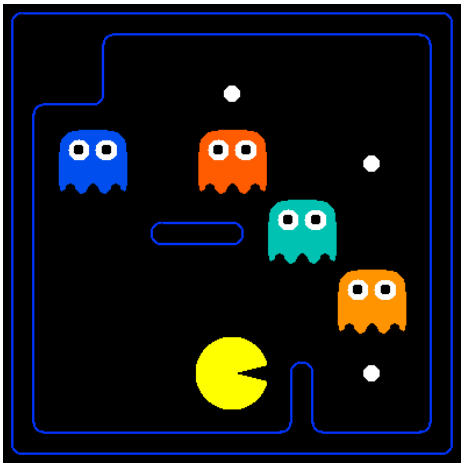
11 Logic and Classical Planning I



本节实验目标：熟悉逻辑推理语句，并进行一次逻辑推理任务

报告说明（重要）：本部分“逻辑”包含两周内容，本节课为第一周内容，请留意该文档目录中标有“报告应实现任务[分值]”的部分，请在实现后及时保存文件和运行结果截图，在下一周课程报告中一并提交。

Introduction
- 你将编辑的文件
- Expr 类
- 函数 `conjoin` and `disjoin`
- 项目的符号名称约定 (重要!)
- 吃豆人物理世界的变量符号
- SAT Solver Setup

Question 1: Logic Warm-up (已实现)
Question 2: Logic Workout 报告应实现任务[1分]
Question 3: Pacphysics and Satisfiability 报告应实现任务[1分]
Question 4: Path Planning with Logic 报告应实现任务[1分]
- 提示

本节课实现内容保存说明 - 为提交第五次上机报告做准备

Introduction

在这堂实验课中，你将使用/编写简单的 Python 函数来**生成**描述 Pacman 物理世界（又名 pacphysics）的**逻辑语句**。 然后，你将**使用 SAT 求解器 pycosat 来解决一些**逻辑推理任务。

与之前一样，此堂实验课相关代码包括一个autograder，可以使用以下命令运行：

```
python autograder.py
```

该项目的代码由几个 Python 文件组成，其中一些你需要阅读和理解才能完成上机练习，而其中一些你可以忽略。

你将编辑的文件

logicPlan.py	**你编辑各种逻辑agent的代码所在位置**

你可能想查看的文件

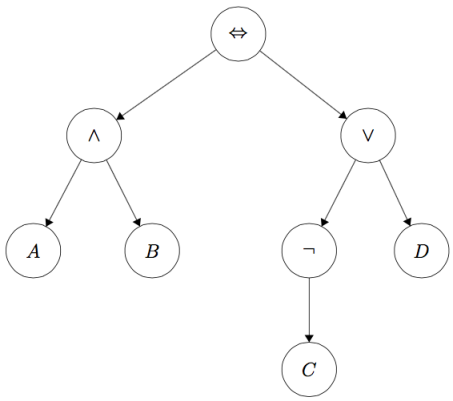
logic.py	**这里有几个有用的实用程序函数可用于处理逻辑。**
logicAgents.py	逻辑规划中 Pacman 在这个项目中将遇到的特定问题。
pycosat_test.py	快速测试主要功能，检查 pycosat 模块是否安装正确。
game.py	Pacman 世界的内部模拟器代码。 **你可能想在这里查看的唯一内容是 Grid 类。**
test_cases/	包含每个问题的测试用例的目录

你不需要（也最好不要）编辑的文件

pacman.py	The main file that runs Pacman games.
logic_util.py	Utility functions for logic.py
util.py	Utility functions primarily for other projects.
logic_planTestClasses.py	Project specific autograding test classes
graphicsDisplay.py	Graphics for Pacman
graphicsUtils.py	Support for Pacman graphics
textDisplay.py	ASCII graphics for Pacman
ghostAgents.py	Agents to control ghosts
keyboardAgents.py	Keyboard interfaces to control Pacman
layout.py	Code for reading layout files and storing their contents
autograder.py	Project autograder
testParser.py	Parses autograder test and solution files
testClasses.py	General autograding test classes

