LISV Laboratoire d'ingénierie des systèmes de Versailles
UVSQ université PARIS-SACLAY

AI Planning Lab 3
Hierarchical Task Network (HTN) with PyHOP

Halim Djerroud

revision 1.0

Introduction: Hierarchical Task Network Planning

Hierarchical Task Network (HTN) planning decomposes complex tasks into simpler subtasks recursively until reaching primitive actions.

Key concepts:

- **Operators**: Primitive actions that modify the world state
- **Methods**: Recipes that decompose abstract tasks into subtasks
- **State**: Current configuration of the world

In this lab, you'll use **PyHOP**, a simple HTN planner in Python.

Basic PyHOP Structure

```python
from pyhop import hop

# 1. Define the initial state
state = hop.State('state0')
state.pos = {'robot': 'rooma'}
state.holding = None

# 2. Define operators
def move(state, from_room, to_room):
    if state.pos['robot'] == from_room:
        state.pos['robot'] = to_room
        return state
    return False

hop.declare_operators(move)

# 3. Define methods
def travel(state, destination):
    if state.pos['robot'] == destination:
        return []  # Already there
    return [('move', state.pos['robot'], destination)]

hop.declare_methods('travel', travel)

# 4. Run the planner
hop.plan(state, [('travel', 'roomb')],hop.get_operators(),hop.get_methods(),verbose=1))
```

Exercise 1: LEGO Kit Assembly Robot

Scenario

A robot assembles customized LEGO kits. A customer orders:

- **House**: red roof + black walls
- **Tree**: brown trunk + branches (no leaves)
- **Ground**: blue
- **Car**: black chassis + red body

The robot must prepare bags hierarchically:

```
main_bag
|-- house_bag
|   |-- roof_bag (red bricks)
|   |-- walls_bag (black bricks)
|-- tree_bag
|   |-- trunk_bag (brown bricks)
|   |-- branches_bag (brown bricks)
|-- ground_bag (blue bricks)
|-- car_bag
    |-- chassis_bag (black bricks)
    |-- body_bag (red bricks)
```

Task 1.1: Define the State (10 min)

Create the initial state with:

- `state.robot_pos`: robot's current zone
- `state.stock`: dictionary of available bricks by type and color
- `state.bags`: dictionary tracking what's in each bag
- `state.ready`: set of completed bags

Hint: Stock example:

```python
state.stock = {
    'roof_brick': {'red': 20, 'blue': 10},
    'wall_brick': {'black': 30, 'red': 15},
    # ... add other brick types
}
```

Task 1.2: Define Primitive Operators (15 min)

Implement these operators:

Operator: `move_to` Move robot to a warehouse zone.

```python
def move_to(state, zone):
    # TODO: Check robot can move to zone
    # TODO: Update state.robot_pos
    # Return modified state or False
    pass
```

Operator: `take_bricks` Pick bricks from stock and put in a bag.

```python
def take_bricks(state, brick_type, color, quantity, bag_name):
    # TODO: Check stock has enough bricks
    # TODO: Decrease stock
    # TODO: Add to state.bags[bag_name]
    # Return modified state or False
    pass
```

Operator: `pack_bag` Put a sub-bag into a larger bag.

```
def pack_bag(state, sub_bag, main_bag):
    # TODO: Check sub_bag is ready
    # TODO: Add sub_bag to main_bag
    # Return modified state or False
    pass
```

Operator: `mark_ready` Mark a bag as complete.

```
def mark_ready(state, bag_name):
    # TODO: Add bag_name to state.ready
    return state
```

Don't forget: `hop.declare_operators(move_to, take_bricks, pack_bag, mark_ready)`

Task 1.3: Define HTN Methods (20 min)

Method: `prepare_roof` Prepare roof bag with red bricks.

```
def prepare_roof(state, color):
    # Return a list of tasks:
    # [('move_to', 'roof_zone'),
    #  ('take_bricks', 'roof_brick', color, 10, 'roof_bag'),
    #  ('mark_ready', 'roof_bag')]
    pass

hop.declare_methods('prepare_roof', prepare_roof)
```

Method: `prepare_house` Prepare complete house bag.

```
def prepare_house(state, roof_color, wall_color):
    # Return tasks to:
    # 1. Prepare roof
    # 2. Prepare walls
    # 3. Pack them into house_bag
    # Example: [('prepare_roof', roof_color),
    #           ('prepare_walls', wall_color),
    #           ('pack_bag', 'roof_bag', 'house_bag'),
    #           ...]
    pass

hop.declare_methods('prepare_house', prepare_house)
```

TODO: Define similar methods for:

- `prepare_walls`
- `prepare_trunk`
- `prepare_branches`
- `prepare_tree`
- `prepare_ground`
- `prepare_chassis`
- `prepare_body`
- `prepare_car`

Task 1.4: Main Method (10 min)

Define the top-level method:

```python
def prepare_order(state, order):
    # order = {'house': ('red', 'black'),
    #          'tree': ('brown', False),  # False = no leaves
    #          'ground': 'blue',
    #          'car': ('black', 'red')}

    # Return tasks to prepare all components
    # and pack them in main_bag
    pass

hop.declare_methods('prepare_order', prepare_order)
```

Task 1.5: Run the Planner (5 min)

```python
order = {
    'house': ('red', 'black'),
    'tree': ('brown', False),
    'ground': 'blue',
    'car': ('black', 'red')
}

plan = hop.plan(state, [('prepare_order', order)], ,hop.get_operators(),hop.get_methods(),verbose=1))
print(f"\nGenerated plan with {len(plan)} actions")
```

Expected output: A plan with 30-40 primitive actions.

Exercise 2: Analysis Questions

Question 2.1: Plan Length

How many actions are in your generated plan? Why?

Question 2.2: Hierarchy Depth

Draw the task decomposition tree for `prepare_house`.

Question 2.3: Comparison with PDDL

What are the advantages of HTN over classical PDDL planning for this problem?

Question 2.4: Optimization

The robot moves between zones multiple times. How could you reduce movements?

Exercise 3: Extensions (Bonus)

Extension 3.1: New Component

Add a "garden" component with flowers. Modify the hierarchy accordingly.

Extension 3.2: Stock Management

What happens if some bricks are out of stock? Modify `take_bricks` to handle this.

Extension 3.3: Alternative Methods

Create an alternative method for `prepare_house` that prepares walls before roof. Does PyHOP find different plans?

Grading Rubric

Component	Points
Task 1.1: State definition	10
Task 1.2: Operators (4 functions)	20
Task 1.3: Methods (8+ functions)	40
Task 1.4: Main method	10
Task 1.5: Working planner	10
Question 2: Analysis	10
Total	**100**
Bonus: Extensions	+10

Submission

Submit a single Python file `lego_htn.py` containing:

- All operator definitions
- All method definitions
- State initialization
- Test code that runs the planner
- Comments explaining your design choices