EPiC Series in Computing

Volume 53, 2018, Pages 64–74

Vampire 2017. Proceedings of the 4th Vampire Workshop

Revisiting Question Answering in Vampire

Giles Reger

University of Manchester, Manchester, UK

Abstract

Question answering is the process of taking a conjecture existentially quantified at the outermost level and providing one or more instantiations of the quantified variable(s) as a form of an answer to the implied question. For example, given the axioms `p(a)` and `f(a)=a` the question `?[X] : p(f(X))` could return the answer `X=a`. This paper reviews the question answering problem focussing on how it is tackled within the VAMPIRE theorem prover. It covers how VAMPIRE extracts single answers, multiple answers, disjunctive answers, and answers involving theories such as arithmetic. The paper finishes by considering possible future directions, such as integration with finite model finding.

1 Introduction

Theorem proving for question answering has been considered at various times in the past (see [10] for a discussion and some historic references) but has not received *a lot* of attention. In this short paper I revisit the topic of question answering from a VAMPIRE [2] perspective.

As a historical point, question answering has been implemented in VAMPIRE for some time and the 'current status' discussed in this paper is not new in general. Although, when theories were originally added to VAMPIRE question answering was not initially supported but they are now. This paper also discusses ways in which question answering could be extended to take advantage of new features in VAMPIRE

The paper is organised as follows:

- Section 2 discusses what the question answering problem is,
- Section 3 introduces the answer literal approach to question answering and gives examples of how this can be applied in different settings.
- Section 4 discusses alternative approaches to question answering in the form of analysing proofs and utilising finite model building.

2 Defining Question Answering

I assume general knowledge of first-order logic, theorem proving in the context of VAMPIRE [2], and some awareness of the TPTP language [9] and surrounding ecosystem.

What is an Answer. The question answering problem considers the setting where there is an axiomatisation A of some problem domain that we can ask questions of. A question is a formula $\phi[\mathbf{x}]$ with free variables **x** and an answer to that question is a set of values **v** for those variables such that $\phi[\mathbf{v}]$ follows from A.

Slightly more formally, given a set of axioms A and a *conjecture* of the form

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

the problem is to find a substitution σ with domain $x_1, \dots, x_n$ such that

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

i.e. an instantiation of the existentially quantified variables that makes ϕ true in A. The range of σ does not need to be ground, indeed the identity substitution would be an answer in some cases e.g. $p(x) \models p(x)\{x \mapsto x\}$. Something that is not explored later is the possibility of asking for the *most general* or *most specific* answer to a query (although this may not be unique).

More than one Answer. There may be more than one possible answer; in some cases there may be an infinite number of possible answers. The extension from the single answer to the multiple answer case is straightforward. Variants of the problem include asking for a fixed number of answers, asking for as many as can be produces in a fixed time limit, or organising proof search as an iterator i.e. asking for more answers on demand.

Disjunctive Answers. The above formulation is not general enough in some sense. Given a set of axioms A and a *conjecture* of the form

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

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

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

such a set of substitutions is a disjunctive answer and tells us that at least one substitution in Θ is an answer. The setup is the same and disjunctive answers are extracted from clauses consisting purely of answer literals e.g. $\neg\mathsf{ans}(t_1) \vee \neg\mathsf{ans}(t_2)$.

3 The Answer Literal Approach

The standard approach to performing question answering with a theorem prover is the *answer literal* approach (again see [10] for some history. The general idea here is to add an extra literal to the conjecture clause that will collect the instantiation of existentially quantified variables.

This is based on the observation that the following clause

$$\neg\phi(x_1, \dots, x_n)$$

which is the result of negating the above conjecture, is equivalent to the pair of clauses

$$\begin{gathered} \neg\mathsf{ans}(x_1, \dots, x_n) \vee \neg\phi(x_1, \dots, x_n) \\ \mathsf{ans}(x_1, \dots, x_n) \end{gathered}$$

for the fresh symbol ans. But if we exclude the second clause then the derivation of the unit clause $\neg\mathsf{ans}(t_1,\ldots,t_n)$ would embody a contradiction of $\phi(x_1,\ldots,x_n)$ whilst holding the instantiating terms needed for that contradiction. For example, the conjecture $\exists x : p(x)$ becomes the clause $\neg p(x) \vee \neg\mathsf{ans}(x)$, which would resolve with $p(a)$ to give $\neg\mathsf{ans}(a)$ i.e. the answer. An important point is that ans only exists in the extended conjecture clause(s) so to get a unit clause containing ans would require the full discharging of the conjecture.

