Automated Reasoning Systems

Resolution Theorem Proving: Prover9

Temur Kutsia

RISC, Johannes Kepler University of Linz, Austria
kutsia@risc.uni-linz.ac.at

May 11, 2011

Prover9

- Automated theorem prover from the Argonne group
- Successor of Otter
- Author: Bill McCune
- Theory: First-order logic with equality
- Inference rules: Based on resolution and paramodulation
- Main Applications: In abstract algebra and formal logic
- Implemented in C.
- Web page: http://www.cs.unm.edu/˜mccune/prover9/

Running Prover9

1. Prepare the input file containing the logical specification of a conjecture and the search parameters.
2. Issue a command that runs Prover9 on the input file and produces an output file.
3. Look at the output.
4. maybe run Prover9 again with different search parameters.

How to Run Prover9

From the command line:

- Run: `prover9 -f inputfile`
  or `prover9 -f inputfile > outputfile`
  or `prover9 < inputfile`
  or `prover9 < inputfile > outputfile`
- There can be several input files, e.g.,
  `prover9 -f file1 ... filen > outputfile`
- Time limit can be specified from the command line:
  `prover9 -t 10 -f inputfile > outputfile`

Using GUI.

Syntax

- Ordinary symbols: made from the characters `a-z, A-Z, 0-9, $,` and _.
- Special symbols: made from the special characters: `{+-*/\^<>=`~?@&|!#';.`
- Quoted symbols: any string enclosed in double quotes.
- Special characters can not be mixed with the others to make symbols, unless quoted symbols are formed.

Syntax

Meaning	Connective	Example
negation	`-`	`(-p)`
disjunction	`|`	`(p | q | r)`
conjunction	`&`	`(p & q & r)`
implication	`->`	`(p -> q)`
backward implic.	`<-`	`(p <- q)`
equivalence	`<->`	`(p <-> q)`
universal quant.	`all`	`(all x all y p(x,y))`
existential quant.	`exists`	`(exists x exists y p(x,y))`
true	`$T`	
false	`$F`	
equality	`=`	`(a = b)`
negated equality	`!=`	`(a != b)`

Syntax

- Symbols are assigned precedences.
- Infix, prefix, and postfix declarations are used.
- All that helps to avoid excessive use of parentheses.
- Details: http://www.cs.unm.edu/~mccune/prover9/manual/2009-11A/syntax.html#declarations
- For instance, terms involving `+`, `*` and binary `-` can be written in infix notation: `a+b, a*b, a-b.`

Syntax

- Prover9 input file: A sequence of lists and commands.
- Lists (of formulas or clauses) can be used to declare, for instance, which formulas or clauses are assumptions and which ones are goals.
- Commands can be used, for instance, to set certain options.
- The order is largely irrelevant, except for some special cases.

Input File

Example

```
assign(max_seconds, 30).

% Is Socrates mortal?

formulas(assumptions).
   all x (man(x) -> mortal(x)).
   man(socrates).
end_of_list.

formulas(goals).
   mortal(socrates).
end_of_list.
```

Input File

Example

```
assign(max_seconds, 30).

formulas(usable).
   all x (man(x) -> mortal(x)).
   man(socrates).
end_of_list.

formulas(sos).
   -mortal(socrates).
end_of_list.
```

Input File

Example

```
assign(max_seconds, 30).

formulas(sos).
   all x (man(x) -> mortal(x)).
   man(socrates).
   -mortal(socrates).
end_of_list.
```

Input File

Example

```
assign(max_seconds, 30).

% Is Socrates mortal?

clauses(assumptions).
   - man(x) | mortal(x).
   man(socrates).
end_of_list.

formulas(goals).
   mortal(socrates).
end_of_list.
```

Distinction Between Clauses and Formulas

- Formulas can have any of the logic connectives, and all variables are explicitly quantified. Formulas go into lists that start with `formulas(list_name).`
- Clauses are simple disjunctions in which all variables are implicitly universally quantified. Clauses go into lists that start with `clauses(list_name).`
- Clauses without variables are also formulas, so they can go into either kind of list. (An exception: clauses can have attributes, and formulas cannot.)
- Because variables in clauses are not explicitly quantified, a rule is needed for distinguishing variables from constants in clauses (see terms below). No such rule is needed for formulas.
- Prover9's inference rules operate on clauses. If formulas are input, Prover9 immediately translates them into clauses.

Mail Loop: Given Clause Algorithm

Operates on the `sos` and `usable` lists.
While the `sos` list is not empty:

1. Select a given clause from `sos` and move it to the `usable` list;
2. Infer new clauses using the inference rules in effect; each new clause must have the given clause as one of its parents and members of the `usable` list as its other parents;
3. Process each new clause;
4. Append new clauses that pass the retention tests to the `sos` list.

end of while loop.

Input File

Example

Barbers paradox:

- There is a town with just one male barber.
- Every man in the town keeps himself clean-shaven.
- Some shave themselves, some are shaved by the barber.
- The barber shaves all and only those men in town who do not shave themselves.
- Does the barber shave himself?

Input File

Example

Barbers paradox:

- There is a town with just one male barber.
- Every man in the town keeps himself clean-shaven.
- Some shave themselves, some are shaved by the barber.
- The barber shaves all and only those men in town who do not shave themselves.
- Does the barber shave himself?

```
formulas(goals).
   exists x (barber(x) &
            (all y (-shaves(y,y) <-> shaves(x,y)))).
end_of_list.
```

Run Prover9 on Examples

Example

$$\begin{aligned} \textit{Assumption}: &\quad \forall x \forall y\ Q(x) \Rightarrow P(y, y). \\ \textit{Assumption}: &\quad \forall x \forall y\ P(x, b) \vee Q(f(y, x)). \\ \textit{Goal}: &\quad \exists x\ P(a, x). \end{aligned}$$

Example

$$\begin{aligned} \textit{Assumption}: &\quad \forall x \forall y \forall z\ (x * y) * z = x * (y * z). \\ \textit{Assumption}: &\quad \forall x\ x * 1 = x. \\ \textit{Assumption}: &\quad \forall x\ x * x^{-1} = 1. \\ \textit{Assumption}: &\quad \forall x\ x * x = 1. \\ \textit{Goal}: &\quad \forall x \forall y\ x * y = y * x. \end{aligned}$$