Mathematical Logics

18 Using Prover9 and Maze4

Luciano Serafini

Fondazione Bruno Kessler, Trento, Italy

November 27, 2013

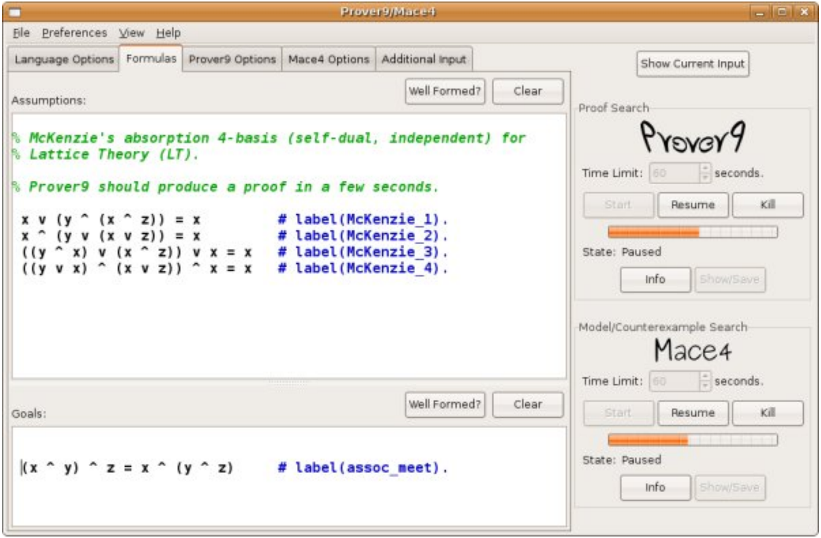
Prover9 Home Page

http://www.cs.unm.edu/ mccune/prover9/

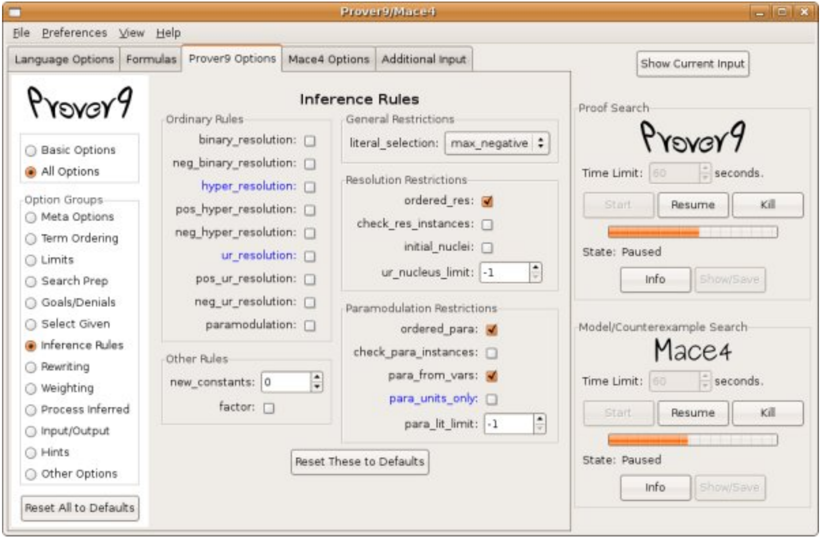
Prover9 and Mace4

- Prover9 is an automated theorem prover for first-order and equational logic,
- Mace4 searches for finite models and counterexamples

Prover9 GUI


Prover9/Mace4
File Preferences View Help
Language Options
Formulas
Prover9 Options
Mace4 Options
Additional Input
Show Current Input
Well Formed?
Clear
Assumptions:
% McKenzie's absorption 4-basis (self-dual, independent) for
% Lattice Theory (LT).
% Prover9 should produce a proof in a few seconds.
x v (y ^ (x ^ z)) = x # label(McKenzie_1).
x ^ (y v (x v z)) = x # label(McKenzie_2).
((y ^ x) v (x ^ z)) v x = x # label(McKenzie_3).
((y v x) ^ (x v z)) ^ x = x # label(McKenzie_4).
Goals:
(x ^ y) ^ z = x ^ (y ^ z) # label(assoc_meet).
Proof Search
Prover9
Time Limit: 60 seconds.
Start
Resume
Kill
State: Paused
Info
Show/Save
Model/Counterexample Search
Mace4
Time Limit: 60 seconds.
State: Paused

