From Zero to Hero: Prover9/Mace4 Problems in FOL

Andris Denis, Danci Patricia

Group: 30434

Contents

1 A2: Logics

Prover9 and Mace4 are powerful tools for automated reasoning and model building in first-order logic, widely used in mathematical research, artificial intelligence, and formal verification. These tools enable users to explore logical problems systematically, proving theorems and finding counterexamples to conjectures.

Prover9, a successor to the Otter theorem prover, is designed to assist with proving theorems by refutation. It uses resolution and paramodulation to check the validity of logical formulas, making it an indispensable tool for researchers working in fields like algebra, geometry, and software verification. On the other hand, Mace4 complements Prover9 by constructing finite models for sets of first-order statements. It excels at finding counterexamples or verifying the consistency of logical systems.

This documentation aims to provide a comprehensive guide to using Prover9 and Mace4 for solving logical problems, starting from basic setup and syntax to advanced techniques for tackling complex scenarios.

1.1 Two Cops, a Thief

We will start with an easy problem, in order to get familiar with Mace4 syntax.

Three people met at a corner of a street. They are all dressed like cops, so they don't know who among them is the thief (the bad guy). The following conditions are given:

- The cops will always tell the truth (because they are good).
- The thief will also tell the truth to make himself appear like a good cop.

The three individuals are labeled as **A**, **B**, and **C**. Each of them makes the following statements:

- **A**: *"C's not the thief."*
- **B**: *"One of you both is the thief!"*
- **C**: *"I'm not the thief."*

Using this information, determine: **Who is the thief?**

We first know that all statements are true. A's the thief. He says C's not the thief, and B says one of the other two is, which means he must not be. So the only one left is A.

```
formulas(assumptions).
p1 & p2 & p3.  % There are three people: p1 (A), p2 (B), and p3 (C), all present.
t1 | t2 | t3.  % At least one person is the thief: t1 (A is thief), t2 (B is thief), or t3 (C is thief).

% only one thief
t1 -> -t2.     % If A is the thief (t1), B cannot be the thief (-t2).
t1 -> -t3.     % If A is the thief (t1), C cannot be the thief (-t3).
t2 -> -t1.     % If B is the thief (t2), A cannot be the thief (-t1).
t2 -> -t3.     % If B is the thief (t2), C cannot be the thief (-t3).
t3 -> -t1.     % If C is the thief (t3), A cannot be the thief (-t1).
t3 -> -t2.     % If C is the thief (t3), B cannot be the thief (-t2).

% adding the cops
c1 & c2 | c2 & c3 | c1 & c3.  % At least two of the three people are cops: c1 (A is a cop), c2 (B is a cop), c3 (C is a cop).

% assign job to people
p1 -> (c1 | t1).  % A (p1) is either a cop (c1) or the thief (t1).
p2 -> (c2 | t2).  % B (p2) is either a cop (c2) or the thief (t2).
p3 -> (c3 | t3).  % C (p3) is either a cop (c3) or the thief (t3).

% adding exclusions
t1 -> (c2 & c3).  % If A is the thief (t1), then B and C must both be cops (c2 and c3).
t2 -> (c1 & c3).  % If B is the thief (t2), then A and C must both be cops (c1 and c3).
t3 -> (c1 & c2).  % If C is the thief (t3), then A and B must both be cops (c1 and c2).
t1 -> -c1.        % If A is the thief (t1), A cannot also be a cop (-c1).
t2 -> -c2.        % If B is the thief (t2), B cannot also be a cop (-c2).
t3 -> -c3.        % If C is the thief (t3), C cannot also be a cop (-c3).

% what they say
p1 -> -t3.        % A (p1) says C is not the thief (-t3).
```

```
p2 -> (t1 | t3).  % B (p2) says either A (t1) or C (t3) is
    the thief.
p3 -> -t3.        % C (p3) says he is not the thief (-t3).

end_of_list.

formulas(goals).
end_of_list.
```

Listing 1: Two Cops and a Thief: Mace4 Code

1.2 Four Siblings

Sam, Alex, Charlie, and Jordan are siblings. One of them is the opposite gender from the other three. The following facts are given:

1. Charlie's only son is either Sam or Alex.
2. Jordan's sister is either Alex or Charlie.
3. Jordan is either Sam's brother or Sam's only daughter.

Question: Who is the opposite gender from the other three? Charlie is the opposite gender.

If Jordan is Sam's only daughter, then Jordan cannot have a sister. Therefore, Jordan must be Sam's brother and a male.

If Jordan's sister is Alex, then fact 1 tells us that Sam is Charlie's only son. But since Jordan is Sam's brother, Sam cannot be Charlie's only son. Therefore, Charlie must be Jordan's sister.

