The sample prompts used for each experiment.

FewShot_Prompt01 - Experiment 01 & 02

ONTOLOGY: This ONTOLOGY contains crops, their varieties, and soil types. It includes object properties such as hasSoilTypeforVariety, which links a Variety to its corresponding SoilType, and hasVariety, which relates a Crop to its Variety. The hasVariety property is explicitly declared as the inverse of isVarietyOf, ensuring bidirectional semantics. Additionally, isVarietyOf is defined as a functional property, indicating that a variety belongs to only one crop. The ontology also includes class definitions for Crop, SoilType, and Variety, with Variety specified as a subclass of Crop, establishing a hierarchical relationship.
Also, Variety class has following data properties namely average plant height, average plant height units, maturity period, maturity period units, average yield, average yield unit, and other features. Crop class has data properties such as hasAverageYield, AverageYeildUnit. SoilType class has data properties such as hasmaxSoilph, hasminSoilph.
Identify the **Assertional Axioms** for the above ONTOLOGY in the following JSON format. Populate the JSON format extracting the relevant data from the given PDF description, ensuring accurate domain and range assignments.Below is an example with dummy data for guidance. Use consistent naming and formatting for all elements.

JSON OUTPUT:

```
{
"ClassAssertional": [
{
"Class": "GrowingProblem",
"Individuals": ["Thrips", "BacterialWilt"]
},
{
"Class": "SoilType",
"Individuals": ["ClaySoil", "SandyLoam"]
}
],
"ObjectPropertyAssertional": [
{
"Axiom": "BasmatiRice hasSoilTypeforVariety ClaySoil",
"Domain": "Variety",
"Range": "SoilType"
},
{
"Axiom": "IR64 hasSoilTypeforVariety SandyLoam",
"Domain": "Variety",
"Range": "SoilType"
}
],
"DataPropertyAssertional": [
{
"Axiom": "BasmatiRice averagePlantHeight 110",
"Domain": "Variety",
"Range": "integer"
},
{
"Axiom": "IR64 averagePlantHeight 95",
"Domain": "Variety",
"Range": "integer" } ] }
```

FewShot_Prompt02 - Experiment 01 & 02

This ontology provides a structured framework for representing knowledge related to crops, focusing on growing problems, control methods, and other agricultural concepts.

Classes

The ontology includes the following key classes:

1. **Crop**: Represents crops and their varieties.
2. **GrowingProblem**: Denotes issues that crops may encounter (e.g., pests, diseases).
3. **GrowingProblemEvent**: Captures specific instances of growing problems.
4. **ControlMethod**: Methods used to manage or mitigate growing problems.
5. **ControlMethodEvent**: Instances of applying control methods in agricultural practices.
6. **CausalAgent**: Entities responsible for causing growing problems (e.g., pathogens).
7. **Symptom**: Observable effects of growing problems on crops.

8. **PreventionMethod**: Strategies aimed at preventing growing problems.
9. **ApplicationMethod**: Methods for applying control measures (e.g., spraying).
10. **Location**: Places where control methods or problems occur.
11. **TimeOfApplication**: Timeframe for applying a control method.
12. **Quantity**: Quantity of resources used in control methods.
13. **Unit**: Units of measurement for quantities.

Object Properties

The ontology defines relationships to connect the concepts:

1. affect

- **Description: Indicates that a GrowingProblem has an effect on a Crop.**
- **Domain: GrowingProblem**
- **Range: Crop**

2. applyInControlMethod

- **Description: Links a ControlMethodEvent to the FarmingStage where it is applied.**
- **Domain: ControlMethodEvent**
- **Range: FarmingStage**

3. cause

- **Description: Indicates a causal relationship between a Cause and a GrowingProblemEvent.**
- **Domain: Cause**
- **Range: GrowingProblemEvent**

4. hasApplicationMethodForCME

- **Description: Links a ControlMethodEvent to its associated ApplicationMethod.**
- **Domain: ControlMethodEvent**
- **Range: ApplicationMethod**