Prover9 GUI



Prover9/Mace4
File
Preferences
View
Help
Language Options
Formulas
Prover9 Options
Mace4 Options
Additional Input
Show Current Input
Prover9
Basic Options
All Options
Option Groups
Meta Options
Term Ordering
Limits
Search Prep
Goals/Denials
Select Given
Inference Rules
Rewriting
Weighting
Process Inferred
Input/Output
Hints
Other Options
Reset All to Defaults
Inference Rules
Ordinary Rules
binary_resolution:
neg_binary_resolution:
hyper_resolution:
pos_hyper_resolution:
neg_hyper_resolution:
ur_resolution:
pos_ur_resolution:
neg_ur_resolution:
paramodulation:
Other Rules
new_constants:
0
factor:
General Restrictions
literal_selection:
max_negative
Resolution Restrictions
ordered_res:
check_res_instances:
initial_nuclei:
ur_nucleus_limit:
-1
Paramodulation Restrictions
ordered_para:
check_para_instances:
para_from_vars:
para_units_only:
para_lit_limit:
-1
Reset These to Defaults
Proof Search
Prover9
Time Limit:
60
seconds.
Start
Resume
Kill
State: Paused
Info
Show/Save
Model/Counterexample Search
Mace4
Time Limit:
60
seconds.
Start
Resume
Kill
State: Paused
Info
Show/Save

Prover9's Proof Method

- The primary mode of inference used by Prover9 is resolution. It repeatedly makes resolution inferences with the aim of detecting inconsistency
- Prover9 will first do some preprocessing on the input file to convert it into the form it uses for inferencing.
  1. First it negates the formula given as a goal
  2. It then translates all formulae into clausal form.
  3. In some cases it will do some further pre-processing, (but you do not need to worry about this)
- Then it will compute inferences and by default write these standard output. Unless the input is very simple it will often generate a large number of inferences.
- If it detects an inconsistency it will stop and print out a proof consisting of the sequence of resolution rules that generated the inconsistency.
- It will also print out various statistics associated with the proof.

Simple example

Example (Reasoning in proposition logic)

Check if $p \wedge s, p \supset q, q \supset r \models r \vee t$ holds

Prover9 simple input file

```
formulas(assumptions).
p & s.                 % "&" symbol is for conjunction "and"
p -> q.                % "->" symbol is for implication "implies"
q -> r.
end_of_list.

formulas(goals).
r | t.                 % "|" symbol is for distunction "or"
end_of_list.
```

Output of Prover9

```
============================== prooftrans ============================
Prover9 (32) version Dec-2007, Dec 2007.
Process 71916 was started by luciano on coccobill.local,
Fri Nov 22 11:36:46 2013
The command was "/Users/luciano/Applications/Prover9-Mace4-v05B.app/Contents/Resources/bin-mac-intel/prove
============================== end of head ===========================

============================== end of input ==========================

============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.00 (+ 0.00) seconds.
% Length of proof is 11.
% Level of proof is 3.
% Maximum clause weight is 2.
% Given clauses 5.

1 p & s # label(non_clause).  [assumption].
2 p -> q # label(non_clause).  [assumption].
3 q -> r # label(non_clause).  [assumption].
4 r | t # label(non_clause) # label(goal).  [goal].
5 p.  [clausify(1)].
7 -p | q.  [clausify(2)].
8 -q | r.  [clausify(3)].
9 -r.  [deny(4)].
11 q.  [ur(7,a,5,a)].
12 -q.  [resolve(9,a,8,b)].
13 $F.  [resolve(12,a,11,a)].

============================== end of proof ==========================
```

