AI HW3

R12922029 陳姵安

1. **Show your autograder result for each question in the report.**
    a. **autograder results**
        - **Q1 – Logic Warm-up**

```
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: 10/10 ###
```

        - **Q2 – Logic Workout**

```
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: 10/10 ###
```

        - **Q3 – Pacphysics and Satisfiability**

```
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: 10/10 ###
```

■ Q4 – Path Planning with Logic

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

[LogicAgent] using problem type PositionPlanningProblem
0
1
2
Path found with total cost of 999999 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
0
1
2
3
4
5
6
7
8
Path found with total cost of 999999 in 0.4 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
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
Path found with total cost of 20 in 52.4 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

### Question q4: 10/10 ###
```

■ Q5 – Eating All the Food

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

[LogicAgent] using problem type FoodPlanningProblem
0
1
2
3
4
5
6
7
8
Path found with total cost of 999999 in 0.3 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
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
Path found with total cost of 999999 in 14.8 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 East North North North North West West West West West West East East East South South West West West South South West

### Question q5: 10/10 ###
```

2. Describe your algorithm for each question in the report.

a. Q1 – Logic Warm-up (2%)

This part is designed for us to practice using Python operators to represent symbolic mathematical expressions and to construct logical sentences.

- **Sentence1**

  Use symbolic mathematical expressions to express A, B, and C. Then, use Python operators to build up the sub-sentences to express

  1. (A or B),
  2. ((not A) if and only if ((not B) or C)), and
  3. ((not A) or (not B) or C)

  , respectively. Finally, use ***conjoin***, which functions as sequential "AND", to combine all the sub-sentences to an instance to make the above expressions all true.

- **Sentence2**

  Use symbolic mathematical expressions to express A, B, C, and D. Then, use Python operators to build up the sub-sentences to express

  1. (C if and only if (B or D)),
  2. (A implies ((not B) and (not D))),
  3. ((not (B and (not C))) implies A), and
  4. ((not D) implies C)

  , respectively. Finally, use ***conjoin***, which functions as sequential "AND", to combine all the sub-sentences to an instance to make the above expressions all true.

- **Sentence3**

  Use symbolic mathematical expressions to express 'PacmanAlive_0', 'PacmanAlive_1', 'PacmanBorn_0', and 'PacmanKilled_0' as A, B, C, and D, respectively. Then, use Python operators to build up the sub-sentences to express

  1. Pacman is alive at time 1 if and only if Pacman was alive at time 0 and it was not killed at time 0 or it was not alive at time 0 and it was born at time 0.
  2. Pacman cannot both be alive at time 0 and be born at time 0.
  3. Pacman is born at time 0.

  , respectively. Finally, use ***conjoin***, which functions as sequential "AND", to combine all the sub-sentences to an instance to make the above expressions all true.

- **findModel**

  This function aims to find a combination of variable values that makes the sentence TRUE. It accepts a propositional logic sentence, represented as an Expr instance, as input and then searches for possible solutions. If no solution is found, it returns '***False'***. Otherwise, it provides a dictionary with possible solutions, such as ***{B: True, A: True}***.

  For instance, these are the results for sentence1, sentence2, and sentence3.

```
print(findModel(sentence1()))
print(findModel(sentence2()))
print(findModel(sentence3()))
```
✓ 0.0s

```
{B: True, A: False, C: True}
False
{PacmanKilled_0: False, PacmanAlive_0: False, PacmanAlive_1: True, PacmanBorn_0: True}
```

- **findModelUnderstandingCheck**

  This function is designed to ensure that we understand the operation of the ***findModel()*** function. Based on the solution-finding capability of ***findModel()***, this function would return the result of ***findModel(Expr('a'))***, which simulates the scenario where lowercase expressions are permitted. In this scenario, the expected result of ***findModel(Expr('a'))*** would be ***{a: True}***.
  Within the ***findModel()*** function, ***pycoSAT*** is utilized to determine whether a cnf_sentence, which is an instance of the Expr class, has a solution. This process emulates the functionality of ***findModel()*** by verifying the validity of an expression and using ***pycoSAT*** to ascertain if a cnf_sentence is solvable.

- **Entails**

  In mathematical logic, entailment refers to a relationship where one statement logically follows from another. Formally, if two statements are A and B, we say that A entails B (written as A ⊨ B) if every situation in which A is true, B must also be true. However, when A is False, B is not necessarily False.
  Therefore, I used ***premise & ~(conclusion)*** to represent entailments, if ***findModel(premise & ~(conclusion))*** returns False, it means that it cannot find a combination of premise and conclusion that can generate a True result, otherwise, it meet the cases the support the entailment relationship between A and B. Therefore, ***premise & ~(conclusion)*** can distinguish the cases whether A entails B.

