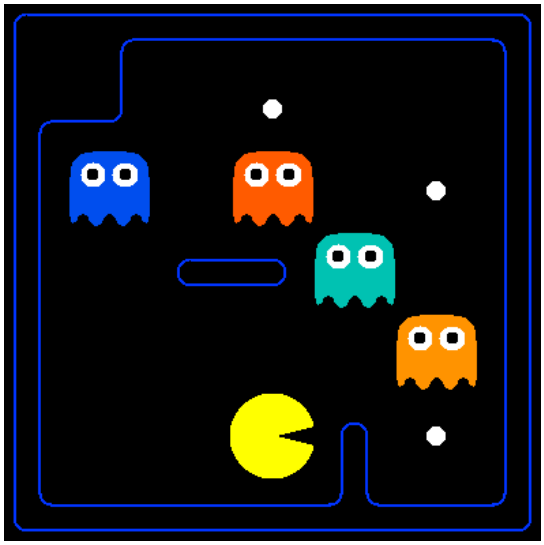
12 Logic and Classical Planning II



本节实验目标：进行几个逻辑推理任务。

报告说明（重要）：本部分“逻辑”包含两周内容，本节课为第二周内容，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，与上周完成的问题一起在对分易提交第五次课程报告。注意，标记有bonus的问题，完成其中一个，即可得到额外的期末总评1分（当然，期末总评最高分还是只能为100分）。

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 1-4 请查看上节课的实验文档 (Q1-3 给出了一个参考实现)

Question 5: Eating All the Food 报告应实现任务[1分]

Q6-Q8所需的辅助函数
- Add pacphysics, action, sensor, and percept information to KB
- Find possible pacman locations with updated KB.
- Find provable wall locations with updated KB

Question 6: Localization 报告应实现任务[1分]

Question 7: Mapping 报告可optionally包含的bonus任务[1分]

Question 8: SLAM 报告可optionally包含的bonus任务[1分]

报告5 “Logic”提交说明

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，如**路径规划**（生成动作序列以到达目标位置并吃掉所有点）、**定位**（在地图中找到自己，给定本地传感器模型）、**地图建模** （从头开始构建地图）和 **SLAM**（同时定位和构建地图）。

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

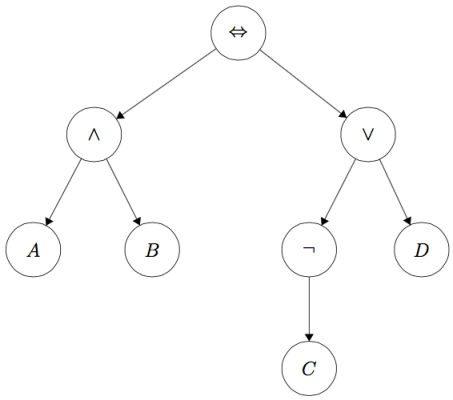
logic.py	**这里有几个有用的实用程序函数可用于处理逻辑。**
logicAgents.py	逻辑规划中 Pacman 在这个项目中将遇到的特定问题。
pycosat_test.py	快速测试主要功能，检查 pycosat 模块是否安装正确。
game.py	Pacman 世界的内部模拟器代码。 **你可能想在这里查看的唯一内容是 Grid 类。**
test_cases/	包含每个问题的测试用例的目录

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**

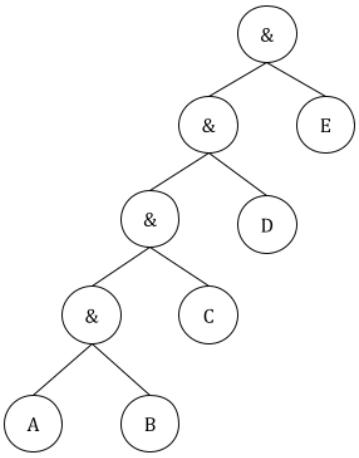
因此，**要构建复合语句 A ∧ B**，你可以输入：

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

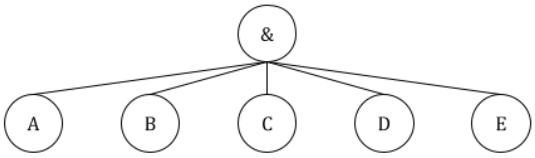
请注意，该示例中赋值运算符左侧的`A`只是一个 Python 变量名，用其它变量名，如`symbol1 = Expr('A')`也是可以的。

