Task 1:Logical Statements to Prolog Rules Conversion Documentation

Description

This documentation explains the conversion of a set of logical statements into Prolog rules. The task is to convert the given logical statements into appropriate Prolog rules based on the provided facts. The goal is to use the inference engine of Prolog to apply the rules and reach conclusions given the facts.

Logical Statements

The following logical statements are provided:

1. If X is a mammal, then X has fur.
2. If X has fur and X has a tail, then X is a rat.
3. If X has claws and X has fur, then X is a cat.
4. If X is a cat, then X meows.
5. If X has feathers, then X is a bird.
6. If X is a bestfriend and X has fur, then X is a dog.
7. If X is a dog and Y meows, then X likes Y.
8. If X is a cat and Y is a bird, then X likes Y.
9. If X is a cat and Y is a rat, then X likes Y.

Prolog Rules

The logical statements can be converted into the following Prolog rules:

```
has_fur(X) :- mammal(X).

rat(X) :- has_fur(X), tail(X).

cat(X) :- claws(X), has_fur(X).

meows(X) :- cat(X).

bird(X) :- feathers(X).

dog(X) :- bestfriend(X), has_fur(X).

likes(X, Y) :- dog(X), meows(Y).
```

likes(X, Y) :- cat(X), bird(Y).

likes(X, Y) :- cat(X), rat(Y).

These Prolog rules define the relationships and conditions stated in the logical statements. By applying these rules to the provided facts, the Prolog inference engine can derive additional information and make logical conclusions.

Usage

To utilize the converted Prolog rules, load them into a Prolog interpreter and use the defined predicates and rules to run queries and make inferences. You can use the provided facts or add new facts to the knowledge base as needed.

Examples

Example 1: Querying if an animal has fur

```
?- has_fur(kitty).

true.
```

This query checks if `kitty` has fur. The output is `true`, indicating that `kitty` indeed has fur based on the fact `mammal(kitty)`.

Example 2: Querying if an animal is a rat

prolog

```
?- rat(ratatat).

true.
```

This query checks if `ratatat` is a rat. The output is `true`, indicating that `ratatat` satisfies the conditions of having fur and a tail, based on the facts `tail(ratatat)` and `has_fur(ratatat)`.

Example 3: Querying if an animal is a cat

prolog

```
?- cat(kitty).

true.
```

This query checks if `kitty` is a cat. The output is `true`, indicating that `kitty` satisfies the conditions of having claws and fur, based on the facts `claws(kitty)` and `has_fur(kitty)`.

Example 4: Querying if an animal meows

prolog

```
?- meows(kitty).

true.
```

This query checks if `kitty` meows. The output is `true`, indicating that `kitty` is a cat and satisfies the condition of the rule `meows(X) :- cat(X)`.

Example 5: Querying if an animal is a bird

prolog

```
?- bird(tweety).

true.
```

This query checks if `tweety` is a bird. The output is `true`, indicating that `tweety` satisfies the condition of having feathers, based on the fact `feathers(tweety)`.

The conversion of the logical statements into Prolog rules enables the use of Prolog's inference engine to derive conclusions based on the given facts. By loading the rules into a Prolog interpreter, one can run queries to obtain information and make logical inferences using the defined rules and facts.

Code:

```
has_fur(X) :- mammal(X).
rat(X) :- has_fur(X), tail(X).
cat(X) :- claws(X), has_fur(X).
meows(X) :- cat(X).
bird(X) :- feathers(X).
dog(X) :- bestfriend(X), has_fur(X).
likes(X, Y) :- dog(X), meows(Y).
likes(X, Y) :- cat(X), bird(Y).
likes(X, Y) :- cat(X), rat(Y).
```

Task 2:Family Tree Program Documentation

Description

This Prolog program represents a family tree and provides predicates and rules to define various relationships within the family. The program uses a set of facts and rules to represent parent-child relationships, gender, and familial connections such as father, mother, son, daughter, grandparent, grandchild, spouse, sibling, aunt, uncle, nephew, niece, and cousin.

Usage

To use the family tree program, load it into a Prolog interpreter and execute queries to test the relationships and connections within the family tree. You can modify the existing facts or add new facts to customize the family tree according to your requirements.

Predicates and Rules

Relationships

- `parent(Parent, Child)`: Represents a parent-child relationship between `Parent` and `Child`.
- `male(Person)`: Indicates that `Person` is male.
- `female(Person)`: Indicates that `Person` is female.