  In mathematical logic, entailment refers to a relationship where one statement logically follows from another. Formally, if two statements are A and B, we say that A entails B (written as A ⊨ B) if in every situation in which A is true, B must also be true. However, when A is false, B is not necessarily false. Therefore, I use the expression ***premise*** ∧ ¬***(conclusion)*** to represent entailments.
  If ***findModel(premise*** ∧ ¬***(conclusion))*** returns false, it means that it cannot find a combination of premise and conclusion that yields a true result. Otherwise, it meets the cases that support the entailment relationship between A and B. Thus, ***premise*** ∧ ¬***(conclusion)*** effectively distinguishes whether A entails B.

Premise	conclusion	entailment	~(premise & ~(conclusion))	premise & ~(conclusion)
T	T	T	T	F
T	F	F	F	T
F	T	T	T	F
F	F	T	T	F

- **plTrueInverse**

  This function aims to check whether the values in assignments make (not inverse_statement) true. If it is true, then it returns True; otherwise, it returns False. Therefore, simply returning ***pl_true(~inverse_statement, assignments)*** suffices. Here are some example outputs:

```
Inverse_statement:  ~A
Assignments:  {A: True}
Returning:  True
Inverse_statement:  A
Assignments:  {A: True}
Returning:  False
```

b. Q2 – Logic Workout (2%)

- **atLeastOne**

  Since that ***disjoin()*** functions as sequential “OR” here, simply returning ***disjoin(literals)*** suffices here. Any Expr in literals is True, ***disjoin(literals)*** will be True.

- **atMostOne**

  The function aims to check whether at most one ***Expr*** is true; i.e., any cases where more than one ***Expr*** is true are not allowed. Therefore, I first create a list that contains the clause ***~Expr | ~Exp***r for any two Expr expressions. This list is designed to hold a series of clauses where each clause enforces that no two literals from the literals list can be true simultaneously. As a result, returning ***conjoin(result)*** fulfills the target of the ***atMostOne()*** function.

- **exactlyOne**

  If the Expr can fulfill both the above functions, that means it fulfill the request of ***exactlyOne()*** function. Therefore, returning ***conjoin([atLeastOne(literals), atMostOne(literals)])*** suffices here.

c. Q3 – Pacphysics and Satisfiability (2%)

■ pacmanSuccessorAxiomSingle

This function accounts for legal actions. Its purpose is to check whether the current position of Pacman is the result of a previous position at time t-1 combined with an action taken to move to (x, y). The list ***possible_causes*** has already defined these combinations of previous positions and actions.

Additionally, we can use ***PropSymbolExpr(pacman_str, x, y, time=now)*** to represent Pacman's current position.

Therefore, returning ***PropSymbolExpr(pacman_str, x, y, time=now) % disjoin(possible_causes)*** can verify whether the current position results exclusively from one of the previous positions at time t-1 and the corresponding actions taken to move to (x, y).

■ SLAMSuccessorAxiomSingle

Different from pacmanSuccessorAxiomSingle function, it accounts for illegal actions. The moved_causes_sent encompasses all valid reasons for movement and holds true under three conditions:

1. ***~PropSymbolExpr(pacman_str, x, y, time=last)*** : Pacman was not at (x, y) at time t-1, since the current discussion focuses on Pacman being at (x, y) at time t. Given that the action STOP is not available, theoretically, there shouldn't be a scenario where Pacman remains in the same position across two time points.
2. ***~PropSymbolExpr(wall_str, x, y)*** : There is no wall at (x, y), aligning with the discussion that there shouldn’t be a wall where Pacman is considered to be at time t.
3. ***disjoin(moved_causes)*** : All potential reasons that might have moved Pacman to the current position, which include possible actions taken at t-1 and the coordinates at that time.

Next, the inference covers all possible actions that could fail. If the method being discussed is that Pacman was at position (x, y) at time t-1, then the wall_dir_clause is true under two circumstances if Pacman's action at t-1 would result in hitting a wall:

1. ***PropSymbolExpr(wall_str, x + dx, y + dy)*** : There is a wall at the position after taking a certain action (direction) at time t-1.
2. ***PropSymbolExpr(direction, time=last)*** : Pacman took a certain action at time t-1.

From these inferences, we can derive the failed_move_causes_sent, which holds true under the following two conditions:

1. ***PropSymbolExpr(pacman_str, x, y, time=last***) : Pacman was at (x, y) at time t-1, as previously mentioned.
2. ***disjoin(failed_move_causes)***: All possible reasons for failed movements.