5. hasCausalAgent

- **Description: Links a GrowingProblemEvent to its CausalAgent.**
- **Domain: GrowingProblemEvent**
- **Range: CausalAgent**

6. hasControlMethod

- **Description: Associates a GrowingProblemEvent with a specific ControlMethod.**
- **Domain: GrowingProblemEvent**
- **Range: ControlMethod**

7. hasControlMethodEvent

- **Description: Links a GrowingProblemEvent to a ControlMethodEvent.**
- **Domain: GrowingProblemEvent**
- **Range: ControlMethodEvent**

8. hasGrowingProblem

- **Description: Links a GrowingProblemEvent to a GrowingProblem.**
- **Domain: GrowingProblemEvent**
- **Range: GrowingProblem**

9. hasGrowingProblemEvent

- **Description: Links a Crop to a GrowingProblemEvent.**
- **Domain: Crop**
- **Range: GrowingProblemEvent**

10. hasLocationForCME

- **Description: Associates a ControlMethodEvent with a Location.**
- **Domain: ControlMethodEvent**
- **Range: Location**

11. hasPesticideOf

- **Description: Relates a Pesticide to a ControlMethodEvent.**
- **Domain: Pesticide**
- **Range: ControlMethodEvent**

12. hasPreventionMethod

- **Description: Links a GrowingProblemEvent to a PreventionMethod.**
- **Domain: GrowingProblemEvent**
- **Range: PreventionMethod**

13. hasQuantityForControlMethodEvent

- **Description: Links a ControlMethodEvent to a Quantity specifying the amount used.**
- **Domain: ControlMethodEvent**
- **Range: Quantity**

14. hasSoilTypeForVariety

- **Description: Associates a Variety with its preferred SoilType.**
- **Domain: Variety**
- **Range: SoilType**

15. hasSymptom

- **Description: Links a GrowingProblemEvent to its associated Symptom.**
- **Domain: GrowingProblemEvent**
- **Range: Symptom**

16. hasTimeOfApplicationForCME

- **Description: Links a ControlMethodEvent to the TimeOfApplication for its execution.**
- **Domain: ControlMethodEvent**
- **Range: TimeOfApplication**

17. hasUnitsForControlMethodEvent

- **Description: Links a ControlMethodEvent to the Unit of measurement for the applied quantity.**
- **Domain: ControlMethodEvent**
- **Range: Unit**

Functional Properties

Some object properties, such as **hasCausalAgent**, **hasGrowingProblem**, and **hasQuantityForControlMethodEvent**, are defined as functional, meaning each instance can only have one associated value for these properties.

Identify the **Assertional Axioms** for the above ONTOLOGY in the following JSON format. Populate the JSON format extracting the relevant data from the given PDF description, ensuring accurate domain and range assignments. Below is an example with dummy data for guidance. Use consistent naming and formatting for all elements.

Output JSON:

```
{
 "ClassAssertional": [
  {
   "Class": "GrowingProblem",
   "Individuals": ["Thrips", "BacterialWilt"]
  },
  {
   "Class": "SoilType",
   "Individuals": ["ClaySoil", "SandyLoam"]
  }
 ],
 "ObjectPropertyAssertional": [
  {
   "Axiom": "BasmatiRice hasSoilTypeforVariety ClaySoil",
   "Domain": "Variety",
   "Range": "SoilType"
  },
  {
   "Axiom": "IR64 hasSoilTypeforVariety SandyLoam",
   "Domain": "Variety",
   "Range": "SoilType"
  }
 ],
 "DataPropertyAssertional": [
  {
   "Axiom": "BasmatiRice averagePlantHeight 110",
   "Domain": "Variety",
   "Range": "integer"
  },
  {
   "Axiom": "IR64 averagePlantHeight 95",
```

```
    "Domain": "Variety",
    "Range": "integer"
   }
  ]
}
```

