课程作业2：逻辑与经典规划实验报告

1 摘要

本实验报告旨在详细记录和总结在“逻辑与经典规划”项目中所进行的工作和取得的成果。项目的核心目标是实现一个基于命题逻辑的吃豆人智能代理，该代理能够在复杂的环境中做出合理的行动决策。具体来说，本实验分为五个主要任务：逻辑语句表示、逻辑函数实现、构建吃豆人世界、使用逻辑进行路径规划，以及使用逻辑规划吃掉所有豆子。

在逻辑语句表示部分，我们主要关注构建和表示各种逻辑语句。实验要求实现了多个函数，用于构建和操作命题逻辑表达式。在逻辑函数实现部分，我们实现了与逻辑操作相关的函数，如判断至少有一个命题为真、至多有一个命题为真和确切地有一个命题为真。

接下来，我们构建了吃豆人世界的逻辑表达式。这包括定义吃豆人的状态转移、感知模型和物理规则。通过合适的逻辑表达式，我们可以推断出吃豆人在各个时间点的可能位置。

在路径规划部分，我们使用已构建的逻辑表达式来进行路径规划，使吃豆人能够基于逻辑推理找到达到目标位置的路径。最后，在规划吃掉所有豆子的任务中，我们进一步扩展了逻辑表达式和规划方法，以实现一个能够吃掉所有豆子的吃豆人智能代理。

本报告将详细描述每个任务的实现方法、过程和结果，以及在项目实施过程中遇到的挑战和解决方案。通过这个项目，我们深入了解了命题逻辑在智能代理和规划中的应用，以及如何将逻辑理论应用于实际问题的解决。

关键词：命题逻辑、逻辑表达式、吃豆人游戏、路径规划、状态转移、感知模型

2 1 引言

逻辑是计算机科学和人工智能（AI）的基石。在AI的许多领域，特别是在知识表示和推理（KRR）领域，逻辑被用作形式化和表达信息的工具。本实验旨在探究命题逻辑在经典规划问题中的应用，特别是在一个经典的吃豆人游戏场景中。吃豆人作为一个历史悠久的游戏，因其简单的规则和易于实现的环境而成为AI实验和研究的热门选择。

本实验主要侧重于使用命题逻辑构建和表示逻辑语句，以此来描述和推理吃豆人的世界。在实验中，我们首先将简单的命题语句转化为逻辑表达式。接着，我们通过命题逻辑构建更为复杂的函数，如逻辑语句的合取、析取等操作。然后，我们在吃豆人的世界中应用这些逻辑表达式，包括描述吃豆人的位置、行动和周围环境。

实验的目的是通过应用命题逻辑来解决经典规划问题，如路径规划和动作序列规划。我们将创建一个能够根据给定的环境和目标，合理规划行动路径的吃豆人智能代理。通过这种方式，本实验旨在提高学生对命题逻辑应用于实际问题的理解和能力。

3 2 实验内容

3.1 2.1 逻辑语句表示（Logic Warm-up）

在本部分的实验中，我们重点关注逻辑语句的表示。实验的目标是正确实现和应用命题逻辑表达式，理解和使用这些逻辑表达式来解决实际问题。

3.1.1 代码实现

在这个部分，我们实现了几个函数，每个函数都是为了解决特定的逻辑问题。下面是一些代码实现的详细描述：

1. **sentence1(), sentence2(), sentence3() 函数：** 这些函数的目的是返回表示特定逻辑语句的 `Expr` 实例。我们用Python的运算符如 `|`（或），`&`（与），`~`（非）来直观地表示逻辑表达式。

2. **findModel() 函数：** 此函数接收一个逻辑句子作为输入，并返回一个满足该逻辑句子的模型（如果存在的话）。它首先将输入句子转换为合取范式（CNF），然后使用 `pycoSAT` 函数找到一个满足该CNF的模型。

3. **findModelUnderstandingCheck() 函数：** 这个函数返回一个特定表达式的模型，用于理解检查。

4. **entails() 函数：** 此函数检查一个前提是否导致结论。它通过找到一个使前提为真而结论为假的模型来完成此操作。

5. **plTrueInverse() 函数：** 这个函数返回给定赋值下非逆语句是否为真。

3.1.2 实验结果

所有的测试案例都已经通过，如下所示：

```
Question q1
===========

*** PASS: test_cases\q1\correctSentence1.test
***     PASS
*** PASS: test_cases\q1\correctSentence2.test
***     PASS
*** PASS: test_cases\q1\correctSentence3.test
***     PASS
*** PASS: test_cases\q1\entails.test
***     PASS
*** PASS: test_cases\q1\entailsLong.test
***     PASS
*** PASS: test_cases\q1\findModelSentence1.test
***     PASS
*** PASS: test_cases\q1\findModelSentence2.test
***     PASS
*** PASS: test_cases\q1\findModelSentence3.test
***     PASS
*** PASS: test_cases\q1\findModelUnderstandingCheck.test
***     PASS
*** PASS: test_cases\q1\plTrueInverse.test
***     PASS

### Question q1: 2/2 ###
```