Ultimately, all failed actions are valid under two main conditions:

1. ***PropSymbolExpr(pacman_str, x, y, time=now) % disjoin([moved_causes_sent, failed_move_causes_sent])*** : Pacman's current position satisfies either the valid movement reasons from moved_causes_sent or the failed implications from failed_move_causes_sent.
2. ***Auxiliary_expression_definitions*** : All scenarios where Pacman would hit a wall after taking a certain action at time t-1.

- **pacphysicsAxioms**

  The function pacphysicsAxioms is designed to generate a bunch of physics axioms which includes all of the following:
  - for all (x, y) in ***all_coords***:
    If a wall is at (x, y) --> Pacman is not at (x, y)
  - Pacman is at exactly one of the squares at timestep t.
  - Pacman takes exactly one action at timestep t.
  - Results of calling ***sensorModel(...)***, unless None.
  - Results of calling ***successorAxioms(...)***, describing how Pacman can end in various locations on this time step. Consider edge cases. Don't call if None.

  First, create a list of possible positions that Pacman can be in at timestep t. Then, for all (x, y) in all_coords, add a sentence to the logic sentence that states if a wall is at (x, y), then Pacman is not at (x, y).
  Next, add a sentence to the logic sentence that ensures Pacman is at exactly one of the squares at timestep t. Then, add a sentence to the logic sentence that ensures Pacman takes exactly one action at timestep t.
  If sensorModel is not None, add the results of calling ***sensorModel(...)*** to the logic sentence. If t > 0 and successorAxioms is not None, add the results of calling ***successorAxioms(...)*** to the logic sentence. Return the logic sentence containing all of the above statements.

- **checkLocationSatisfiability**

  Given a transition (x0_y0, action0, x1_y1), action1, and a problem, this is a function that will return a tuple of two models (model1, model2) where model1 is a model where Pacman is at (x1, y1) at time t = 1 and model2 is a model where Pacman is not at (x1, y1) at time t = 1. First, create a list of all the coordinates that are walls. Then, create a list of all the coordinates that are not walls. Next, create a knowledge base (KB) and add the possible locations for Pacman at time t = 0 and t = 1 using the ***pacphysicsAxioms*** function. Add the location of Pacman at time t = 0, the action of Pacman at time t = 0, and the action of Pacman at time t = 1 to the KB. Find a model where Pacman is at (x1, y1) at time t = 1 and a model where Pacman is not at (x1, y1) at time t = 1 using the findModel function. Return the models where Pacman is at (x1, y1) at time t = 1 and where Pacman is not at (x1, y1) at time t = 1.

d. Q4 – Path Planning with Logic (2%)

The function aims to solve a position planning problem by progressively building a knowledge base (KB) to determine a series of actions that lead to the goal state.
The function aims to solve a position planning problem by progressively building a knowledge base (KB) to determine a series of actions that lead to the goal state.
First, add the initial state to the KB. Then, for each timestep, add the possible locations that Pacman can be at that time step, where there is no wall, and add exactly one of the locations at timestep t to KB. Then, add the possible actions that Pacman can take at that time step and add exactly one of the actions at timestep t to KB. Next, try to find a model where Pacman is at the goal at time t. If a model is found, return the sequence of actions that lead to the goal. If a model is not found, add more information to the KB by adding the possible locations that Pacman can be at time t+1 using the ***pacmanSuccessorAxiomSingle*** function. Repeat the process until a model is found or the maximum number of timesteps is reached.

e. Q5 – Eating All the Food (2%)

The function foodLogicPlan is designed to solve a ***FoodPlanningProblem*** in a grid-based game environment. The goal of this function is to return a list of actions that enable Pacman to eat all of the food present on the grid.
This function is basically the same as ***positionLogicPlan(problem)*** function but adding some additional food settings.
First, add the initial state to the KB. Then, for each timestep, add the possible locations that food will be at that time step, and add exactly one of the locations at timestep t to the KB. Next, add the possible locations that Pacman can be at that time step and add exactly one of the locations at timestep t to the KB.Then, add the possible actions that Pacman can take at that time step and add exactly one of the actions at timestep t to the KB. Add the successor axioms for Pacman at the next time step using the pacmanSuccessorAxiomSingle function. Add the successor axioms for food at the next timestep, where a specific location (x,y) doesn't have food at time t+1 if and only if Pacman was there at time t or if it didn't have food at time t. Finally, add the goal sentence to the KB, which is that all the food sentences are false (there is no food left). Try to find a model where the goal is satisfied at time t. If a model is found, return the sequence of actions that lead to the goal. If a model is not found, repeat the process of adding more information to the KB until a model is found or the maximum number of timesteps is reached.