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 1-4 请查看上节课的实验文档 (Q1-3 给出了一个参考实现)

Question 5: Eating All the Food 报告应实现任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q5 的运行结果，获得评分器给的全部3分，这道题才算得它占的1分。**

Pacman正试图吃掉游戏板上所有的食物。使用命题逻辑实现以下方法来规划 Pacman 帮助它达到目标的一系列动作。

- `foodLogicPlan(problem)`：给定一个`logicPlan.PlanningProblem`的实例，返回一系列动作字符串供Pacman agent执行。

此问题的一些与Question 4 相同。Question 4 中的注释和提示也适用于此问题。不同的是，你需要实现任何必要的在Q4中没实现的后继状态公理。

Q5与Q4不同的是，在Q5中，你要实现：

- 使用 `PropSymbolExpr(food_str, x, y, t)` 初始化 Food[x,y,t] 变量，每个Food[x,y,t]为真当且仅当在时间 t 在 (x, y) 处有食物。
- 更改目标断言：如果所有食物都已吃完，你的目标断言语句必须为真。提示：当**所有** Food[x,y,t] 在时间 t 为假时，就会发生这种情况。
- 添加食物的后继状态公理：Food[x,y,t+1] 与 Food[x,y,t] 和 Pacman[x,y,t] 之间的关系是什么？对于任何给定的 (x, y, t)，食物的后继状态公理应该只涉及这三个变量。想想食物变量的转移模型是什么样的，并在每个时间步将这些句子添加到你的知识库中。

使用以下方法测试你的代码：

```
python pacman.py -l testSearch -p LogicAgent -a fn=flp,prob=FoodPlanningProblem --frameT
ime 0.8
```

Autograder不会在任何需要超过 50 个时间步的迷宫布局上测试你的代码。

要测试和调试你的代码，请运行：

```
python autograder.py -q q5
```

Q6-Q8所需的辅助函数

对于剩下的问题，我们将依赖以下辅助函数，这些辅助函数将被伪代码引用以进行定位、地图建模和 SLAM。

Add pacphysics, action, sensor, and percept information to KB

- 添加到知识库：`pacphysics_axioms(...)`，你在q3写的。
- 添加到知识库：Pacman 采取由`agent.actions[t]` 规定的动作
- 添加到知识库：`sensorAxioms(...)` 用于定位和地图建模，或`SLAMSensorAxioms(...)` 用于 SLAM。

- 通过调用 `agent.getPercepts()` 获取感知并将感知传递给 `four_bit_percept_rules(...)` 用于定位和映射，或 `num_adj_walls_percept_rules(...)` 用于SLAM。 将生成的 percept_rules 添加到 `KB`。

Find possible pacman locations with updated KB.

- `possible_locations_t = []`
- 迭代 `non_outer_wall_coords`。
  - 我们能证明吃豆人是否在 (x, y) 处吗？ 我们能证明吃豆人是否不在 (x, y) 处吗？ 使用 `findModel` 和 `KB`。 想想你在 q3 中是如何做到的。
  - 如果 Pacman 在时间 t 位于 (x, y) 处存在可满足的变量赋值，则将 (x, y) 添加到 `possible_locations_t`。
  - 添加到 KB: Pacman 在时间 t 可证明在的位置集（x, y）。
  - 添加到 KB：在时间 t，Pacman 可证明不在的位置集 (x, y)。

Append `possible_locations_t` to `possible_locs_by_timestep`。

Find provable wall locations with updated KB

- 迭代 `non_outer_wall_coords`。
  - 我们能证明一堵墙是否在 (x, y) 处吗？ 我们能证明一堵墙是否不在 (x, y) 处吗？ 使用 `findModel` 和 `KB`。 想想你在 q3 中是如何做到的。
  - 添加到 KB 并更新 `known_map`：(x, y) 可证明有墙的位置。
  - 添加到 KB 并更新 `known_map`：(x, y) 可证明没有墙的位置。
- Append `copy.deepcopy(known_map)` to `known_map_by_timestep`.