Another way of looking at this is that we transform the conjecture

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

into

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

such that the fresh $\mathsf{ans}(x_1,\ldots,x_n)$ is a witness for ϕ. Then when this is negated it becomes

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

which is equivalent to

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

which means that if we refute the conjecture we produce the ans unit literal.

Once we have transformed the input in this way then we run saturation (see [2]) as normal, with a few caveats. One important one is that we need to make sure that we never select an answer literal for inference as we want this literal left at the end. If we do not do this then we may saturate instead of finding a proof, as the ans literal is not explicitly added to the search space. To achieve this we alter the term ordering and literal selection functions. Whilst on the topic of proof search, notice that using answer literals alters proof search in other ways. The two main ways are that

1. Clauses with answer literals become heavier. The weight of a clause is used in clause selection and this can have the effect that clauses with answer literals are selected later. *However*, this could easily be fixed so that answer literals are ignored in clause weights (but is not currently).

2. Clauses that were units before adding the answer literal may not be units anymore. This means that they will no longer be eligible for demodulation.

3. Very subtly, as the shape of clauses are changed, the order in which they are retrieved from term indexing structures may change, which can also change the order they are met in clause selection.

If we want to avoid altering proof search then we need to extract answers directly from proofs (see later). The reason this matters is a proof without question answering may not translate across to one with question answering (with answer literals).

A key advantage of this approach is that we can extract an *explanation* of the answer in the form of a *proof* of the derived answer literal. Below I use some examples to discuss how the approach can be used to extract single answers, multiple answers, and disjunctive answers.

3.1 Finding Single Answers

In this section we consider a few examples demonstrating the use of answer literal reasoning to extract single answers to questions. Our first example was motivated by the workshops appearing at CADE alongside the VAMPIRE workshop. Consider the following problem (in TPTP) that states some facts about provers and workshps and then asks for something that is a prover and a workshop:

```
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))).
```

The conjecture will be transformed to the clause

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

and the following three clauses derived

$$\begin{gathered}\neg\texttt{prover}(\texttt{vampire}) \vee \neg\texttt{ans}(\texttt{vampire})\\ \neg\texttt{prover}(\texttt{arcade}) \vee \neg\texttt{ans}(\texttt{arcade})\\ \neg\texttt{ans}(\texttt{vampire})\end{gathered}$$

which gives the answer `vampire`. VAMPIRE would produce the following proof in this case where the last 2 steps implicitly add the answer literal `ans0(X0)` to resolve against. The first line uses the proposed TPTP format[1] to present the answers.

```
% 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 lit 6]
8. ! [X0] : (~workshop(X0) | ~prover(X0) | ~ans0(X0)) [ennf trans 7]
9. prover(vampire) [cnf transformation 1]
11. workshop(vampire) [cnf transformation 3]
13. ~workshop(X0) | ~prover(X0) | ~ans0(X0) [cnf transformation 8]
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]
```

Multi-Variable Questions. I now use a slightly more realistic example to demonstrate a question with multiple variables. Consider the following definition of a `max` function and a conjecture about its correctness.

```
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))
)).
```

[1] http://www.cs.miami.edu/~tptp/TPTP/Proposals/AnswerExtraction.html

The conjecture does not hold as the definition of the function is correct. We can transform the conjecture into a question asking for which inputs the function does not satisfy the required condition as follows:

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

VAMPIRE then gives the answer

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

which tells us that when any number x and $x + 1$ are input to the function it returns the wrong result.

Questions and Theories. Over the last few years, VAMPIRE has been extended with various techniques for reasoning with theories [4, 8, 5]. These can be used for question answering directly. For example, the very simple question of whether there are numbers greater than zero

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

results in the following answer from VAMPIRE

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

later we discuss what happens if we want more than one answer. The proof in this case is

```
1. ? [X0 : $int] : $greater(X0,0) [input]
2. ~? [X0 : $int] : $greater(X0,0) [negated conjecture 1]
3. ~? [X0 : $int] : $less(0,X0) [evaluation 2]
9. ~$less(X0,X0) (TD) [theory axiom]
13. $less(X1,$sum(X0,1)) | $less(X0,X1) (TD) [theory axiom]
17. ~? [X0 : $int] : ($less(0,X0) & ans0(X0)) [answer literal 3]
18. ! [X0 : $int] : (~$less(0,X0) | ~ans0(X0)) [ennf trans 17]
20. ~$less(0,X0) | ~ans0(X0) [cnf transformation 18]
37. $less(X0,$sum(X0,1)) (TD) [resolution 13,9]
43. ~ans0($sum(0,1)) [resolution 37,20]
44. ans0(X0) [answer literal]
45. $false [unit resulting resolution 44,43]
```

Here we can see (in steps 9 and 13) that VAMPIRE derives this result from two axioms of arithmetic i.e. this proof gives an explicit explanation for the given answer.