3. Q1(8%). According to AI Weekly in the lecture, some experts and scholars such as Karl Friston and Yann LeCun believe: "You can't get to AGI with LLMs .” Nowadays, the prospects of LLM are so optimistic. Why do you think these experts have such ideas? Please elaborate on your views.

Here are several reasons that might inform their skepticism, and my elaboration on these points:

1. Generalization: LLMs rely heavily on pattern recognition from training data and lack the ability to generalize beyond these patterns to understand new, untrained contexts.
2. World Understanding: LLMs do not have a genuine understanding of the physical world, as they only manipulate text-based symbols without true comprehension of causality or context.
3. Sensory and Motor Integration: Human intelligence integrates sensory experiences and physical interaction, a feature absent in text-only LLMs.
4. Energy Efficiency: LLMs require substantial computational resources, making them less sustainable compared to the natural efficiency of human learning processes.
5. Active Inference: Friston advocates for AI that mimics biological systems through models like the Free Energy Principle, suggesting AI should actively engage with and adapt to its environment for true intelligence.

In conclusion, while LLMs like GPT have made significant advances in handling and generating human-like text, reaching AGI would likely require overcoming these foundational differences. It entails developing systems that can genuinely understand and interact with the world, learn in an energy-efficient manner, and possess the ability to reason, plan, and make decisions based on an internal model of reality, rather than merely responding based on pre-existing data patterns.

4. Q2(8%). According to the paper ”CLIP-Event: Connecting Text and Images with Event Structures,” in CVPR 2022, after the process of generating the event-centric structured data, how does this work implement contrastive learning? Specifically, how does this work choose the positive and negative samples for contrastive learning?

a. **Positive Sample Selection**:
For a given data point, the positive sample is the one where the text description and the image match correctly in terms of the event structure. That means the text and image both depict the same event, with the image typically showing the primary event mentioned in the text caption. The primary event is determined using various criteria like closeness to the root of the dependency parsing tree and similarity between trigger words in the text and the visual content, as identified by a pretrained model like CLIP.

b. **Negative Sample Selection**:
The negative samples are chosen based on the strategies designed to generate challenging negatives, which include:

- Negative Event Sampling:
  - A confusion matrix is computed for the event type classifier based on the similarity scores between event type labels and the input image.

        - Event types that are frequently confused or share similar visual features with the primary event type are selected as negative event types. These are the challenging cases that the model needs to learn to distinguish.
    - Negative Argument Sampling:
        - Arguments of an event are manipulated by changing their order (right-rotation) to create misalignment between argument roles and their actual meaning.
        - If an event has only one argument, a negative role is sampled based on a confusion matrix from a text argument extraction system.
        - This misordering of arguments creates a negative sample that is structurally different from the correct event structure, presenting a challenge for the model to identify and learn from.

c. **Contrastive Learning with Event Structures**:
   After the positive and negative samples are defined, the learning process involves aligning the text and image representations in such a way that positive pairs are closer in the representation space, and negative pairs are further apart. This is achieved through techniques like:
    - Image-level alignment, using cosine similarity between text and image representations.
    - Entity-level alignment, considering both the mentioned similarity and type similarity.
    - Event-level alignment, using optimal transport to get the minimal distance between text event graph and image event graph representations.

The objective function during training will then use these alignments and distances to optimize the model parameters, encouraging the model to correctly match related text-image pairs and separate unrelated pairs.

In summary, contrastive learning in this work is implemented by defining a clear set of criteria for what constitutes positive and negative samples based on the structure and semantics of events. The model is trained to distinguish between these samples, thereby enhancing its ability to understand and connect text and images based on event structures.

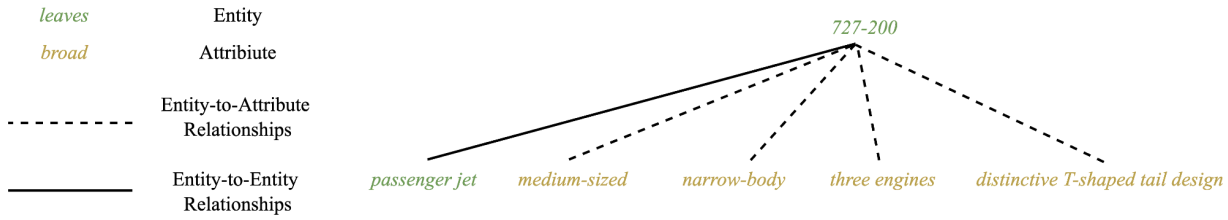
5. Q3(8%). Referring to the paper, what is the main problem with the current description of the VLM pre-training process? Please describe the steps to generate the structured graph- based data in this work.