Expr 类

你将使用在 `logic.py` 中定义的 `Expr` 类来构建命题逻辑语句。 `Expr` 对象是在每个结点处具有逻辑运算符（∧、∨、¬、→、↔）并在叶子处具有文字（A、B、C、...）的树。 例如，这个语句 (A ∧ B) ↔ (¬ C ∨ D) 将被表示为树:



要将一个名为“A”的符号**定义为命题词**，请像这样调用构造函数：

```
A = Expr('A')
```

`Expr` 类允许你使用 Python 运算符来构建这些表达式。 以下是**可用的 Python 运算符**及其含义：

- `~A`: **¬ A**
- `A & B`: **A ∧ B**
- `A | B`: **A∨B**
- `A >> B`: **A → B**
- `A % B`: **A ↔ B**

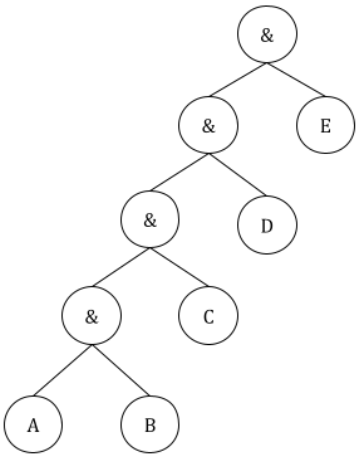
因此，**要构建复合语句 A ∧ B**，你可以输入：

```
A = Expr('A')
B = Expr('B')
a_and_b = A & B
```

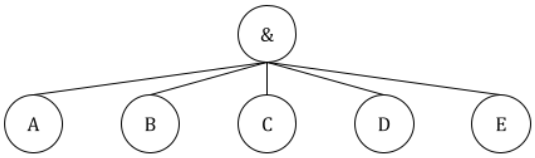
请注意，该示例中赋值运算符左侧的`A`只是一个 Python 变量名，用其它变量名，如`symbol1 = Expr('A')`也是可以的。

函数 `conjoin` and `disjoin`

最后要注意的重要一点是，你**应尽可能使用** `conjoin` **和** `disjoin` **函数**。请查看logic.py中这两个函数的使用方式。 `conjoin` **创建一个链式** `&` **（逻辑与）表达式**，而 `disjoin` **创建一个链式** `|` **（逻辑或）表达式**。 假设你想检查条件 A、B、C、D 和 E 是否都为真，一个简单方法是编写 `condition = A & B & C & D & E` ，但这实际上转换为 `((((A & B) & C) & D) & E)` ，它创建了一个非常嵌套的逻辑树 （下图中的 (1)），它会成为调试的噩梦。 相反， `conjoin([A,B,C,D,E])` 会生成一棵扁平树（参见下图中的 (2)）。



(1) A & B & C & D & E



(2) conjoin([A, B, C, D, E])

项目的符号名称约定 (重要!)

在你的代码中，请使用以下变量命名约定：

通用**规则**

- 当我们引入**变量**时，它们必须以**大写字符开头**（包括 `Expr` 对象)。
- 只有这些字符应出现在变量名称中： `A-Z` ， `a-z` ， `0-9` ， `_` 、 `^` ， `[` , `]` 。
- 逻辑连接字符（ `&` 、 `|` ）**不得出现在变量名**中。因此， `Expr('A & B')` 是非法的，因为它试图创建一个名为 `'A & B'` 的符号。我们可以使用 `Expr('A') & Expr('B')` 来表达。