Rules

- `father(Father, Child)`: Defines a father-child relationship where `Father` is the father of `Child`.
- `mother(Mother, Child)`: Defines a mother-child relationship where `Mother` is the mother of `Child`.
- `child(Child, Parent)`: Specifies that `Child` is a child of `Parent`.
- `son(Child, Parent)`: Specifies that `Child` is a son of `Parent`.
- `daughter(Child, Parent)`: Specifies that `Child` is a daughter of `Parent`.
- `grandparent(GP, GC)`: Establishes a grandparent-grandchild relationship where `GP` is the grandparent of `GC`.
- `grandmother(GM, GC)`: Specifies that `GM` is the grandmother of `GC`.
- `grandfather(GF, GC)`: Specifies that `GF` is the grandfather of `GC`.
- `grandchild(GC, GP)`: Specifies that `GC` is a grandchild of `GP`.
- `grandson(GS, GP)`: Specifies that `GS` is a grandson of `GP`.
- `granddaughter(GD, GP)`: Specifies that `GD` is a granddaughter of `GP`.
- `spouse(Husband, Wife)`: Defines a spouse relationship between `Husband` and `Wife`.
- `husband(Person, Wife)`: Specifies that `Person` is the husband of `Wife`.
- `wife(Person, Husband)`: Specifies that `Person` is the wife of `Husband`.
- `sibling(Person1, Person2)`: Specifies that `Person1` and `Person2` are siblings.
- `brother(Person, Sibling)`: Specifies that `Person` is a brother of `Sibling`.
- `sister(Person, Sibling)`: Specifies that `Person` is a sister of `Sibling`.
- `aunt(Aunt, Person)`: Defines an aunt-niece/nephew relationship where `Aunt` is the aunt of `Person`.
- `uncle(Uncle, Person)`: Defines an uncle-niece/nephew relationship where `Uncle` is the uncle of `Person`.
- `nephew(Nephew, Person)`: Specifies that `Nephew` is a nephew of `Person`.
- `niece(Niece, Person)`: Specifies that `Niece` is a niece of `Person`.
- `cousin(Person, Cousin)`: Specifies that `Person` and `Cousin` are first cousins.

Note: The program can be extended to include rules for nth cousins, removed cousins, and greatn-grandparents as per specific requirements.

Examples

Example 1: Checking Father-Child Relationship

```
?- father(john, sarah).
true.
```

This query checks if `john` is the father of `sarah`. The output is `true`, indicating that `john` is indeed the father of `sarah`.

Example 2: Checking Daughter-Mother Relationship

```
?- daughter(sarah, mary).
true.
```

This query checks if `sarah` is the daughter of `mary`. The output is `true`, indicating that `sarah` is indeed the daughter of `mary`.

Example 3: Finding Grandparents

```
?- grandparent(GP, emily).
GP = john ;
GP = mary.
```

his query finds all the grandparents of `emily`. The output shows `john` and `mary` as the grandparents of `emily`.

Example 4: Checking Spouse Relationship

```
?- spouse(john, mary).
true.
```

This query checks if `john` and `mary` are spouses. The output is `true`, indicating that `john` and `mary` are indeed spouses.

Example 5: Checking Nephew Relationship

```
?- nephew(alex, sarah).
true.
```

This query checks if `alex` is the nephew of `sarah`. The output is `true`, indicating that `alex` is indeed the nephew of `sarah`.

Example 6: Finding Cousins

```
?- cousin(michael, emily).
true.
```

This query finds if `michael` and `emily` are first cousins. The output is `true`, indicating that `michael` and `emily` are indeed first cousins.

The family tree Prolog program provides a flexible framework to define and explore various familial relationships. By using the provided predicates and rules, users can query and analyze relationships between individuals in the family tree. The program can be expanded and customized to include additional relationships and connections as needed.

Code:

```
% Relationships
parent(john, sarah).
parent(john, michael).
parent(mary, sarah).
parent(mary, michael).
parent(sarah, emily).
parent(michael, alex).
parent(sarah, liam).
parent(michael, sophia).

% Gender
male(john).
male(michael).
male(alex).
male(liam).
female(sarah).
female(mary).
female(emily).
female(sophia).

% Rules
father(Father, Child) :- parent(Father, Child), male(Father).
mother(Mother, Child) :- parent(Mother, Child), female(Mother).
child(Child, Parent) :- parent(Parent, Child).
son(Child, Parent) :- child(Child, Parent), male(Child).
daughter(Child, Parent) :- child(Child, Parent), female(Child).

grandparent(GP, GC) :- parent(GP, Parent), parent(Parent, GC).
grandmother(GM, GC) :- grandparent(GM, GC), female(GM).
grandfather(GF, GC) :- grandparent(GF, GC), male(GF).
grandchild(GC, GP) :- grandparent(GP, GC).
grandson(GS, GP) :- grandchild(GS, GP), male(GS).
granddaughter(GD, GP) :- grandchild(GD, GP), female(GD).

spouse(Husband, Wife) :- parent(Husband, Child), parent(Wife, Child), Husband \= Wife.
husband(Person, Wife) :- spouse(Person, Wife), male(Person).
wife(Person, Husband) :- spouse(Husband, Person), female(Person).
```

```
sibling(Person1, Person2) :- parent(Parent, Person1), parent(Parent, Person2), Person1 \= Person2.
brother(Person, Sibling) :- sibling(Person, Sibling), male(Person).
sister(Person, Sibling) :- sibling(Person, Sibling), female(Person).

% Aunt, Uncle, Nephew, Niece, and First Cousin
aunt(Aunt, Person) :- sister(Aunt, Parent), parent(Parent, Person).
uncle(Uncle, Person) :- brother(Uncle, Parent), parent(Parent, Person).
nephew(Nephew, Person) :- male(Nephew), (uncle(Person, Nephew) ; aunt(Person, Nephew)).
niece(Niece, Person) :- female(Niece), (uncle(Person, Niece) ; aunt(Person, Niece)).
cousin(Person, Cousin) :- parent(Parent1, Person), parent(Parent2, Cousin), sibling(Parent1, Parent2).

% nth cousin, removed cousins, and greatn-grandparents can be added as needed.
```