Current descriptions fall short of structured information that effectively represents the interconnections among entities or attributes linked to a particular category.
The paper proposed an approach called ***Hierarchical Prompt Tuning (HPT)***, which enables simultaneous modeling of both structured and conventional linguistic knowledge. ***First***, they input a few hand-written instructions (instruction T) into LLM with defined [CLASS] and [TYPE].

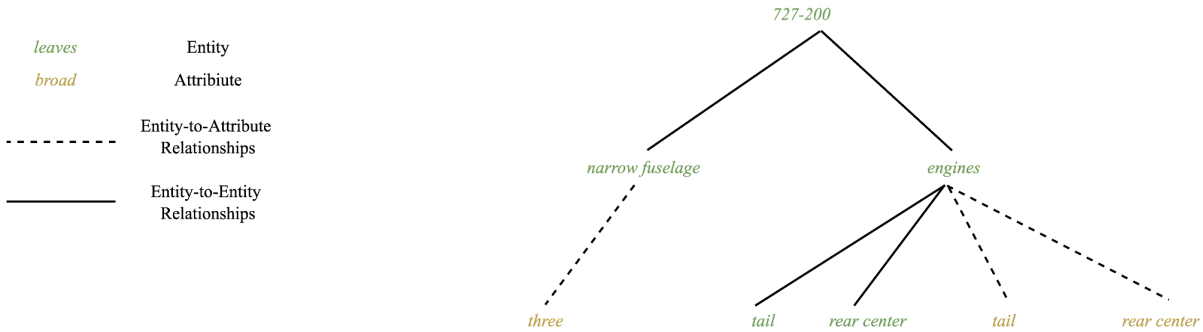
This could guide the LLM to generate human-like category-related descriptions. ***Second***, they input an instruction to specify rules that show the LLM how to reconstruct the structured graph (instruction T'). ***Last but not least***, with the human-like category-related descriptions and instruction T', the LLM could further generate a structured graph.

6. **Q4(8%). Please select one of the datasets provided in this work and visualize two categories.**
    a. **Path to the gpt-generated data: ./data/gpt_data**
    b. **You must show the corresponding description and two graph-based structured data components for each category as shown on the right.**

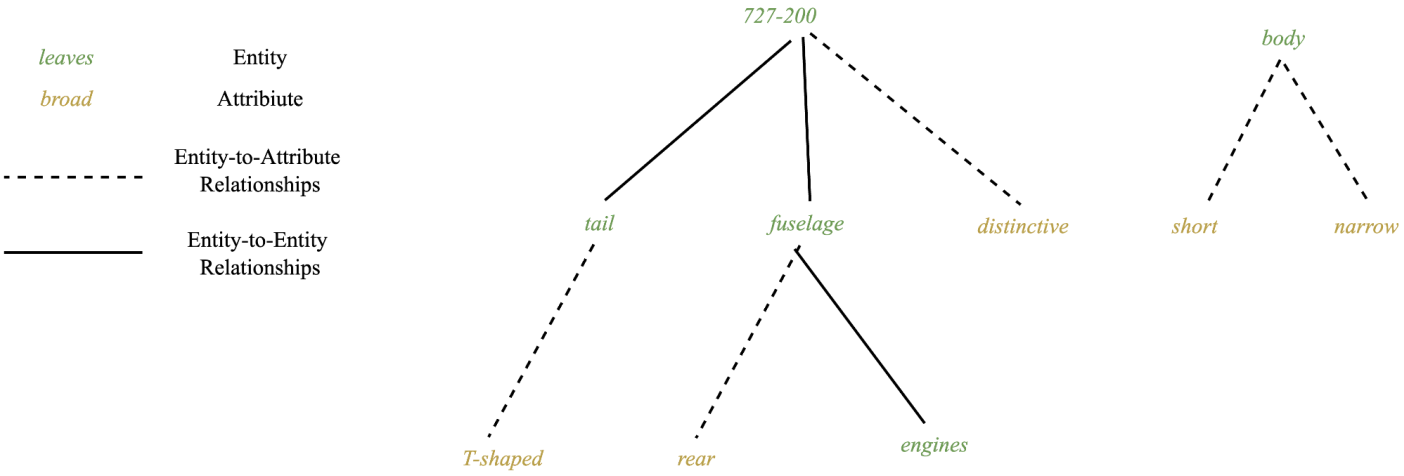
- Category 1 : "727-200" (Class : FGVCAircraft)
    - "The 727-200 is a medium-sized, narrow-body passenger jet with three engines and a distinctive T-shaped tail design."