吃豆人物理世界的变量符号

可以使用PropSymbolExpr()函数创建组合式的命题词，例如：

- `PropSymbolExpr(pacman_str, x, y, time=t)` 将创建这样一个命题词：Pacman 在时间 t 是否位于 (x, y)，**变量名写成** `P[x,y]_t` 。
- `PropSymbolExpr(wall_str, x, y)` ：墙是否在 (x, y) 处，**变量名写成** `WALL[x,y]` 。
- `PropSymbolExpr(action, time=t)` ： Pacman 是否在时间 t 采取行动 `action` ，其中 `action` 是 `DIRECTIONS` 的一个，**变量名写成如** `North_t` 。
- 一般情况下，函数 `PropSymbolExpr(str, a1, a2, a3, a4, time=a5)` 创建表达式 `str[a1,a2,a3,a4]_a5` ，其中 `str` 只是一个字符串。

在 `logic.py` 中有关于 `Expr` 类的额外、更详细的文档说明。

SAT Solver Setup

SAT（可满足性）求解器输入对世界规则进行编码的逻辑表达式，并返回一个模型（对逻辑符号的true和false的分配），如果存在这样的模型，则该模型满足输入的该表达式。为了有效地从表达式中找到可能的模型，我们**利用了 pycosat 模块**。

不幸的是，这**需要在每台机器上安装这个模块/库**。

要在你的 python 环境中安装此软件，请按照以下步骤操作：

在实验室电脑的命令行运行： `pip install pycosat` 。

测试 pycosat 安装：

解压本堂课的项目代码并切换到项目代码目录后，运行：

```
python pycosat_test.py
```

这应该输出：

[1, -2, -3, -4, 5]。

如果你对此设置有任何疑问，请告诉我们。**这对于完成项目至关重要**，我们不希望你花时间与此安装过程作斗争。

Question 1: Logic Warm-up (已实现)

练习用 `Expr` 数据类型来表示命题逻辑语句。 **你将在 `logicPlan.py` 中看到以下功能的实现，帮助你了解我们这个项目怎么创建命题词和语句：**

- `sentence1()`： 创建一个 `Expr` 实例（如A = logic.Expr('A'); Expr, disjoin, conjoin都在logic.py里定义，因此在logicPlan.py中需要以logic.XXX，如logic.disjoin(...)调用），**表示以下三个句子为真。** 不做任何逻辑式上的简化，只是把它们按这个顺序放在一个列表（如）中，然后返回这个列表。 列表中的元素应与以下三个语句一一对应。
  - A ∨ B
  - ¬ A ↔ (¬ B ∨ C)
  - ¬ A ∨ ¬ B ∨ C
- `sentence2()`： 创建一个 `Expr` 实例，**表示以下四个句子为真。** 同样，不要做任何逻辑式上的简化，只需将它们按此顺序放在一个列表中，然后返回这个列表。
  - C ↔ (B ∨ D)
  - A → (¬ B ∧ ¬ D)
  - ¬ (B ∧ ¬ C) → A
  - ¬ D → C
- `sentence3()`： 使用 `PropSymbolExpr` 构造函数创建命题词 `PacmanAlive[0]`, `PacmanAlive[1]`, `PacmanBorn[0]`, 和 `PacmanKilled[0]`, **并创建一个 `Expr` 实例，该实例将以下三个句子按此顺序编码为命题逻辑放在一个列表中**作为函数返回，不做任何简化：
  1. Pacman 在时间点 1 是活着的当且仅当他在时间点 0 还活着并且他在时间点 0 没有被吃掉，或者他在时间 0 不活着并且他在时间点 0 出生。
  2. 在时间点 0，吃豆人不能既活着又出生。
  3. 吃豆人在时间点 0 出生。
- **`findModel(sentence)`**：这个函数需**使用 `to_cnf`** （也在logic.py中定义）将输入的句子转换为Conjunctive Normal Form（SAT求解器所需的形式）。然后使用 `pycoSAT` （也在logic.py中）将其传递给SAT求解器，以找到满足 `sentence` 为真时的各符号的赋值，即模型（如找不到可满足的模型，该函数返回False）。模型是表达式中符号的字典以及对应的 True 或 False 分配。你可以通过在 Python 中（项目所在文件夹）打开一个交互式python会话窗口（即在命令行输入python然后按回车键）并运行 `findModel(sentence1())` 以及对其他两个类似的查询，在 `sentence1()`、`sentence2()` 和 `sentence3()` 上测试你的代码(如下)。它们符合你的想法吗?

  ```
  >>> from logicPlan import *
  >>> findModel(sentence1())
  {A: False, B: True, C: True}
  >>> findModel(sentence2())
  False
  >>> findModel(sentence3())
  {PacmanAlive[1]: True, PacmanAlive[0]: False, PacmanKilled[0]: False, PacmanBorn[0]:
  True}
  ```