A slightly more complex example using quantifiers

Example (Transitivity of subset relation)

Show that the containment relation between sets is transitive. I.e., For any set A, B, and C

$$A \subseteq B \wedge B \subseteq C \rightarrow A \subseteq C$$

Where $A \subseteq B$ is defined as $\forall x(x \in A \rightarrow x \in B)$

Prover9 input file

```
formulas(assumptions).
all x all y (subset(x,y) <-> (all z (member(z,x) -> member(z,y)))).
end_of_list.

formulas(goals).
all x all y all z (subset(x,y) & subset(y,z) -> subset(x,z)).
end_of_list.
```

Output of Prover9

```
============================== prooftrans ============================
Prover9 (32) version Dec-2007, Dec 2007.
Process 71873 was started by luciano on coccobill.local,
Fri Nov 22 11:32:23 2013
The command was "/Users/luciano/Applications/Prover9-Mace4-v05B.app/Contents/Resources/bin-mac-intel/prove
============================== end of head ===========================

============================== end of input ==========================

============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.00 (+ 0.00) seconds.
% Length of proof is 14.
% Level of proof is 4.
% Maximum clause weight is 6.
% Given clauses 6.

1 (all x all y (subset(x,y) <-> (all z (member(z,x) -> member(z,y))))) # label(non_clause).  [assumption].
2 (all x all y all z (subset(x,y) & subset(y,z) -> subset(x,z))) # label(non_clause) # label(goal).  [goal
3 subset(x,y) | member(f1(x,y),x).  [clausify(1)].
4 -subset(x,y) | -member(z,x) | member(z,y).  [clausify(1)].
5 subset(x,y) | -member(f1(x,y),y).  [clausify(1)].
6 subset(c1,c2).  [deny(2)].
7 subset(c2,c3).  [deny(2)].
8 -subset(c1,c3).  [deny(2)].
11 -member(x,c1) | member(x,c2).  [resolve(6,a,4,a)].
12 -member(x,c2) | member(x,c3).  [resolve(7,a,4,a)].
13 member(f1(c1,c3),c1).  [resolve(8,a,3,a)].
14 -member(f1(c1,c3),c3).  [resolve(8,a,5,a)].
15 member(f1(c1,c3),c2).  [resolve(13,a,11,a)].
18 $F.  [ur(12,b,14,a),unit_del(a,15)].
```

An even more complex example

Example (Schubert's "Steamroller" Problem)

- Wolves, foxes, birds, caterpillars, and snails are animals, and there are some of each of them.
- 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 are in turn 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.
- Prove there is an animal that likes to eat a grain-eating animal. (where a grain eating animal is one that eats all grains)

An even more complex example

Example (Schubert's "Steamroller" Problem)

- Wolves, foxes, birds, caterpillars, and snails are animals, and there are some of each of them.

$$\forall x.(Wolf(x) \vee Fox(x) \vee Bird(x) \vee Caterpillar(x) \vee Snail(x) \supset animal(x))$$

$$\exists x.Worlf(x) \wedge \exists x.Fox(x) \wedge \exists x.Bird(x) \wedge \exists x.Caterpillar(x) \wedge \exists x.Snail(x)$$

- Also there are some grains, and grains are plants.

$$\exists x.Grain(x) \wedge \forall x.(Grain(x) \supset Plant(x))$$

An even more complex example

Example (Schubert's "Steamroller" Problem)

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

$$\begin{aligned}\forall x.(Animal(x) \supset (\forall y.(Plant(y) \supset Eats(x,y)) \vee \\ \forall z.(Animal(z) \wedge Smaller(z,x) \wedge \\ (\exists u(plant(u) \wedge eats(z,u))) \supset \\ Eats(x,z))))\end{aligned}$$

- Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which are in turn much smaller than wolves.