这意味着我们实现的逻辑函数能够正确地解释和操作逻辑表达式，并且函数的实现是正确的。

3.1.3 结论

逻辑语句表示部分的实验任务成功完成。代码实现正确地表示了逻辑语句，并且所有的函数都已通过了所提供的测试案例。这验证了我们实现的逻辑表达式操作的准确性和可靠性。

3.2 2.2 逻辑函数实现（Logic Workout）

在本部分中，我们集中实现了逻辑函数，这些函数可以处理和解释涉及一组逻辑文字（例如，A 或 ~A）的命题逻辑表达式。

3.2.1 代码实现

我们实现了三个主要函数：`atLeastOne()`，`atMostOne()` 和 `exactlyOne()`。以下是每个函数的详细描述：

1. **atLeastOne(literals: List[Expr]) -> Expr：** 这个函数接受一个逻辑文字的列表，并返回一个 `Expr` 实例，该实例以合取范式（CNF）表示至少有一个列表中的文字为真的逻辑。

   ```
   return disjoin(literals)
   ```

   通过这个简单的实现，函数可以通过一个或多个文字的析取来正确返回结果。

2. **atMostOne(literals: List[Expr]) -> Expr：** 这个函数也接受一个逻辑文字的列表，并返回一个表示列表中最多有一个文字为真的逻辑的 `Expr` 实例。

   ```
   res = []
   for i in itertools.combinations(literals, 2):
       res.append(~i[0] | ~i[1])
   return conjoin(res)
   ```

   通过迭代文字的所有可能的组合，并且对每一对都进行析取和否定，我们能得到符合条件的逻辑表达式。

3. **exactlyOne(literals: List[Expr]) -> Expr：** 此函数接受逻辑文字的列表，并返回一个 `Expr` 实例，该实例表示列表中恰好有一个文字为真的逻辑。

   ```
   return conjoin([atLeastOne(literals), atMostOne(literals)])
   ```

   函数通过结合 `atLeastOne` 和 `atMostOne` 函数的结果来得出结论。

3.2.2 实验结果

实验测试了所有的实现函数，并且所有的测试都已通过

```
Question q2
===========

*** PASS: test_cases\q2\atLeastOne.test
***     PASS
*** PASS: test_cases\q2\atLeastOneCNF.test
***     PASS
*** PASS: test_cases\q2\atLeastOneEff.test
***     PASS
*** PASS: test_cases\q2\atMostOne.test
***     PASS
*** PASS: test_cases\q2\atMostOneCNF.test
***     PASS
*** PASS: test_cases\q2\atMostOneEff.test
***     PASS
*** PASS: test_cases\q2\exactlyOne.test
***     PASS
*** PASS: test_cases\q2\exactlyOneCNF.test
***     PASS
*** PASS: test_cases\q2\exactlyOneEff.test
***     PASS
```

```
### Question q2: 2/2 ###
```

3.2.3 结论

逻辑函数实现部分的任务已经成功完成。我们实现的函数能够正确地处理和解释涉及逻辑文字集合的命题逻辑表达式。实验结果表明，我们的实现是正确和可靠的，可以用于进一步的逻辑推理和评估。

3.3 2.3 构建吃豆人世界（Pacphysics and Satisfiability）

在此部分，我们主要聚焦于Pacman的物理世界，即怕Pacman如何在网格世界中移动，以及我们如何确定其特定位置的可满足性。我们引入了几个函数来描述和处理Pacman在其世界中的动作和位置。

3.3.1 代码实现

1. **PacmanSuccessorAxiomSingle**

   这个函数定义了Pacman从时间 (t-1) 到时间 (t) 的转移公理。它考虑了Pacman可用的动作（北，东，南，西）和相应时间的可能位置。

2. **SLAMSuccessorAxiomSingle**

   这个函数类似于 `PacmanSuccessorAxiomSingle`，但它也考虑了Pacman可能未移动的情况，例如当Pacman尝试移动到墙壁时。

3. **pacphysicsAxioms**

   此函数返回描述Pacman物理世界的逻辑句子。这包括：

   - Pacman不能位于墙壁位置。
   - 在每个时间步，Pacman应该在一个位置。
   - Pacman在每个时间步做一个动作。
   - 如果提供了传感器模型和后继公理，则会加入这些内容。