要测试和调试你的代码，请运行：

```
python autograder.py -q q1
```

Question 2: Logic Workout 报告应实现任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q2 的运行结果，获得评分器给的全部2分，这道题才算得它占的1分。**

在 `logicPlan.py` 中实现以下三个逻辑表达式：

- `atLeastOne(literals)`：返回单个CNF表达式 (Expr)，仅当输入列表中的至少一个表达式为真时才为真。 每个输入表达式都是一个文字(literal)。
- `atMostOne(literals)`：返回 单个CNF表达式 (Expr)，仅当输入列表中最多一个表达式为真时才为真。 每个输入表达式都是一个文字。例如：输入[A, B, C]，输出((~A | ~B) & (~A | ~C) & (~B | ~C))。
  提示：可以使用
  `itertools.combinations` （可到itertools.py中查看相关定义）。如果你有 n 个文字，并且最多有一个为真，那么你生成的 CNF 表达式应该是 n(n-1)/2 子句的合取。
- `exactlyOne(literals)`：返回单个CNF表达式 (Expr)，仅当输入列表中的恰好只有一个表达式为真时才为真。 每个输入表达式都是一个文字。例如：输入[A, B, C]，输出((~A | ~B) & (~A | ~C) & (~B | ~C) & (A | B | C | D)) 。(如果你决定调用之前实现的 `atLeastOne` 和 `atMostOne`，请先调用 `atLeastOne` 以通过我们的 q3 的autograder。)

这些方法中都输入一个`Expr`文字列表，并返回一个`Expr`表达式表示输入列表中表达式之间的适当逻辑关系。另一个要求是返回的 `Expr` 必须是 CNF（合取范式）。 **你不能在方法实现中使用 `to_cnf` 函数（或任何辅助函数 `logic.eliminate_implications`、`logic.move_not_inwards` 和 `logic.distribute_and_over_or`）。**

在后面的问题中实现你的planning agents时，你不必担心 CNF，直接将你的表达式发送到 SAT 求解器即可（此时你可以使用Q1中的`findModel`）。 `to_cnf` 实现了参考教材中的CNF转换算法。然而，在某些最坏情况的输入上，该算法的直接实现可能会导致句子大小呈指数增长。事实上， `atMostOne` 的某种非 CNF 实现就是这样一种最坏的情况。因此，**如果你发现自己需要使用 `atLeastOne`、`atMostOne` 或 `exactlyOne` 的功能来解决后面的问题，请确保使用你已经在此处实现的功能，以避免意外**。如果你不这样做，你的代码会很慢，以至于你甚至无法解决没有墙壁的 3x3 迷宫。

你可以使用 `logic.pl_true` 函数来测试表达式的输出。 `pl_true` 接受一个表达式和一个模型，当且仅当给定模型的表达式为真时才返回 True。

要测试和调试你的代码，请运行：

```
python autograder.py -q q2
```

Question 3: Pacphysics and Satisfiability 报告应实现任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q3 的运行结果，获得评分器给的3分中，这道题才算得它占的1分。**