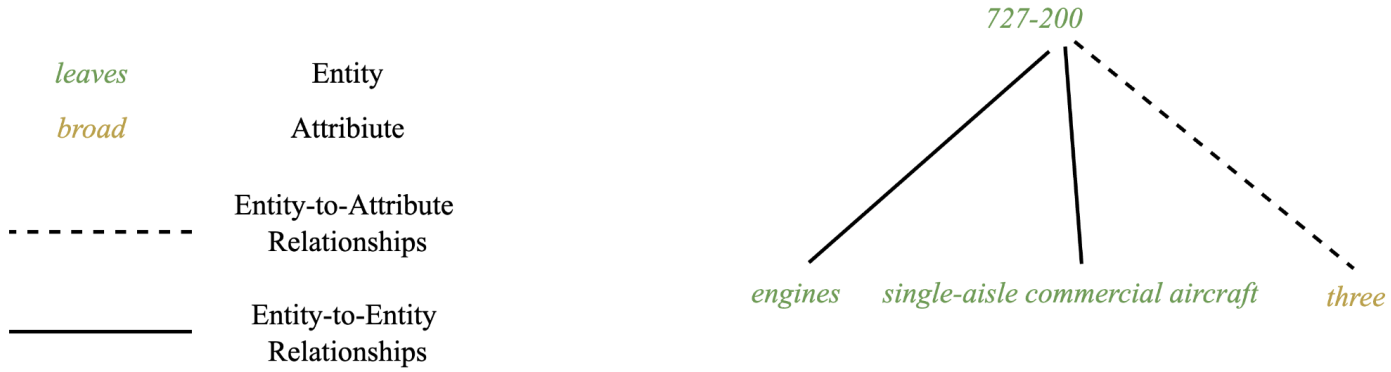
    - "The 727-200 has a narrow, fuselage with three engines - two mounted on the tail and one in the rear center.",



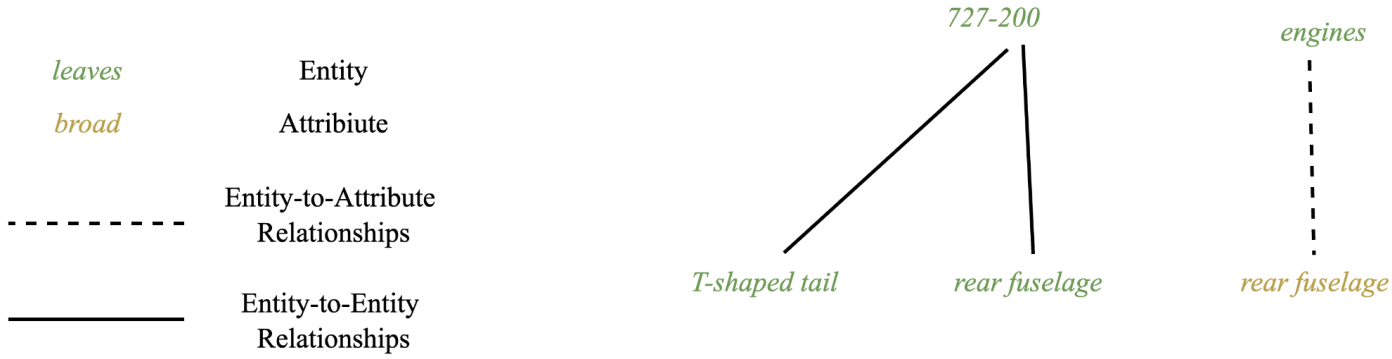
    - "The 727-200 has a distinctive T-shaped tail, three engines mounted on the rear fuselage, and a short, narrow body.",



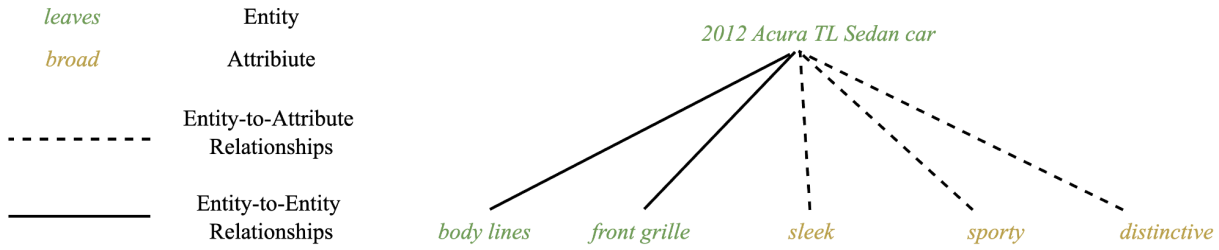
- "The 727-200 aircraft has three engines, a T-shaped tail, and angled wingtips, distinguishing it from other single-aisle commercial aircraft.",



- "The 727-200 aircraft features a T-shaped tail with two engines mounted on the rear fuselage."

