Revisiting Question Answering in Vampire

Giles Reger

School of Computer Science, University of Manchester, UK

The 4th Vampire Workshop

Introduction

In this talk we will

- Revisit the question answering problem
- Look at the answer literal approach
  - Multiple answers
  - Theories
  - Making it work with AVATAR
- Look at new ideas (unimplemented)
  - Analysis of unification-free proofs
  - Using finite-model-building
  - Other stuff

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,workshop(vampire)).
fof(a,axiom,workshop(arcade)).
fof(a,conjecture,?[X]: (prover(X) & workshop(X))).
```

Example

$$prover(vampire)$$
$$prover(E)$$
$$workshop(vampire)$$
$$workshop(arcade)$$

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

Example

$prover(vampire)$
$prover(E)$
$workshop(vampire)$
$workshop(arcade)$

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

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

Example

$prover(vampire)$
$prover(E)$
$workshop(vampire)$
$workshop(arcade)$

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

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

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

Example

```
% SZS answers Tuple [[vampire]|_] for question
1. prover(vampire) [input]
3. workshop(vampire) [input]
5. ? [X0] : (workshop(X0) & prover(X0)) [input]
6. ~? [X0] : (workshop(X0) & prover(X0)) [negated conjecture 5
7. ~? [X0] : (workshop(X0) & prover(X0) & ans0(X0)) [answer li
8. ! [X0] : (~workshop(X0) | ~prover(X0) | ~ans0(X0)) [ennf tr
9. prover(vampire) [cnf transformation 1]
11. workshop(vampire) [cnf transformation 3]
13. ~workshop(X0) | ~prover(X0) | ~ans0(X0) [cnf transformatio
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] : (workshop(X0) & prover(X0)) [input]
6. ~? [X0] : (workshop(X0) & prover(X0)) [negated conjecture 5
7. ~? [X0] : (workshop(X0) & prover(X0) & ans0(X0)) [answer li
8. ! [X0] : (~workshop(X0) | ~prover(X0) | ~ans0(X0)) [ennf tr
9. prover(vampire) [cnf transformation 1]
11. workshop(vampire) [cnf transformation 3]
13. ~workshop(X0) | ~prover(X0) | ~ans0(X0) [cnf transformatio
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,workshop(vampire)).
fof(a,axiom,workshop(arcade)).
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 => workshop(vampire)).
fof(a,axiom,sunday => workshop(arcade)).
fof(a,conjecture,?[X]: ((sunday | monday) => workshop(X))).
```

We get the answer

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

Example

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

Missing a Trick with Instantiation

Currently instantiation either:

- Heuristically uses constants already in the search space
- Uses Z3 to find a single solution

Neither allows us to find interesting answers

Todo: extend the Z3 approach to query for different answers

Making Answer Literal Reasoning work with AVATAR

Problem

- As soon as we split away the answer literal it becomes non-obvious when we have found an answer. In many cases it wouldn't split anyway (shared variables)
- Unclear how to handle `unsat` in the SAT solver

Simple Solution (implemented)

- Don't split any clause containing an answer literal
- Takes away a lot of the advantages of AVATAR

Todo: A better solution should exist

Analysis of unification-free proofs

Last year I introduced a new proof output for Vampire that removed unification completely (not actually in master yet)

This expanded each proof step into instantiation + unification-free step

The idea is to analyse these proofs directly to find answers

The reason that this new proof output is useful is because we only need to track instantiations

Some inference steps not covered (yet) and some things will need special treatment (e.g. AVATAR)

Analysis of unification-free proofs

For this problem

```
fof(a,axiom,p(b)).
fof(a,conjecture,?[X]:p(X)).
```

We get this proof

```
1. p(b) [input]
2. ? [X0] : p(X0) [input]
3. ~? [X0] : p(X0) [negated conjecture 2]
4. ! [X0] : ~p(X0) [ennf transformation 3]
5. p(b) (0:2:1) [cnf transformation 1]
6. ~p(X0) (0:2:1) [cnf transformation 4]
8. ~p(b) (0:2) [instantiation 6]
9. $false (1:0) [resolution 5,8]
```

Analysis of unification-free proofs

For this problem

```
fof(a,axiom,p(c) | p(d)).
fof(a,conjecture,?[X]:p(X)).
```

We get this proof

```
1. p(d) | p(c) [input]
2. ? [X0] : p(X0) [input]
3. ~? [X0] : p(X0) [negated conjecture 2]
4. ! [X0] : ~p(X0) [ennf transformation 3]
5. p(d) | p(c) (0:4:1) [cnf transformation 1]
6. ~p(X0) (0:2:1) [cnf transformation 4]
8. ~p(d) (0:2) [instantiation 6]
9. p(c) (1:2:1) [resolution 5,8]
11. ~p(c) (0:2) [instantiation 6]
12. $false (2:0) [resolution 9,11]
```

Analysis of unification-free proofs

For this problem

```
fof(a,axiom,p(f(X)) | q(X)).
fof(a,axiom,~q(a)).
fof(a,conjecture,?[X]:p(X)).
```

We get this proof

```
1. ! [X0] : (q(X0) | p(f(X0))) [input]
2. ~q(a) [input]
3. ? [X0] : p(X0) [input]
4. ~? [X0] : p(X0) [negated conjecture 3]
5. ! [X0] : ~p(X0) [ennf transformation 4]
6. p(f(X0)) | q(X0) (0:5:1) [cnf transformation 1]
10. ~p(f(X0)) (0:3) [instantiation 5]
11. q(X0) (1:2:1) [resolution 6,10]
12. q(a) (1:2) [instantiation 11]
14. $false (2:0) [resolution 12,2]
```

Using finite-model-building

Idea: build a finite model of the axioms and ask questions of this model

Already have most of the machinery as there is a `model_check` mode that evaluates formulas to true or false in a given finite model.

Extend this to search for instances matching a given formula

Of course, restricted by the finite model we get

Combine this to make a general question answering search method?

Further Areas to Explore

- Fix AVATAR issues
- Implement the proof analysis idea
- Implement the finite-model-building idea
- Detect repeated answers
- Remove less general answers
- Focus on ground answers only?
- Vampire as an Iterator