4. **checkLocationSatisfiability**

   此函数检查在给定的动作和Pacman的初始位置下，某个位置在时间 (t = 1) 是否可满足。它返回两个模型：一个是Pacman在指定位置的模型，另一个是Pacman不在指定位置的模型。

3.3.2 实验结果

实验结果显示了Pacman位置的可满足性测试和物理公理的应用。

```
Question q3
===========

*** Testing checkLocationSatisfiability
[LogicAgent] using problem type LocMapProblem
Ending game
Average Score: 0.0
Scores:        0.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** PASS: test_cases\q3\location_satisfiability1.test
```

```
*** Testing checkLocationSatisfiability
[LogicAgent] using problem type LocMapProblem
Ending game
Average Score: 0.0
Scores:        0.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** PASS: test_cases\q3\location_satisfiability2.test
*** Testing pacphysicsAxioms
*** PASS: test_cases\q3\pacphysics1.test
*** Testing pacphysicsAxioms
*** PASS: test_cases\q3\pacphysics2.test
*** PASS: test_cases\q3\pacphysics_transition.test
***     PASS

### Question q3: 4/4 ###
```

- 通过使用`checkLocationSatisfiability`函数进行测试，我们可以观察到在特定动作和初始位置的条件下，某些位置在 (t = 1) 是可满足的，而某些位置则不是。
- `pacphysicsAxioms`函数也通过测试，确认了其有效性，能够正确地应用Pacman的物理世界的规则。

实验的输出结果显示，测试用例已经成功通过，证明了实现的代码可以准确地描述和处理Pacman在其世界中的移动。

3.3.3 结论

通过本部分的实验，我们成功地应用逻辑规则和知识库构建了吃豆人世界的物理规则（Pacphysics）。我们也尝试了不同的方法和策略来实现满足性问题。实验结果表明，我们的实现能够有效地处理和解决吃豆人游戏中的基本逻辑和物理规则问题。这验证了我们方法的可行性和准确性，也为后续实验，如路径规划和食物搜索，提供了坚实的基础。

3.4 2.4 使用逻辑进行路径规划（Path Planning with Logic）

在本部分的实验中，我们采用逻辑方法来解决路径规划问题。具体来说，我们通过逻辑断言和推理来找到一个合法的动作序列，从而使Pacman达到目标位置。我们使用的逻辑符号化语言来描述Pacman的动作和位置，以及迷宫的墙壁和目标的位置。

3.4.1 代码实现

我们定义了`positionLogicPlan`函数，其输入是一个位置规划问题实例，输出是一个导致目标的动作序列。

1. **初始知识：** 首先，我们从问题实例中提取有关迷宫的信息，包括墙壁的位置、Pacman的初始位置和目标位置。
2. **可能的动作和位置：** 确定了可能的动作（‘North’, ‘East’, ‘South’, ‘West’）和没有墙壁的可能位置。
3. **知识库构建：**
    - 确保在每个时间步，Pacman都只能在一个没有墙的位置。
    - 确保Pacman在每个时间步都只能采取一个动作。
    - 定义Pacman的后继状态，根据当前动作和位置确定Pacman在下一时间步的位置。
4. **目标查询：**

- 对每个时间步，都尝试找到一个模型（满足所有已知逻辑的一个实例），使Pacman达到目标。
- 一旦找到这样的模型，就从中提取动作序列并返回。

3.4.2 实验结果

我们对`positionLogicPlan`函数进行了测试，得到了以下结果：

```
Question q4
===========

[LogicAgent] using problem type PositionPlanningProblem
Path found with total cost of 2 in 0.0 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 508
Average Score: 508.0
Scores:        508.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases\q4\positionLogicPlan1.test
***     pacman layout:          maze2x2
***     solution score:         508
***     solution path:          West South
[LogicAgent] using problem type PositionPlanningProblem
Path found with total cost of 8 in 0.3 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases\q4\positionLogicPlan2.test
***     pacman layout:          tinyMaze
***     solution score:         502
***     solution path:          South South West South West West South West
[LogicAgent] using problem type PositionPlanningProblem
Path found with total cost of 19 in 65.1 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 491
Average Score: 491.0
Scores:        491.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases\q4\positionLogicPlan3.test
***     pacman layout:          smallMaze
***     solution score:         491
***     solution path:          East East South South West South South West West South West
West West West West West West West West

### Question q4: 3/3 ###
```

- 在一个2x2的迷宫中，Pacman在0.0秒内找到了总成本为2的路径。路径是：West, South。
- 在一个名为'tinyMaze'的迷宫中，Pacman在0.3秒内找到了总成本为8的路径。路径是：South, South, West, South, West, West, South, West。

- 在一个名为'smallMaze'的迷宫中，Pacman在65.1秒内找到了总成本为19的路径。路径是：East, East, South, South, West, South, South, West, West, South, West, West, West, West, West, West, West, West。