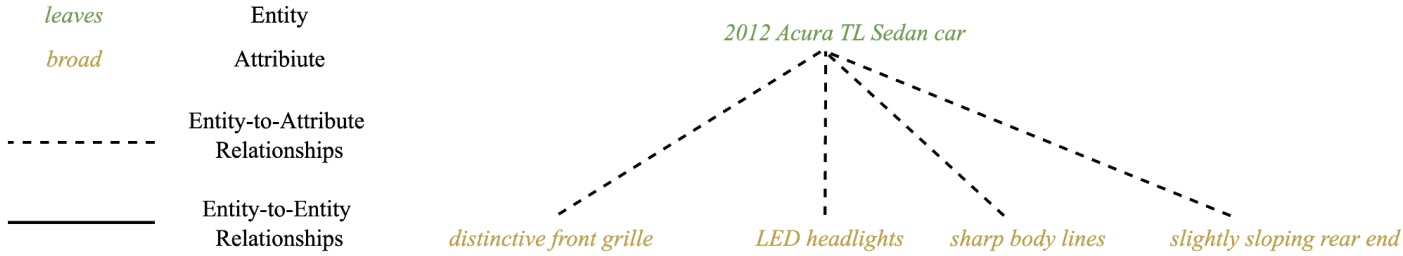


- Category 2 : "2012 Acura TL Sedan" (Class : StanfordCars)
  - "A 2012 Acura TL Sedan car has a sleek and sporty appearance with a distinctive front grille and sharp body lines."

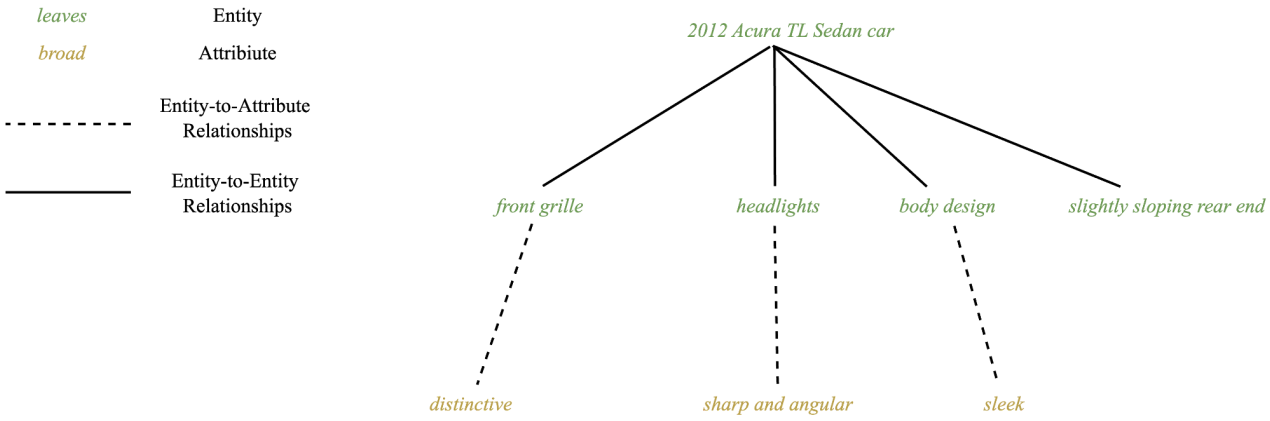


- "Distinct features of the 2012 Acura TL Sedan car for recognition include its unique grille, sleek headlights, and distinctive body lines.",

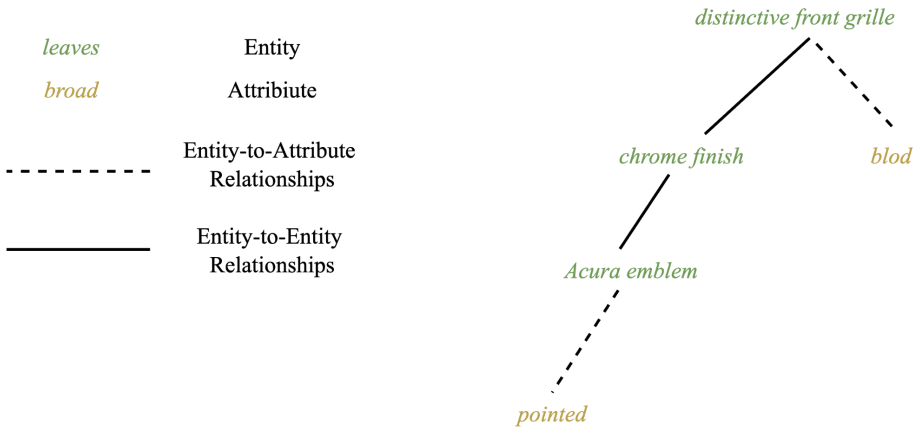
- "2012 Acura TL Sedan has a distinctive front grille, LED headlights, sharp body lines, and a slightly sloping rear end.",