Pattern-based Prompt examples - Experiment 03

Hierarchical Patterns - Template

```
prompt = (
        "You will be provided with three components: USER_JSON, TEXT_CONTENT, and OUTPUT_JSON_FORMAT. "
        "1. **USER_JSON**: This contains a hierarchical pattern; it represents subclasses and their associated data properties without values. "
        "2. **TEXT_CONTENT**: This contains the text content from which you need to extract information. "
        "3. **OUTPUT_JSON_FORMAT**: This specifies the format in which you should return your response. "

        "Your task is to read the USER_JSON and identify the individuals from the TEXT_CONTENT for the given classes and subclasses.  "
        "For each individual, extract the corresponding values for the data properties. "
        "Return the results formatted as specified in OUTPUT_JSON_FORMAT. "

        "Do not include any additional messages or content in your response."

        f"\n\nUSER_JSON: {hierarchical_classes }"
        f"\nOUTPUT_JSON_FORMAT: {output_format}"
        f"\nTEXT_CONTENT: {docdata}"
```

Examples for the variables:

```
Hierarchical classes = {
  "classes": {
    "Crop": {
      "subclasses": {
        "variety":{
          "data_properties": [
            "average_yield",
            "average_yield_unit",
            "features",
            "average plant height",
            "average plant height units",
            "maturity period",
            "maturity period units"
          ]
        }
          "rootCrop":{
          "data_properties": [
            "scientific_name",
            "color"
          ]
        }

      }
    }
  }
}
```

```
output_format = '''{
"ParentClass": {
  "SubClass": [
  {
    "individual_name": "value",
    "dataproperty_01": "value",
    "dataproperty_02": "value"
  },
  {
    "individual_name": "value",
    "dataproperty_01": "value",
    "dataproperty_02": "value"
    "dataproperty_03": "value"
  }
  ]
}
}'''
```

Binary Patterns

```
prompt = (
"You will be provided with three components: USER_JSON, TEXT_CONTENT, and OUTPUT_JSON_FORMAT. "
"1. **USER_JSON**: This contains a binary pattern; it represents OBJECT_PROPERTY relations of the ontology with their DOMAINS and RANGES."
"2. **TEXT_CONTENT**: This contains the text content from which you need to extract individuals for ontology. "
"3. **OUTPUT_JSON_FORMAT**: This specifies the format in which you should return your response. "

"Your task is to read the USER_JSON and identify the object properties with their domains and ranges. Then, identify the individuals those follows the given object properties from the TEXT_CONTENT."
"Return the results formatted as specified in OUTPUT_JSON_FORMAT. "

"Do not include any additional messages or content in your response."

f"\n\nUSER_JSON: {binary_relations}"
f"\nOUTPUT_JSON_FORMAT: {jsonExample}"
f"\nTEXT_CONTENT: {docdata}"
)
```

Examples for the variables:

```
binary_relations = '''
{
"OntologyAxiom": [
  {
   "Binary_Relation": "Crop IsaffectedBy GrowingProblem"
```

```
jsonExample= '''
{
"ObjectPropertyAxiom": [
  {
  "INDIVIDUAL": ["individual_1", "individual_2"],
  "OBJECT_PROPERTY": "relationship",
  "DOMAIN": "domain_of_the_relationship",
```

```
    "DOMAIN": "Crop",
    "RANGE": "GrowingProblem",
    "OBJECT_PROPERTY": "IsaffectedBy"
    "Annotation": "Crop can be a rice considered in the doc. Growing Problme can be a pest or disease"
    }
  ]
  }
  '''
```

```
    "RANGE": "range_of_the_relationship",
    "AXIOM": "individual_1 relationship individual_2",
    },
    {
    "INDIVIDUAL": ["individual_1", "individual_3"],
    "OBJECT_PROPERTY": "relationship",
    "DOMAIN": "class_type_of_indididual_1",
    "RANGE": "class_type_of_indididual_3",
    "AXIOM": "individual_1 relationship individual_3",
    }
  ]
  }
  '''
```