通过这个实验，我们可以看到逻辑规划方法在解决路径规划问题上的有效性。对于较小和简单的迷宫，该方法能够快速找到解决方案。但是，对于更复杂的迷宫，计算时间明显增加，这可能是由于在更多的时间步中需要处理更多的逻辑断言和查询。

3.4.3 结论

通过这个实验，我们可以看到逻辑规划方法在解决路径规划问题上的有效性。对于较小和简单的迷宫，该方法能够快速找到解决方案。但是，对于更复杂的迷宫，计算时间明显增加，这可能是由于在更多的时间步中需要处理更多的逻辑断言和查询。

3.5 2.5 吃掉所有豆子（Eating All the Food）

在本部分，我们通过逻辑推理解决了食物规划问题，目标是找到一个有效的动作序列使Pacman能吃掉迷宫中的所有食物。

3.5.1 代码实现

函数 `foodLogicPlan` 被用来实现这个目标。这个函数接受一个食物规划问题的实例，并返回一个动作列表。

1. **初始知识：** 从问题实例中提取墙壁位置，Pacman的初始位置，和食物位置等信息。
2. **可能的动作和位置：** 定义了Pacman可能的动作（'North', 'East', 'South', 'West'）和不包含墙壁的可达位置。
3. **知识库构建：**
   - 确保Pacman在每个时间步都位于一个可能的位置，并且只执行一个动作。
   - 如果Pacman和食物在同一位置，则食物在下一个时间步消失。
   - 如果Pacman没有吃掉食物，则食物的位置保持不变。
4. **目标查询：**
   - 对于每个时间步，查询一个模型以检查Pacman是否可以吃到所有的食物。
   - 一旦找到了这样的模型，从中提取一个动作序列并返回。

3.5.2 实验结果

实验中，`foodLogicPlan` 函数在多个测试用例上进行了验证：

```
Question q5
===========

[LogicAgent] using problem type FoodPlanningProblem
Path found with total cost of 8 in 0.1 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 513
Average Score: 513.0
Scores:        513.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases\q5\foodLogicPlan1.test
```

```
***     Pacman layout:          testSearch
***     solution score:         513
***     solution path:          West East East South South West West East
[LogicAgent] using problem type FoodPlanningProblem
Path found with total cost of 28 in 11.7 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 573
Average Score: 573.0
Scores:        573.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases\q5\foodLogicPlan2.test
***     Pacman layout:          tinySearch
***     solution score:         573
***     solution path:          South South West East East East East North North North
North West West West West West West East East East South South West West West South South
West

### Question q5: 3/3 ###
```

- 在 `testSearch` 布局中，Pacman在0.1秒内找到了总成本为8的路径。这个路径是：West, East, East, South, South, West, West, East。
- 在 `tinySearch` 布局中，Pacman在11.7秒内找到了总成本为28的路径。这个路径是：South, South, West, East, East, East, East, North, North, North, North, West, West, West, West, West, West, East, East, East, South, South, West, West, West, South, South, West。

3.5.3 结论

本部分的实验显示，逻辑规划方法可以有效地解决食物规划问题，即找到一个动作序列使Pacman吃掉所有的食物。但是，与前面的实验相比，这个任务的复杂性增加了，因为除了找到到达目标位置的路径外，还需要考虑如何消耗掉所有的食物。这增加了查询合适模型的计算复杂性。然而，即便如此，该方法在处理不同复杂性的迷宫时仍然表现出了可行性和有效性。

4 3 结束语

通过本实验，我们深入探讨了逻辑编程在Pacman路径规划问题中的应用。我们使用逻辑规则和断言构建了知识库，并利用这些知识实现了不同的路径规划策略。这包括简单的目标达成（到达指定位置），考虑到墙壁和其他障碍物的路径规划，以及更复杂的任务，如吃掉迷宫中的所有食物。

实验结果表明，逻辑编程方法能够成功应用于解决这类路径规划问题。我们所实现的方法不仅能够找到从起点到终点的有效路径，还能够处理更复杂的场景，例如在迷宫中消耗所有食物。在处理这些问题时，逻辑编程提供了一种结构化和形式化的方法来表示和处理问题的不同方面，如Pacman的动作、位置、迷宫的布局、食物的位置等。

但是，我们也注意到，随着问题复杂性的增加（例如，更大的迷宫和更多的食物），所需的计算时间也会增加。这表明，在实际应用中，我们可能需要考虑优化方法和策略，以提高逻辑编程在更复杂场景下的性能和可扩展性。

总之，本实验增强了我们对逻辑编程在实际问题中应用的理解和认识。未来，我们可以探索更多的优化策略和应用领域，以充分发挥逻辑编程的潜力。