Question 6: Localization 报告应实现任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q6 的运行结果，获得评分器给的全部4分，这道题才算得它占的1分。**

Pacman 以已知地图开始，但起始位置未知。 它有一个 4 位传感器，可返回其 NSEW 方向是否有墙。 （例如，1001 表示 pacman 的北和西方向有一堵墙，这 4 位使用带有 4 个布尔值的列表表示。）通过跟踪这些传感器读数和它在每个时间步采取的动作，Pacman 能够确定它的位置。 你将编写有助于 Pacman 确定其在每个时间步的可能位置的句子，具体要实现 logicPlan.py中的：

- **`localization(problem, agent)`**：给定 `logicPlan.LocalizationProblem` 的实例和 `logicAgents.LocalizationLogicAgent` 的实例，对于 0 和 `agent.num_steps-1` 之间的时间步长 t，重复生成 t 处可能位置 (x_i, y_i) 的列表：[(x_0_0, y_0_0), (x_1_0, y_1_0), ...]。 请注意，你无需担心generator是如何工作的，因为该行已经为你编写好了。返回列表（列表元素也是列表）：[[(x_0_0, y_0_0), (x_1_0, y_1_0), . ..], [...], ...]。

为了让 Pacman 在定位期间利用传感器信息，你将使用已经为你实现的两种方法： `sensorAxioms` -- 即 Blocked[Direction]_t ↔ [(P[x_i, y_j]_t ∧ WALL[x_i+dx, y_j+dy]) ∨ (P[x_i', y_j']_t ∧ WALL[x_i'+dx, y_j'+dy]) ...] -- 和 `four_bit_percept_rules`，将时间t的感知转换为逻辑语句

要通过autograder，请根据我们以下的伪代码实现：

- 添加到 `KB`：墙在哪里（`walls_list`）和不在哪里（不在 `walls_list` 中）。
- 对于 t in `range(agent.num_timesteps)`：
  - **Add pacphysics, action, sensor, and percept information to KB（在这个文档Q5后面有描述）.**
  - **Find possible pacman locations with updated KB.**
  - 在时间步 t 对当前agent操作调用 `agent.moveToNextState(...)`。
  - 添加到 `KB`： `allLegalSuccessorAxioms(...)`。
  - 提示：检查蕴含的结果是否相互矛盾（即 KB 蕴含 A 且蕴含 ¬ A）。 如果是这样，可以试着print一些中间产出以帮助调试。
- 返回 `possible_locs_by_timestep`

注意 (i)：我们采取这一步是因为将单元文字添加到 KB 有助于加快未来的推理。

如果你卡在“Find possible pacman locations with updated KB”这一步，想想你在 question 3 的可满足性函数中做了什么。

要测试和调试你的代码，请运行：

```
python autograder.py -q q6
```

Question 7: Mapping 报告可optionally包含的bonus任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q7 的运行结果，获得评分器给的全部3分，这道题可为你期末总评加1分bonus（Q7 Q8最多加1分，总评最多100分）。**

Pacman 现在知道他的起始位置，但不知道墙在哪里（除了外部坐标的边界是墙这一事实之外）。 与定位类似，它有一个 4 位传感器，用于返回其 NSEW 方向是否有墙。 你将编写帮助 Pacman 确定墙壁位置的语句，具体要实现logicPlan.py中的：

- `mapping (problem, agent)`：给定`logicPlan.MappingProblem`的实例和`logicAgents.MappingLogicAgent`的实例，对于在 0 和`agent.num_steps-1`之间的时间步长 t 重复产生关于地图 [[1, 1, 1, 1], [1, -1, 0, 0], ...] 在 t 的知识。 请注意，你无需担心generator是如何工作的，因为该行已经为你编写好了。返回`known_maps_by_timestep`，格式为[known_map_0, known_map_1, .. 。]， 在哪里：
  - known_map_t 是一个大小为 (problem.getWidth()+2, problem.getHeight()+2) 的二维数组（一个元素也为列表的列表）
  - 如果 (x, y) 在时间步 t 被确认是墙，则 known_map_t 的每个条目为 1，如果 (x, y) 被确认不是墙，则为 0，如果 (x, y) 仍然不能确认是墙，则为 -1
  - 当不能证明 (x, y) 是一堵墙并且不能证明 (x, y) 不是一堵墙时，就会产生歧义。