Nary Patterns

```
  prompt = (
  "You will be provided with three components: USER_JSON, TEXT_CONTENT, and OUTPUT_JSON_FORMAT. "
  "1. **USER_JSON**: This is the JSON defining my ontology's structure, representing classes with n-ary relationships and their connected range classes."
  "2. **TEXT_CONTENT**: This contains the text content from which you need to extract individuals for ontology following the relationships. "
  "3. **OUTPUT_JSON_FORMAT**: This specifies the format in which you should return your response. "

  "Your task: Based on the n-ary class and their associated range classes in USER_JSON, generate individuals that adhere to the given relationships from the TEXT_CONTENT"
  "For the n-ary class, assign a primary key by shortening the class name and appending a number"
  "Then, connect each primary key instance with depending individuals of its range classes using their relationships. "
  "Finally, provide the response following the OUTPUT_JSON_FORMAT"
  "Do not include any additional messages or content in your response. List all possible patterns the defined context"

  f"\n\nUSER_JSON: {n_ary_entity }"
  f"\nOUTPUT_JSON_FORMAT: {jsonExample}"
  f"\nTEXT_CONTENT: {docdata}"
  )
```

```
  n_ary_entity = '''
  {
  "NaryPropertyAxiom": {
    "ontology structure": [
      "Crop hasGrowingProblemEvent GrowingProblemEvent"
      "GrowingProblemEvent hasGrowingProblem GrowingProblem",
      "GrowingProblemEvent hasSymptom Symptom",
      "GrowingProblemEvent hasCausalAgent CausalAgent",
      "GrowingProblemEvent hasVailablePeriod Season",
      "GrowingProblemEvent hasControlMethod ControlMethod",
      "GrowingProblemEvent hasPreventionMethod PreventionMethod"
    ]
  }
  }
  '''
```

```
  jsonExample= '''
  {
  "NaryPropertyAxiom": {
    "pattern01": {
      "Axioms":  [
          "<intance01_name,"> relationship<event_id>"
          "<event_id> depend_relationship<intance02_name,">",
          "<event_id> depend_relationship<intance03_name,">",
          "<event_id> depend_relationship<intance04_name,">",
        ],
      "Individuals": [
      "intance01_name","intance02_name","","intance03_name","intance04_name",""]
    }
    "pattern02": {
      "Axioms":  [
          "<intance05_name,"> depend_relationship<event_id>"
          "<event_id> depend_relationship<intance06_name,">",
          "<event_id> depend_relationship<intance07_name,">",
          "<event_id> depend_relationship<intance08_name,">",
        ],
      "Individuals": [
      "intance05_name","intance06_name","","intance07_name","intance08_name",""]
    }
  }
  }
  '''
```

Class Attributes Pattern

```
    prompt = (
      "You will be provided with three components: USER_JSON, TEXT_CONTENT, and OUTPUT_JSON_FORMAT. "
      "1. **USER_JSON**: This contains an attribute pattern. It contains classes and their associated data properties without values. "
      "2. **TEXT_CONTENT**: This contains the text content from which you need to extract information. "
      "3. **OUTPUT_JSON_FORMAT**: This specifies the format in which you should return your response. "

      "Your task is to read the USER_JSON and identify the individuals and their corresponding data properties."
      "Then, extract values for data properties of each individual from the TEXT_CONTENT."
      "Then, return the results formatted as specified in OUTPUT_JSON_FORMAT. "
      "Do not include any additional messages or content in your response."

      f"\n\nUSER_JSON: {classList }"
      f"\nOUTPUT_JSON_FORMAT: {output_format}"
      f"\nTEXT_CONTENT: {docdata}"
    )
```

```
classList = {
  "classes": {
    "SoilType": {
```

```
  output_format = '''{
"Individuals": {
  "individual_name_01": {
```

```
            "data_properties": [
                "hasmaxphvalue",
                "hasminphvalue"
            ]
        },
        "Crop": {
            "data_properties": [
                "scientific_name"
            ]
        }

    }
}
```

```
        "class": "add the corresponding class_name given in the USER_JSON",
        "dataproperty_01": "value",
        "dataproperty_02": "value"
    },
    "individual_name_02": {
        "class": ""add the corresponding class_name given in the USER_JSON"",
        "dataproperty_01": "value",
        "dataproperty_02": "value"
    }
  }
}
'''
```