- "The 2012 Acura TL Sedan car has a distinctive front grille, sharp and angular headlights, and a sleek body design.",



- "A distinctive front grille with a bold chrome finish, featuring a pointed Acura emblem at the center of the grille."



7. **Q5(8%). Based on current VLM auxiliary data improvement methods, such as the event-centric structure data in the lecture and the structured linguistic knowledge in this paper, what other deep semantic knowledge do you think humans possess that can be provided to VLM for learning?**

   In my opinion, there are several other types of deep semantic knowledge that humans possess which can be systematically provided to VLMs to further boost their effectiveness.

   1. ***Commonsense Knowledge***: Although this is a broad category, enhancing VLMs with more nuanced commonsense knowledge that covers less common scenarios or deeper everyday reasoning can help bridge the gap between human-like understanding and machine interpretation. An example is '***ESC: Exploration with Soft Commonsense Constraints for Zero-shot Object Navigation***' (https://arxiv.org/pdf/2301.13166.pdf), which demonstrates how commonsense reasoning can enhance the effectiveness of agents performing Vision-and-Language Navigation (VLN) tasks. Though this paper targets VLN tasks rather than VLMs directly, it illustrates the additional layers of information that commonsense can provide
   2. ***Causal Reasoning***: Humans inherently understand cause-and-effect relationships, which could greatly aid VLMs in tasks requiring logical inference or prediction. By embedding causal relationships within the training data, VLMs could learn to predict outcomes or reasons behind certain events in images or texts. For instance, '***Causal Attention for Vision-Language Tasks***' (https://openaccess.thecvf.com/content/CVPR2021/papers/Yang_Causal_Attention_for_Vision-Language_Tasks_CVPR_2021_paper.pdf#:~:text=URL%3A%20https%3A%2F%2Fopenaccess.thecvf.com%2Fcontent%2FCVPR2021%2Fpapers%2FYang_Causal_Attention_for_Vision) presents a novel attention mechanism designed to mitigate the biases caused by spurious correlations in attention-based VLMs. By focusing on the causal relationships within the data, this approach aims to improve the generalization capabilities of the model, an essential factor in enhancing VLM quality.
   3. ***Emotional Intelligence***: Recognizing and responding to emotional cues is a sophisticated aspect of human interaction. Including emotional intelligence in VLMs through training on

data annotated with emotional states or responses could enable models to generate more empathetic and contextually appropriate responses in human-machine interactions. The paper '***Solution for Emotion Prediction Competition of Workshop on Emotionally and Culturally Intelligent AI***' (https://arxiv.org/abs/2403.17683) deliberates on the advantages of integrating emotional intelligence into VLMs.

4. ***Counterfactual Thinking***: Humans often think about alternatives to reality, which is known as counterfactual thinking. VLMs trained to understand and generate counterfactual scenarios could enhance creative content generation and improve problem-solving by considering what could have been instead of just what is. Regarding Counterfactual Thinking, while the papers directly discussing this in the context of VLMs were not found in this search session, the papers on Social and Cultural Context mention the generation of counterfactuals to mitigate biases, which closely relates to training models on alternative scenarios ("***Probing and Mitigating Intersectional Social Biases in Vision-Language Models with Counterfactual Examples***" (https://ar5iv.labs.arxiv.org/html/2312.00825)). The creation of counterfactual examples helps models to learn from diverse and nuanced data, which can enhance their problem-solving capabilities and creativity.
5. ***Ethical and Moral Reasoning***: Embedding ethical considerations and moral reasoning capabilities in VLMs could be revolutionary, especially for applications where decision-making in complex scenarios is required. This involves training on datasets that include dilemmatic situations, where models must learn to weigh different ethical outcomes. The paper '***Training Large Multimodal Language Models with Ethical Values***' (https://papyrus.bib.umontreal.ca/xmlui/handle/1866/32832) explores the ethics embedded within VLMs.