要通过autograder，请根据我们的伪代码实现：

- 获取 Pacman 的初始位置`(pac_x_0, pac_y_0)`，并将其添加到`KB`。
- 对于 t in `range(agent.num_timesteps)`：
  - **Add pacphysics, action, sensor, and percept information to KB（在Q6-Q9所需的辅助函数有所描述）**
  - **Find provable wall locations with updated KB**
  - 在时间步 t 对当前agent操作调用`agent.moveToNextState(...)`。
  - 添加到`KB`：`allLegalSuccessorAxioms(...)`。
- 返回`known_map_by_timestep`

要测试和调试你的代码，请运行：

```
python autograder.py -q q7
```

Question 8: SLAM 报告可optionally包含的bonus任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q8 的运行结果，获得评分器给的全部4分，这道题可为你期末总评加1分bonus（Q7 Q8最多加1分，总评最多100分）。**

有时吃豆人是真的迷路了，同时也处于黑暗之中。在 SLAM (Simultaneous Localization and Mapping) 中，Pacman 知道他的初始坐标，但不知道墙壁在哪里。在 SLAM 中，Pacman 可能会无意中采取非法行动（例如，当北边有墙时还向北走），这将随着时间的推移增加 Pacman 位置的不确定性。此外，在我们的 SLAM 设置中，Pacman 不再有一个 4 位传感器来告诉我们四个方向是否有墙，而只有一个 3 位传感器来显示他相邻的墙的数量。 （这有点像 wifi 信号强度条；000 = 不与任何墙相邻；100 = 恰好与 1 堵墙相邻；110 = 恰好与 2 堵墙相邻；111 = 恰好与 3 堵墙相邻。这 3 位表示为3 个布尔值的列表。）因此，你将使用`SLAMSensorAxioms`和`num_adj_walls_percept_rules`，而**不是**使用`sensorAxioms`和`four_bit_percept_rules`。你将编写语句帮助 Pacman 确定 (1) 他在每个时间步的可能位置，以及 (2) 墙壁的位置，具体要实现logicPlan.py中的

- `slam (problem, agent)`：给定一个`logicPlan.SLAMProblem`和`logicAgents.SLAMLogicAgent`的实例，返回一个包含两项的元组：
  - 每个时间步 t 的 known_map 列表（格式与Question 6（mapping）相同）
  - 每个时间步 t 可能的 pacman 位置列表（格式与Question 5（localization）相同）

要通过autograder，请根据我们的伪代码实现功能：

- 获取 Pacman 的初始位置 `(pac_x_0, pac_y_0)`，并将其添加到 `KB`。
- 对于 t in `range(agent.num_timesteps)`：
  - Add pacphysics, action, sensor, and percept information to KB. Use `SLAMSensorAxioms`, `SLAMSuccessorAxioms`, and `numAdjWallsPerceptRules`.
  - **Find provable wall locations with updated KB (在Q6-Q9所需的辅助函数有所描述)** 确保在下一步之前将其添加到KB。
  - **Find possible pacman locations with updated KB.**
  - 在时间步 t 对当前agent操作调用 `agent.moveToNextState(...)`。
  - 添加到 `KB`： `SLAMSuccessorAxioms(...)`。
- 返回 `known_map_by_timestep, possible_locs_by_timestep`

要测试和调试你的代码，请运行：

```
python autograder.py -q q8
```

报告5 “Logic”提交说明

- 按照第11周和本周第12周的Question 2-6 (optionally, bonus questions 7-8) 的要求实现代码和获得运行结果截图
- **提交压缩包命名为”姓名_学号_报告序号.zip”** (如“彭振辉_2106666_报告5.zip”)
- 压缩包应**包含内容：**
  - **已实现的完整项目文件夹”Project_5_Logic_full“**
    - 其中应包含 Question 2-6 (optionally, bonus questions 7-8) 的要求实现代码
  - **一个doc或pdf说明文档，上面需要有：**
    - 开头一段说明 “**整体实现参考 + 2-3句简要体会（如教训、思路、拓展应用等）**”，如：-
      - “自行实现。挑战最大的是xxx内容，初始时报了什么错，通过什么方式解决，该部分的实现思路为xxx”
      - “xxx内容参考xxx同学/xxx网址。思考不出算法思路，探究后学习到了什么方法。”
    - **要求实现的5个任务(and optionally, 2个bonus任务)的成功运行截图，说明截图对应任务。**