$$\forall x \forall y(Caterpillar(x) \wedge Bird(y) \supset Smaller(x,y))$$
$$\forall x \forall y(Snail(x) \wedge Bird(y) \supset Smaller(x,y))$$
$$\forall x \forall y(Bird(x) \wedge Fox(y) \supset Smaller(x,y))$$
$$\forall x \forall y(Fox(x) \wedge Wolf(y) \supset Smaller(x,y))$$

An even more complex example

Example (Schubert's "Steamroller" Problem)

- Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails.

$$\forall x \forall y.(Wolf(x) \wedge (Fox(y) \vee Grain(y)) \rightarrow \neg Eatis(x, y)$$
$$\forall x \forall y.(Bird(x) \wedge Caterpillar(y) \supset eats(x, y))$$
$$\forall x \forall y.(Bird(x) \wedge Snail(y) \supset \neg eats(x, y))$$

- Caterpillars and snails like to eat some plants.

$$\forall x(Caterpillar(x) \vee Snail(x) \supset \exists y(Plant(y) \wedge Eats(x, y)))$$

- Prove there is an animal that likes to eat a grain-eating animal. (where a grain eating animal is one that eats all grains)

$$\exists xy.(Animal(x) \wedge Animal(y) \wedge Eats(x, y) \wedge$$
$$(\forall z.(Grain(z) \supset Eats(y, z)))$$

Prover9 input file 1/2

```
formulas(assumptions).
all x (wolf(x) -> animal(x)).
all x (fox(x) -> animal(x)).
all x (bird(x) -> animal(x)).
all x (caterpillar(x) -> animal(x)).
all x (snail(x) -> animal(x)).
all x (grain(x) -> plant(x)).

exists x wolf(x).
exists x fox(x).
exists x bird(x).
exists x caterpillar(x).
exists x snail(x).
exists x grain(x).

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)))).
```

Prover9 input file 2/2

```
all x all y (caterpillar(x) & bird(y) -> smaller(x,y)).
all x all y (snail(x) & bird(y) -> smaller(x,y)).
all x all y (bird(x) & fox(y) -> smaller(x,y)).
all x all y (fox(x) & wolf(y) -> smaller(x,y)).
all x all y (bird(x) & caterpillar(y) -> eats(x,y)).

all x (caterpillar(x) -> (exists y (plant(y) & eats(x,y)))).
all x (snail(x) -> (exists y (plant(y) & eats(x,y)))).

all x all y (wolf(x) & fox(y) -> -eats(x,y)).
all x all y (wolf(x) & grain(y) -> -eats(x,y)).
all x all y (bird(x) & snail(y) -> -eats(x,y)).
end_of_list.

formulas(goals).
exists x exists y (animal(x) & animal(y) & eats(x,y) &
                   (all z (Grain(z) -> eats(y,z)))).
end_of_list.
```

Exercize (A Murder Mystery Problem)

Translate the following sentences into FOL

1. Someone who lives in Dreadbury Mansion killed Aunt Agatha.
2. Agatha, the butler, and Charles live in Dreadbury Mansion, and are the only people who live therein.
3. A killer always hates his victim, and is never richer than his victim.
4. Charles hates no one that Aunt Agatha hates.
5. Agatha hates everyone except the butler.
6. The butler hates everyone not richer than Aunt Agatha.
7. The butler hates everyone Aunt Agatha hates.
8. No one hates everyone.
9. Agatha is not the butler.

Now use the Prover9 to show

1. prover to deduce who killed Aunt Agatha. (Hint: try for each of the possibilities).

