Theorem Proving with Vampire for Rigorous Systems Engineering

Question Answering

Giles Reger and Martin Suda

University of Manchester and TU Wein

Introduction

The idea here is sometimes when we have an existential conjecture we don't only want to know if it is true but also why i.e. what makes it true.

This general concept is captured in question answering.

Question Answering

Given a set of axioms A and a conjecture of the form

$$\exists x_1, \ldots x_n : \phi(x_1, \ldots, x_n)$$

find a substitution σ with domain $x_1, \ldots, x_n$ such that

$$A \models \phi(x_1, \ldots, x_n)\sigma$$

i.e. an instantiation of the existentially quantified variables that makes ϕ true in A.

There may be more than one such σ and we may want to find multiple answers.

The Answer Literal Approach

Transform the conjecture

$$\exists x_1, \ldots x_n : \phi(x_1, \ldots, x_n)$$

into

$$\exists x_1, \ldots x_n : \mathsf{ans}(x_1, \ldots, x_n) \wedge \phi(x_1, \ldots, x_n)$$

For example, $\exists x : p(x)$ becomes the clause $\neg p(x) \vee \neg \mathsf{ans}(x)$

Conceptually our conjecture then becomes

$$\exists x_1, \ldots x_n : \neg \mathsf{ans}(x_1, \ldots, x_n)$$

but we don't add this

Run saturation and look for unit clauses containing an ans literal.

The Answer Literal Approach

Transform the conjecture

$$\exists x_1, \ldots x_n : \phi(x_1, \ldots, x_n)$$

into

$$\forall x_1, \ldots x_n : \mathsf{ans}(x_1, \ldots, x_n) \rightarrow \neg\phi(x_1, \ldots, x_n)$$

For example, $\exists x : p(x)$ becomes the clause $\neg p(x) \vee \neg \mathsf{ans}(x)$

Conceptually our conjecture then becomes

$$\exists x_1, \ldots x_n : \neg\mathsf{ans}(x_1, \ldots, x_n)$$

but we don't add this

Run saturation and look for unit clauses containing an ans literal.

Example

```
fof(a,axiom,prover(vampire)).
fof(a,axiom,prover(e)).
fof(a,axiom,tutorial(vampire)).
fof(a,axiom,tutorial(probabilistic)).
fof(a,conjecture,?[X]: (prover(X) & tutorial(X))).
```

Example

$$prover(vampire)$$
$$prover(E)$$
$$tutorial(vampire)$$
$$tutorial(probabilistic)$$

$$\neg tutorial(X) \vee \neg prover(X) \vee \neg \mathsf{ans}(X)$$

Example

$$prover(vampire)$$
$$prover(E)$$
$$tutorial(vampire)$$
$$tutorial(probabilistic)$$

$$\neg tutorial(X) \vee \neg prover(X) \vee \neg \mathsf{ans}(X)$$

$$\neg prover(vampire) \vee \neg \mathsf{ans}(vampire)$$
$$\neg prover(probabilistic) \vee \neg \mathsf{ans}(probabilistic)$$

Example

$prover(vampire)$
$prover(E)$
$tutorial(vampire)$
$tutorial(probabilistic)$

$\neg tutorial(X) \vee \neg prover(X) \vee \neg \mathsf{ans}(X)$

$\neg prover(vampire) \vee \neg \mathsf{ans}(vampire)$
$\neg prover(probabilistic) \vee \neg \mathsf{ans}(probabilistic)$

$\neg \mathsf{ans}(vampire)$

Running Vampire for Question Answering

Make sure you have either a conjecture or question formula with a top-level existential quantification.

Then run

```
./vampire -av off --question_answering answer_literal prob
```

note that question answering is not currently compatible with AVATAR or FMB.

Example

```
% SZS answers Tuple [[vampire]|_] for question
1. prover(vampire) [input]
3. workshop(vampire) [input]
5. ? [X0] : (tutorial(X0) & prover(X0)) [input]
6. ~? [X0] : (tutorial(X0) & prover(X0)) [negated conjectu
7. ~? [X0] : (tutorial(X0) & prover(X0) & ans0(X0)) [answe
8. ! [X0] : (~tutorial(X0) | ~prover(X0) | ~ans0(X0)) [enn
9. prover(vampire) [cnf transformation 1]
11. tutorial(vampire) [cnf transformation 3]
13. ~tutorial(X0) | ~prover(X0) | ~ans0(X0) [cnf transforma
14. ~prover(vampire) | ~ans0(vampire) [resolution 13,11]
16. ~ans0(vampire) [subsumption resolution 14,9]
17. ans0(X0) [answer literal]
18. $false [unit resulting resolution 17,16]
```

Example

```
% SZS answers Tuple [[vampire]|_] for question
1. prover(vampire) [input]
3. workshop(vampire) [input]
5. ? [X0] : (tutorial(X0) & prover(X0)) [input]
6. ~? [X0] : (tutorial(X0) & prover(X0)) [negated conjectur
7. ~? [X0] : (tutorial(X0) & prover(X0) & ans0(X0)) [answer
8. ! [X0] : (~tutorial(X0) | ~prover(X0) | ~ans0(X0)) [enni
9. prover(vampire) [cnf transformation 1]
11. tutorial(vampire) [cnf transformation 3]
13. ~tutorial(X0) | ~prover(X0) | ~ans0(X0) [cnf transforma
14. ~prover(vampire) | ~ans0(vampire) [resolution 13,11]
16. ~ans0(vampire) [subsumption resolution 14,9]
17. ans0(X0) [answer literal]
18. $false [unit resulting resolution 17,16]
```