VAMPIRE can be used to solve quadratic equations. For example, the following question asks for a solution to $x^2 - 4 = 0$:

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

and VAMPIRE gives the expected solution:

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

Another example that I like is the following (taken from TPTP problem `MSC023=2.p`) where a function for converting from Celsius to Fahrenheit is defined and then the question of what Fahrenheit 451 is in Celsius is asked.

```
tff(celsius_fahrenheit_temperature_convert_type,type,(
  celsius_fahrenheit_temperature_convert: ( $real * $real ) > $o )).

tff(celsius_fahrenheit_temperature_conversion,axiom,(
  ! [C: $real,F: $real] :
    ( $sum($product(1.8,C),32.0) = F
   => celsius_fahrenheit_temperature_convert(C,F) ) )).

tff(fahrenheit_451_to_celsius,question,(
  ? [C: $real] : celsius_fahrenheit_temperature_convert(C,451.0) )).
```

To see what the answer is, try running VAMPIRE (but you will need to enable unification with abstraction [8]).

3.2 Finding Multiple Answers

Sometimes we might be interested in more than one answer. This solution is simple: to obtain multiple answers we don't stop when we have one answer. We can either print answers as they are found, or record answers and print them all when saturation is completed. In the latter case it is important to put a limit on the number of answers we want (in case there are an infinite number) and this may be useful in any case. For example, if we vary the question in our earlier example to

```
fof(a,question,?[X]: (prover(X)).
```

we get the following set of answers

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

Here we can see how the answer format handles multiple answers. Answers are given as a list and the last part `|_` indicates that there may be possibly more answers. Currently, VAMPIRE will always include this as there is no support for determining otherwise.

What happens when there are a finite set of answers is somewhat obvious. But, what happens when there are possibly infinite answers (especially in the context of arithmetic) is more interesting. However, the results are a little disappointing currently (because so far nothing has been done to make them otherwise). Given the question

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

asking for three answers gives us

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

which is interesting in some respects as it tells us that x and $-x + 5$ add together to make 5 for any x, which is more general than the second two (expected) answers. But when given the question

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

and asked for 10 answers, VAMPIRE will currently return

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

as there is not yet any mechanism for asking for *unique* answers. But note that each instance of 1 represents a separate derivation (something I omitted earlier is that *explanations* are less straightforward in the multiple answer case). A workaround here would be to remember previous answers and skip them. However, there is no obvious way to alter proof search to avoid such answers. The observant reader will notice that the answer `([1]|[0])` appears in the above; this means that *either* $0 > 0$ or $1 > 0$. We discuss such disjunctive answers next.

3.3 Disjunctive Answers

Consider a variant of our problem about workshops above. This problem tells us on which day each workshop takes place and asserts it is either Monday or Sunday before asking for an example of a workshop.

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

In this case there is no definitive answer but there is a disjunctive answer:

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

as we do not know if it is Monday or Sunday. This answer is produced by the following proof where we can see (on step 17) a clause containing two ans literals being produced.

```
1. monday => workshop(vampire) [input]
2. sunday => workshop(arcade) [input]
3. sunday | monday [input]
4. ? [X0] : workshop(X0) [input]
5. ~? [X0] : workshop(X0) [negated conjecture 4]
6. ~? [X0] : (workshop(X0) & ans0(X0)) [answer literal 5]
7. workshop(vampire) | ~monday [ennf transformation 1]
8. workshop(arcade) | ~sunday [ennf transformation 2]
9. ! [X0] : (~workshop(X0) | ~ans0(X0)) [ennf transformation 6]
10. workshop(vampire) | ~monday [cnf transformation 7]
11. workshop(arcade) | ~sunday [cnf transformation 8]
12. sunday | monday [cnf transformation 3]
13. ~workshop(X0) | ~ans0(X0) [cnf transformation 9]
14. ~monday | ~ans0(vampire) [resolution 13,10]
15. ~sunday | ~ans0(arcade) [resolution 13,11]
16. monday | ~ans0(arcade) [resolution 15,12]
17. ~ans0(arcade) | ~ans0(vampire) [resolution 16,14]
18. ans0(X0) [answer literal]
19. $false [unit resulting resolution 18,18,17]
```

3.4 Answer Literals and Other Parts of VAMPIRE

Here we consider how the answer literal approach interacts with other (modern) parts of VAMPIRE.

