0 for i in range(max_steps)])
s.add([X[i+1] <= 6 for i in range(max_steps)])
s.add([Y[i+1] >= 0 for i in range(max_steps)])
s.add([Y[i+1] <= 4 for i in range(max_steps)])

obs = [(3, 1), (4, 1), (4, 2), (2, 2), (2, 3), (3, 3)]
s.add([Or(X[i+1] != m, Y[i+1] != n) for i in range(max_steps-1) for (m, n) in
↪  obs])

def right_step(i):
    return And(X[i+1] == X[i] + 1, Y[i+1] == Y[i])

def up_step(i):
    return And(X[i+1] == X[i], Y[i+1] == Y[i] + 1)

def module_a(solver):
    new_solver = Solver()
    new_solver.add(solver.assertions())
    for i in range(max_steps):
        new_solver.add(Or(
            right_step(i),
            up_step(i)
        ))
    return new_solver

def print_all_paths(solver,c):
    count = c
    s_local = Solver()
    s_local.append(solver.assertions())
    while s_local.check() == sat:
        m = s_local.model()
```

```
        path = [(m.evaluate(X[i]).as_long(), m.evaluate(Y[i]).as_long()) for i
        ↪  in range(max_steps+1)]
        print(f"path {count+1} :", path)

        # Block current model
        s_local.add(Or([
            Or(X[i] != m.evaluate(X[i]), Y[i] != m.evaluate(Y[i]))
            for i in range(max_steps+1)
        ]))
        count += 1
    return count

solver_module_a = module_a(s)
count_module_a = print_all_paths(solver_module_a,c=0)
print(f"Total number of paths: {count_module_a}")

C = (2, 1)

solver_module_a.add(Or([And(X[i] == C[0], Y[i] == C[1]) for i in
↪  range(max_steps + 1)]))

if solver_module_a.check() == unsat:
  print(f"No valid path includes coordinate C{C}.")
else:
  print(f"There is a valid path that includes coordinate C{C}.")

def upward_diagonal_step(i):
    return And(X[i+1] == X[i] + 1, Y[i+1] == Y[i] + 1)

min_steps = 1

def module_b(solver):
    counter = 0
    for i in range(min_steps, max_steps+1):
        new_solver = Solver()
        new_solver.add(solver.assertions())
        new_solver.add(X[i]== 6, Y[i] == 4)
        new_solver.add([And(X[k] == 0, Y[k] == 0) for k in range(i+1,
        ↪  max_steps)])
        for j in range(i):
            new_solver.add(Or(
                right_step(j),
                up_step(j),
                upward_diagonal_step(j)
            ))
        if new_solver.check() == sat:
            print(f"Module B with {i} steps:")
            counter = print_all_paths(new_solver,counter)
        else:
```

```
            print(f"Module B with {i} steps: No valid paths found.")
    print(f"Total number of paths: {counter}")

module_b(s)

def module_b_ii(solver):
    new_solver = Solver()
    new_solver.add(solver.assertions())
    new_solver.add([Not(Or(And(X[i] == 2,Y[i] == 1),And(X[i] == 3,Y[i] ==
    ↪  2),And(X[i] == 4,Y[i] == 3))) for i in range(1,max_steps)])
    new_solver.add(X[6]== 6, Y[6] == 4)
    new_solver.add([And(X[k] == 0, Y[k] == 0) for k in range(7, max_steps)])
    for j in range(6):
        new_solver.add(Or(
            right_step(j),
            up_step(j),
            upward_diagonal_step(j)
        ))
    if new_solver.check() == sat:
        print("There exists a minimal path of 6 steps that does NOT pass
        ↪  through C(2,1), D(3,2), or E(4,3):")
        print_all_paths(new_solver, 0)
    else:
        print("Not exist paths of 6 steps without pass through C(2,1), D(3,2),
        ↪  or E(4,3).")

module_b_ii(s)
```