Since Sam is Jordan and Charlie's sibling, Alex must be Charlie's son. Now we know that Alex and Jordan are male, while Charlie is female.

```
formulas(assumptions).

% Gender constraints
girl(x) -> -boy(x).

% Exactly three are of one gender, one is opposite
((girl(a) & girl(s) & girl(c) & boy(j))
| (girl(a) & girl(j) & girl(c) &  boy(s))
| (girl(s) & girl(j) & girl(c) &  boy(a))
| (girl(s) & girl(j) & girl(a) & boy(c))
| boy(a) & boy(s) & boy(c) & girl(j)
| boy(a) & boy(j) & boy(c) & girl(s)
| boy(s) & boy(j) & boy(c) & girl(a)
| boy(s) & boy(j) & boy(a) & girl(c)).

%y being x's son means y is a boy
son(x,y) -> boy(y).
```

```
%Charli's son is either Sam or Alex -> Sam or Alex is a boy
son(c,s) | son(c, a).

%y being x's sister means y is a girl
sister(x,y) -> girl(y).

%Jordan's sister is either Alex or Charlie
sister(j,a) | sister(j,c).

%y being x's brother means y is a boy
brother(x,y) -> boy(y).

%y being x's daughter means y is a girl
daughter(x,y) -> girl(y).

%Jordan is either Sam's brother or Sam's only daughter
%if Jordan is Sam's ONLY daughter, Jordan can't have sisters or brothers
brother(s,j) | (dauther(s,j) & -sister(j,c) & -sister(j,a) & -brother(j,c) & -brother(j,a)).

end_of_list.

formulas(goals).
end_of_list.
```

Listing 2: Four Siblings: Mace4 Code

1.3 Schubert's Steamroller

Wolves, foxes, birds, caterpillars, and snails are animals, and there are some of each. Also there are some grains, and grains are plants.

Every animal either likes to eat all plants, or, all animals much smaller than itself that like to eat some plants.

Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which in turn are much smaller than wolves. Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails. Caterpillars and snails like to eat some plants.

Therefore there is an animal that likes to eat a grain-eating animal. What is it? The fox likes to eat the bird which likes to eat the grain.

Reason: The bird is a plant eating animal because it does not like to eat snails, which like to eat some plant, so it does not like to eat ALL such animals.

Therefore it must like to eat all plants.

The fox is not a plant eating animal, because the wolf does not like to eat all plants (specifically grain), and therefore must like eating all animals which like to eat some plants.

But it does not like to eat foxes.

Therefore foxes cannot be animals which like to eat some plant.

```
formulas(assumptions).
% Define basic types
all x (wolf(x) -> animal(x)).            % All wolves are animals
all x (fox(x) -> animal(x)).             % All foxes are animals
all x (bird(x) -> animal(x)).            % All birds are animals
all x (caterpillar(x) -> animal(x)).     % All caterpillars are animals
all x (snail(x) -> animal(x)).           % All snails are animals
all x (grain(x) -> plant(x)).            % All grains are plants

% Existence assertions
exists x wolf(x).                        % There exists at least one wolf
exists x fox(x).                         % There exists at least one fox
exists x bird(x).                        % There exists at least one bird
exists x caterpillar(x).                 % There exists at least one caterpillar
exists x snail(x).                       % There exists at least one snail
exists x grain(x).                       % There exists at least one grain

% Eating rules for animals
all x (animal(x) -> (all y (plant(y) -> eats(x, y))) |
          (all z (animal(z) & smaller(z, x) &
            (exists u (plant(u) & eats(z, u))) ->
            eats(x, z))).                % Either all plants are eaten by the animal, or smaller animals that eat plants are eaten by the animal

% Size ordering between caterpillars and birds
all x all y (caterpillar(x) & bird(y) -> smaller(x, y)).  % Caterpillars are smaller than birds
all x all y (snail(x) & bird(y) -> smaller(x, y)).        % Snails are smaller than birds
all x all y (bird(x) & fox(y) -> smaller(x, y)).          % Birds are smaller than foxes
all x all y (fox(x) & wolf(y) -> smaller(x, y)).          % Foxes are smaller than wolves

```

```
% Eating rules for specific pairs
all x all y (bird(x) & caterpillar(y) -> eats(x, y)).    % Birds eat caterpillars
all x (caterpillar(x) -> (exists y (plant(y) & eats(x, y)))). % Caterpillars eat plants
all x (snail(x) -> (exists y (plant(y) & eats(x, y)))). % Snails eat plants

% Exclusion of eating for certain pairs
all x all y (wolf(x) & fox(y) -> -eats(x, y)).            % Wolves do not eat foxes
all x all y (wolf(x) & grain(y) -> -eats(x, y)).           % Wolves do not eat grains
all x all y (bird(x) & snail(y) -> -eats(x, y)).           % Birds do not eat snails
end_of_list.

formulas(goals).
% Goal: Find an animal-eating relationship
exists x exists y (animal(x) & animal(y) & eats(x, y) &
                   (all z (Grain(z) -> eats(y, z)))).  % Find two animals where one eats the other and the other only eats grains
end_of_list.
```

Listing 3: Schubert's streamroller: Mace4 Code