Answer Literals in Proof Search

- To make this work we need to make sure that we never select an answer literal for inference
- Notice that using answer literals alters proof search in other ways
  - Things that were units may not be units anymore, changing things like demodulation
  - Answer literals have some weight, changes clause selection
- If we want to avoid this then we need to extract answers directly from proofs (see later)

Multiple Answers

To obtain multiple answers we don't stop when we have one answer

We just record the answer and carry on going

When we saturation we print all found answers

Example

```
fof(a,axiom,prover(vampire)).
fof(a,axiom,prover(e)).
fof(a,axiom,tutorial(vampire)).
fof(a,axiom,tutorial(probabilistic)).
fof(a,conjecture,?[X]: (prover(X)).
```

We get the answer (without proof now)

```
% SZS answers Tuple [[e],[vampire]|_] for question
```

Disjunctive Answers

The above formulation of the question answering question was not general enough in some sense. Given a set of axioms A and a conjecture of the form

$$\exists x_1, \ldots x_n : \phi(x_1, \ldots, x_n)$$

it may be sufficient/useful to find a set of substitutions Θ such that

$$A \models \bigvee_{\sigma \in \Theta} \phi(x_1, \ldots, x_n)\sigma$$

such a set of substitutions is a disjunctive answer and tells us that at least one substitution in Θ is an answer.

Example

```
fof(a,axiom,monday => tutorial(vampire)).
fof(a,axiom,friday => tutorial(veriflow)).
fof(a,conjecture,?[X]: ((friday | monday) => tutorial(X))).
```

We get the answer

```
% SZS answers Tuple [([veriflow]|[X0]|[vampire])|_]
```

Example

```
fof(a,axiom,monday => tutorial(vampire)).
fof(a,axiom,friday => tutorial(arcade)).
fof(a,axiom, monday | friday).
fof(a,conjecture,?[X]: ( tutorial(X))).
```

We get the answer

```
% SZS answers Tuple [([arcade]|[vampire])|_]
```

Theories

It would be nice to be able to ask questions involving theories.

We can, it just works.

```
tff(a,conjecture,?[X:$int]: $greater(X,0)).
```

Gives

```
% SZS answers Tuple [[1]|_]
```

```
tff(a,conjecture,?[X:$int]:
      0 = $sum($product(X,X),$uminus(4))).
```

Gives

```
% SZS answers Tuple [[-2]|_]
```

Revisiting Multiple Answers

How many answers are there to this question?

```
tff(a,conjecture,?[X: $int,Y:$int]: 5 = $sum(X,Y)).
```

Revisiting Multiple Answers

How many answers are there to this question?

```
tff(a,conjecture,?[X: $int,Y:$int]: 5 = $sum(X,Y)).
```

We need to add a counter to give a limit on the number of answers we want e.g. 3

```
% SZS answers Tuple [[X0,$sum($uminus(X0),5)],[0,5],[5,0]|_
```

Although it is not as fun as we might expect

```
tff(a,conjecture,?[X:$int]: $greater(X,0)).
```

Gives the following 10 answers

```
% SZS answers Tuple [[1],[1],[2],([1]|[0]),[1],
                                    [1],[1],[1],[1],[1]|_]
```

Using Question Answering

Consider this definition of a function and an assertion about it. Vampire proves the assertion.

```
thf(max,type,max : $int * $int > $int).
thf(max_def, axiom,
    ![X:$int,Y:$int] : (
         max(X,Y) = $ite($less(X, Y),Y,X)
)).

thf(assert,conjecture,
  ![X:$int,Y:$int] : (
     $let(m := max(X,Y),
         $greatereq(m, X) &
         $greatereq(m, Y) &
         (m = X | m = Y))
)).
```

Using Question Answering

Now let's break the definition, the proof disappears (as expected).
But for what inputs to the function does the assertion break?

```
thf(max,type,max : $int * $int > $int).
thf(max_def, axiom,
    ![X:$int,Y:$int] : (
         max(X,Y) = $ite($less(X, Y),Y,X)
)).

thf(assert,conjecture,
  ![X:$int,Y:$int] : (
     $let(m := max(X,Y),
         $greatereq(m, X) &
         $greatereq(m, Y) &
         (m = X | m = Y))
)).
```

Using Question Answering

Now let's break the definition, the proof disappears (as expected).
But for what inputs to the function does the assertion break?

```
thf(max,type,max : $int * $int > $int).
thf(max_def, axiom,
    ![X:$int,Y:$int] : (
         max(X,Y) = $ite($less(X, Y),Y,X)
)).

thf(assert,conjecture,
  ?[X:$int,Y:$int] : (
     $let(m := max(X,Y),
          ~($greatereq(m, X) &
          $greatereq(m, Y) &
          (m = X | m = Y)))
)).
```

Using Question Answering

Now let's break the definition, the proof disappears (as expected).
But for what inputs to the function does the assertion break?

```
thf(assert,conjecture,
  ?[X:$int,Y:$int] : (
     $let(m := max(X,Y),
          ~($greatereq(m, X) &
          $greatereq(m, Y) &
          (m = X | m = Y)))
)).

% SZS answers Tuple [[X0,$sum(X0,1)]|_] for max_broke
```