在这个问题中，你将实现基本的 pacphysics 逻辑表达式，并学习如何通过构建适当的逻辑表达式知识库 (KB) 来证明 pacman 在哪里和不在哪里。

在 `logicPlan.py` 中实现以下函数：

- `pacphysics_axioms`：在这里，你需要写逻辑语句添加以下公理到`pacphysics_sentences`。 对于给定的时间步长 t：
  - 对于 `all_coords` 中的所有 (x, y)，添加以下含义（if-then 形式）：如果墙位于 (x, y)，则 Pacman 不在 (x, y, t)
  - Pacman 在时间 t 处恰好处于`non_outer_wall_coords` 之一。
  - Pacman 在时间 t 处恰好采取了`DIRECTIONS` 中的四个动作之一。
- `check_location_satisfiability`：给定一个转换（`x0_y0, action0, x1_y1)`， `action1`和一个`problem`，你将写一个函数**返回一个包含两个模型`(model1, model2)`的元组**。
  - `model1` 将尝试证明给定`x0_y0, action0, action1`，**Pacman 在时间 t = 1 可能位于 (x1, y1)。 值得注意的是，如果 `model1` 为 `False`，我们知道 Pacman 肯定不在(x1, y1)。**
  - `model2` 将尝试证明给定`x0_y0, action0, action1`，**Pacman 在时间 t = 1 时可能不在 (x1, y1)。值得注意的是，如果 `model2` 为 `False`，我们知道 Pacman 肯定在(x1, y1)。**
  - `action1` 对确定 Pacman 是否在该位置没有影响； 它只是为了将你的解与autograder解决方案相匹配。
  - 要实现这个问题，**你需要按顺序将以下表达式添加到你的知识库中**：
    - **在 t = 0 时**：
      - 添加到知识库：吃豆人的当前位置 (x0, y0)

    - 添加到知识库：`pacphysics_axioms(...)` （上面实现的函数）
    - 添加到知识库：吃豆人采取`action0`
    - 添加到知识库：`allLegalSuccessorAxioms(t+1, ...)`
  - **在 t = 1 时**：
    - 添加到知识库：`pacphysics_axioms(...)`
    - 添加到知识库：吃豆人采取`action1`

如果你卡在`check_location_satisfiability`上，回想一下如何证明知识库蕴涵一些查询命题q：我们证明KB∧¬q是不可满足的。 同样，为了证明给定知识库的某个查询 q 是false的，我们证明 KB ∧ q 是不可满足的。 因此，**你的两个模型都应该调用`findModel`**，但一个作用在 KB ∧（Pacman at (x1, y1)）上，另一个作用在 KB ∧ ¬（Pacman at (x1, y1)）上（思考：哪个模型作用在哪个语句呢?）。

转移模型：在这个项目中，我们将使用两种不同的转移模型：

- `allLegalSuccessorAxioms`：这个转移模型假设所有采取的行动都是有效的（没有行动将吃豆人带到有墙的格子中）。 我们将在大部分情况下使用这个函数来生成我们的后继状态，因为它是轻量级的并且优化了运行时间。
- `SLAMSuccessorAxioms`：这个转换模型不假设所有采取的行动都是有效的，这增加了很多计算开销。 我们将使用这个函数为 SLAM 问题生成后继状态。

要记住的一点：在下一周实验课的定位和地图建模问题中，我们将使用 4 位传感器（指示 Pacman 的 NSEW 方向是否有墙）与`allLegalSuccessorAxioms`。 但是对于 SLAM，我们将使用带有`SLAMSuccessorAxioms`的较弱的传感器（它只告诉我们与吃豆人相邻的墙的数量）。

提醒：Pacman 在时间 t 是否在 (x, y) 的变量是 PropSymbolExpr(pacman_str, x, y, t)，墙是否在 (x, y) 是 PropSymbolExpr(wall_str, x, y)，在 t 采取行动是 PropSymbolExpr(action, t)。

要测试和调试你的代码，请运行:

```
python autograder.py -q q3
```

Question 4: Path Planning with Logic 报告应实现任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q4 的运行结果，获得评分器给的4分中，这道题才算得它占的1分。**

Pacman试图找到迷宫的末端（目标位置）。 使用命题逻辑**实现以下方法，规划Pacman到达目标位置的行动序列**：

- **`positionLogicPlan(problem)`**：给定一个`logicPlan.PlanningProblem`的实例，返回Pacman agent的一系列动作字符串。