Exercize (A Murder Mystery Problem)

Translate the following sentences into FOL

1. Someone who lives in Dreadbury Mansion killed Aunt Agatha.
   `exists x (livesin(x,DM) & kills(x,Agatha)).`
2. Agatha, the butler, and Charles live in Dreadbury Mansion, and are the only people who live therein.
   ```
   livesin(Agatha,DM) & lives(Thebutler,DM) & lives(Charles,DM).
   all x (livesin(x,DM) <-> x=Agatha | x=Thebutler | x=Charles).
   ```
3. A killer always hates his victim, and is never richer than his victim.
   `all x all y (kills(x,y) -> hates(x,y) & -richer(x,y)).`
4. Charles hates no one that Aunt Agatha hates.
   `all x all y (hates(Agatha,y) -> -hates(Charles,y)).`
5. Agatha hates everyone except the butler.
   `all x (hates(Agatha,x) <-> -x=Thebutler & -x=Agatha).`
6. The butler hates everyone not richer than Aunt Agatha.
   `all x all y (richer(x,Agatha) -> hates(Thebutler,x)).`
7. The butler hates everyone Aunt Agatha hates.
   `all x (hates(Agatha,x) -> hates(Thebutler,x)).`
8. No one hates everyone. `all x exists y -hates(x,y).`
9. Agatha is not the butler. `-Agatha = Thebutler.`
10. who killed Aunt Agatha? `kills(Thebutler,Agatha).`

Model generation - Mace4

- Prover9 tries to show that $\Gamma \models \phi$ by making attempts to show that the set of formulas $\Gamma \cup \{\neg\phi\}$ is not satisfiable.
- If Prover9 succeeds ok in showing that $\Gamma \cup \{\neg\phi\}$ is not satisfiable, then clearly $\Gamma \models \phi$.
- But what about if Prover9 fails in showing that $\Gamma \cup \{\neg\phi\}$ is not satisfiable? i.e., when $\Gamma \cup \{\neg\phi\}$ is satisfiable?
- Can we have a model for $\Gamma \cup \{\neg\phi\}$?
- Yes, we have to use Mace4.

Mace4

- Mace4 is a program that searches for finite models of first-order formulas.
- For a given domain size, all instances of the formulas over the domain are constructed. The result is a set of ground clauses with equality.
- Then, a decision procedure based on ground equational rewriting is applied. If satisfiability is detected, one or more models are printed.

Mace4 – example

Input file:

```
arc(x,y) -> node(x) & node(y).
exists x1 exists x2 exists x3 (color(x1) & color(x2) & color(x3) &
              x1 != x2 & x2 != x3 & x1 != x3).
color(x1) & color(x2) & color(x3) & color(x4) ->
             x1=x2 | x1=x3 | x1=x4 | x2=x3 | x2=x4 | x3=x4.
hascolor(x,y) -> node(x) & color(y).
color(x) -> -node(x).
color(x) | node(x).
node(x) -> exists y hascolor(x,y).
hascolor(x,y1) & hascolor(x,y2) -> y1=y2.
N1 != N2 & N1 != N3 & N1 != N4 & N2 != N3 & N2 != N4 & N3 != N4.
arc(N1,N2).
arc(N2,N3).
arc(N3,N1).
arc(N1,N4).
arc(N2,N4).
% arc(N3,N4).
arc(x,y) -> arc(y,x)
-arc(x,x).
arc(x,y) & hascolor(x,z) -> -hascolor(y,z).
```

Mace4 – example

Produced model:

```
interpretation( 7, [number = 1,seconds = 0], [
    function(N1, [0]),                    function(c1, [4]),
    function(N2, [1]),                    function(c2, [5]),
    function(N3, [2]),                    function(c3, [6]),
    function(N4, [3]),
    function(f1(_), [4,5,6,6,0,0,0]),
    relation(color(_), [0,0,0,0,1,1,1]),
    relation(node(_), [1,1,1,1,0,0,0]),
    relation(arc(_,_), [                relation(hascolor(_,_), [
        0,1,1,1,0,0,0,                      0,0,0,0,1,0,0,
        1,0,1,1,0,0,0,                      0,0,0,0,0,1,0,
        1,1,0,0,0,0,0,                      0,0,0,0,0,0,1,
        1,1,0,0,0,0,0,                      0,0,0,0,0,0,1,
        0,0,0,0,0,0,0,                      0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,                      0,0,0,0,0,0,0,
        0,0,0,0,0,0,0]),                    0,0,0,0,0,0,0])]).
```