Logic and Classical Planning

Logical Pacman,
Food is good AND ghosts are bad,
Spock would be so proud

Table of Contents

Introduction

In this project, you will use/write simple Python functions that generate logical sentences describing Pacman physics, aka pacphysics. Then you will use a SAT solver, `pycosat`, to solve the logical inference tasks associated with planning, localization, mapping, and SLAM.

Important Files:

- **logicPlan.py** - Where you will put your code for the various logical agents.
- **logic.py** - Propositional logic code originally from aima-python.
- **logicAgents.py** - The file that defines planning problems.
- **pycosat_test.py** - Quick test for pycosat module installation.

The Expr Class

The `Expr` class represents propositional logic sentences. An `Expr` object is implemented as a tree with logical operators at each node and literals at the leaves.

Example Expression

The expression $(A \wedge B) \leftrightarrow (\neg C \vee D)$ is represented as a tree with ↔ at the root.

Python Operators

Python Operator	Logic Symbol	Meaning
~A	¬A	NOT A
A & B	A ∧ B	A AND B
A | B	A ∨ B	A OR B
A >> B	A → B	A IMPLIES B
A % B	A ↔ B	A IFF B

Important Note on conjoin and disjoin

Use `conjoin` and `disjoin` operators wherever possible. `conjoin` creates a chained & expression, and `disjoin` creates a chained | expression.

Instead of: `condition = A & B & C & D & E`

Use: `condition = conjoin([A, B, C, D, E])`

Prop Symbol Names

Rules

- Variables must start with an upper-case character
- Only these characters: A-Z, a-z, 0-9, _, ^, [,]
- No logical connective characters (&, |) in variable names

Pacphysics Symbols

- `PropSymbolExpr(pacman_str, x, y, time=t)` : Whether Pacman is at (x,y) at time t
- `PropSymbolExpr(wall_str, x, y)` : Whether a wall is at (x,y)
- `PropSymbolExpr(action, time=t)` : Whether Pacman takes action at time t

SAT Solver Setup

Install pycosat using one of the following:

```
pip install pycosat
pip3 install pycosat
conda install pycosat
```

Test installation:

```
python pycosat_test.py
```

Expected output: `[1, -2, -3, -4, 5]`

Q1 (2 points): Logic Warm-up

Implement the following functions in `logicPlan.py`:

sentence1()

Create one Expr instance representing these three sentences:

1. $A \vee B$
2. $\neg A \leftrightarrow (\neg B \vee C)$
3. $\neg A \vee \neg B \vee C$

sentence2()

Create one Expr instance representing these four sentences:

1. $C \leftrightarrow (B \vee D)$
2. $A \rightarrow (\neg B \wedge \neg D)$
3. $\neg(B \wedge \neg C) \rightarrow A$
4. $\neg D \rightarrow C$

sentence3()

Create symbols and encode three English sentences:

1. Pacman is alive at time 1 iff he was alive at time 0 and not killed, or not alive at time 0 and born
2. At time 0, Pacman cannot both be alive and be born
3. Pacman is born at time 0

Additional Functions

- `findModelUnderstandingCheck()`
- `entails(premise, conclusion)`
- `plTrueInverse(assignments, inverse_statement)`

Q2 (2 points): Logic Workout

Implement these functions in CNF:

atLeastOne(literals)

Return an expression that is true if at least one literal is true.

atMostOne(literals)

Return an expression that is true if at most one literal is true. Hint: Use itertools.combinations.

exactlyOne(literals)

Use atLeastOne and atMostOne to return an expression true if exactly one literal is true.

> All expressions must be in CNF without using to_cnf().

Q3 (4 points): Pacphysics and Satisfiability

Implement basic pacphysics logical expressions:

pacmanSuccessorAxiomSingle

Generate expressions for Pacman to be at (x,y) at time t.

pacphysicsAxioms

Generate physics axioms for timestep t:

- Wall implications
- Pacman at exactly one location
- Pacman takes exactly one action
- Sensor axioms
- Successor axioms

checkLocationSatisfiability

Return two models:

- Model where Pacman IS at (x_1,y_1)
- Model where Pacman is NOT at (x_1,y_1)

Q4 (3 points): Path Planning with Logic

Implement `positionLogicPlan(problem)`.

Algorithm

1. Add initial Pacman location to KB
2. For each timestep t:
    - Add Pacman at exactly one location
    - Check if goal is satisfied
    - Add action constraints
    - Add transition model
3. Return action sequence if goal found

Q5 (3 points): Eating All the Food

Implement `foodLogicPlan(problem)`.

Changes from Q4

- Initialize Food[x,y]_t variables
- Change goal: all food eaten (all Food[x,y]_t false)
- Add food successor axiom:

$$Food[x,y]\_\{t+1\} \leftrightarrow (Food[x,y]\_t \wedge \neg Pacman[x,y]\_t)$$

Helper Functions

These helper functions are used in the remaining questions:

Add pacphysics to KB

- Add pacphysics_axioms
- Add Pacman's action
- Add percept rules

Find possible Pacman locations

- Check satisfiability for each coordinate
- Add provable locations to KB

Find provable wall locations

- Check satisfiability for walls at each coordinate
- Update known_map

Q6 (4 points): Localization

Implement `localization(problem, agent)`.

4-bit Sensor

Returns walls in North, South, East, West directions.

Algorithm

1. Add walls to KB
2. For each timestep:
    - Add pacphysics and percepts
    - Find possible locations
    - Move agent
    - Yield possible locations

Q7 (3 points): Mapping

Implement `mapping(problem, agent)`.

Known Map Format

- 1: Guaranteed wall
- 0: Guaranteed not wall
- -1: Ambiguous

Algorithm

1. Add initial Pacman location to KB
2. For each timestep:
    - Add pacphysics and percepts
    - Find provable walls
    - Move agent
    - Yield known_map

Q8 (4 points): SLAM

Implement `slam(problem, agent)`.

SLAM Challenges

- Unknown initial walls
- Illegal actions possible
- 3-bit sensor: adjacent wall count

3-bit Sensor

- 000: No adjacent walls
- 100: 1 adjacent wall
- 110: 2 adjacent walls
- 111: 3 adjacent walls

Algorithm

1. Add initial location to KB
2. For each timestep:
    - Add pacphysics using SLAM axioms
    - Find provable walls
    - Find possible locations
    - Move agent
    - Yield (known_map, possible_locations)

Submission

To submit your assignment:

1. Complete all functions in `logicPlan.py`
2. Test your code: `python autograder.py`
3. Submit all .py files to Gradescope

Important Rules

- Do not change provided function names
- Academic integrity is strictly enforced
- Contact course staff for help if needed

Additional Resources

- AIMA Chapter 7.7 for pacphysics reference
- Office hours and discussion forums for support