你不需要实现搜索算法，而是需要在每一时间步为所有可能的位置创建表示pacphysics的命题逻辑语句。 这意味着，在每一时间步中，你应该在为知识库添加所有可能位置的逻辑语句，其中语句不假设Pacman当前所在的位置。

你**需要为知识库编写以下语句**：

- 添加到知识库：初步知识：吃豆人在时间 0 的初始位置
- for t in range(50) [Autograder 不会测试需要 ≥ 50 个时间步长的迷宫布局]
  - Print time step; 这是为了查看代码是否正在运行以及它运行了多远。
  - 添加到 KB：初步知识：Pacman 在时间 t 只能位于`non_wall_coords`中的`exactlyOne`个位置。这类似于`pacphysics_axioms`，但不要使用该方法，因为我们在首先生成可能位置列表时使用 non_wall_coors（以及稍后生成`walls`）。
  - 到目前为止，给定知识库的变量是否有令人满意的赋值？ 使用`findModel`并传入 Goal Assertion 和`KB`。
    - 如果有，则使用`extractActionSequence`返回从开始到目标的一系列动作。
    - 这里，Goal Assertion 是断言 Pacman 在时间步 t 达到目标的表达式。
  - 添加到知识库：Pacman 每个时间步只采取一个动作。
  - 添加到知识库：转换模型语句：调用`pacmanSuccessorStateAxioms(...)`以获取`non_wall_coords`中所有可能的 pacman 位置。

使用以下方法在较小的迷宫上测试你的代码：

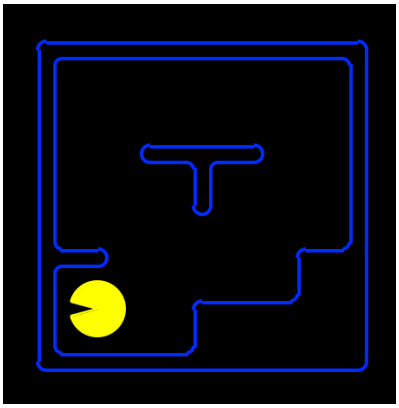
```
python pacman.py -l maze2x2 -p LogicAgent -a fn=plp --frameTime 0.8
```

```
python pacman.py -l tinyMaze -p LogicAgent -a fn=plp --frameTime 0.8
```

要测试和调试你的代码，请运行：

```
python autograder.py -q q4
```

请注意，按照我们布置 Pacman 网格的方式，Pacman 可占用的最左边、最底部的空间（假设那里没有墙）是 **(1, 1)**，如下所示（**不是 (0, 0)**）。



提示

- 如果你遇到困难，请看一下我们理论课逻辑相关课件给出的 pacphysics，也可以尝试阅读 参考教程 第 7 章“基于命题逻辑的Agent”相关内容（尽管跟我们吃豆人的设置不完全一样）。
- 如果它正在寻找长度为 0 或长度为 1 的解决方案：简单地为 Pacman 在给定时间步的位置提供公理就足够了吗？ 是什么阻止它也在其他地方？
- 制定其中一些计划可能需要很长时间。 在你的主循环中有一个print语句很有用，这样你就可以在计算时监控你的进度。
- 如果你的解决方案需要超过几分钟才能完成运行，你可能需要重新检查 `exactlyOne` 和 `atMostOne` 的实现（如果你依赖这些），并确保你使用尽可能少的子句。

本节课实现内容保存说明 - 为提交第五次上机报告做准备

- **第12周上机课结束后，一并提交本部分”逻辑“的报告，该报告包含本节课的Q2,Q3,Q4的代码和运行截图，请按Q2,Q3,Q4大标题下的说明进行截图，保存。**
- **本节课实现的代码文件也请保存，你将需要把其中实现的代码加到第12周相应的代码文件中跟第12周的任务实现一并进行提交。**