Working with AVATAR. AVATAR [6, 11] is a very useful component of VAMPIRE but it is not immediately compatible with question answering. This is because if we ever split the answer literal it becomes non-obvious when we have found an answer as a unit clause does not mean that the rest of the clause has been discharged. Although, in many cases it wouldn't split anyway as it will share variables with the rest of the clause. The current solution to this is that we do not split any clause containing an answer literal. This means that answers can still be found but it removes much of the advantage gained by AVATAR. A better solution should be found.

(Not) Working with Instantiation. There is an observation that instantiation methods would be eminently suitable for question answering approaches as their role is to find compatible instances. Currently, in VAMPIRE, there are two instantiation techniques for theories. The first heuristically uses constants already in the search space and the second uses Z3 to find a single useful instance [8]. Neither is particularly suited at finding interesting answers. Similarly, VAMPIRE incorporates the instanced-based method `InstGen` [1] and this is not utilised for question answering either. In general, it would be good if instantiation could be utilised to enumerate multiple instances (thus multiple answers) and to guide question answers towards particular 'kinds' of answers.

4 Alternative Approaches

In this section I describe two alternative approaches to question answering that could be implemented in VAMPIRE but have not yet been explored.

4.1 The Proof Analysis Approach

Previously, I introduced a new proof output for VAMPIRE that removed unification completely [3]. The idea of this approach was to expand each proof step into an instantiation step followed by a unification-free inference step. This proof output is implemented but not currently (at the time of writing) in the main branch of VAMPIRE. It is also missing some inference rules (i.e. they have not been split as described above).

Due to the fact that instantiation is made explicit in these proofs, it should be relatively straightforward to analyse the proofs directly to extract the instantiations of the conjecture required to reach a contradiction. As a short example, consider the following problem

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

which has the following VAMPIRE proof (in the format with explicit instantiation):

```
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]
```

Here we can directly see the conjecture clause being instantiation on step 8 to give the answer.

This can also work for disjunctive answers. For example, take the following problem

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

and VAMPIRE 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]
```

and notice that `X0` must be instantiated with both `c` and `d` during the proof (steps 8 and 11).

So far I have only presented cases where the conjecture clause is instantiated directly. However, this seems likely only in the simplest cases. In the following example I demonstrate the need for *tracking* the substitution and which variables in clauses will contribute to it. Take the problem

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

which has the 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]
```

and notice that we have two instantiations. First, `~p(X0)` is instantiated with `f(X0)` (step 10), which means that we have a partial substitution for the question variable of $f(y)$ where y corresponds to `X0` in the clause on line 10. Then, a resolution is carried out, meaning that the y in the substitution now also corresponds to `X0` in the clause on line 11. Finally, this clause is instantiated with `a` before resolving to give `$false`. This means that the final substitution (and answer) is $f(a)$.

It is clear that there are some details that still need working out. Note that this method cannot be used to find multiple answers, although one could attempt to force different proofs with the hope that they lead to different answers.

4.2 Using Finite Models

The last idea is to extract answers to questions from finite models constructed by VAMPIRE's finite model building mode [7]. Here the approach would be a little different; one would first construct a model for the axioms A and then evaluate queries in this model. VAMPIRE already has support for evaluating formulas in an existing finite model using the (currently undocumented) `model_check` mode. This mode takes a finite model given in TPTP format and a list of terms or formulas and then reports the evaluation of these terms and formulas in this model.

An example where we have explored this idea previously is in the set of *Knights and Knaves* problems in the TPTP library. These problems typically describe some constraints on what knight and knave characters do and then conjecture some relationship between them (e.g. that at least one tells the truth). Here we would build a model of the underlying constraints and then ask for an individual satisfying a given relationship.

It would be relatively straightforward to extend the above machinery to perform the finite model building and question evaluation in one step. This appears to be a strong approach that should be explored. However, there are two caveats:

1. Not all problems have finite models, or finite models that are small enough to be found by our method.

2. Finite model finding in VAMPIRE cannot currently handle theories.

Therefore, for some problems this approach could not be applicable. Additionally, only a single answer would be extracted from the single finite model. To generate multiple finite models VAMPIRE would need to be extended to enumerate finite models. In general it could be possible to combine this approach with the previous ideas to implement a general portfolio question answering service that attempted various question answering methods.

5 Conclusion

In this short paper I have revisited the question answering problem from a VAMPIRE perspective. This is an old problem that I haven't brought much new insight to, but have discussed some VAMPIRE-specific issues. The paper suggests some improvements that could be made to VAMPIRE. Here I summarise these and add a couple of new ones:

- Find a better solution to using AVATAR for question answering

- Look at proof search to see where we can avoid unnecessarily altering proof search when reasoning with answer literals.

- Detect repeated answers in the saturation-based approach. Also investigate whether it is possible to filter for *more* or *less* general answers using the groundedness of the answer as a proxy for this.

- Implement a method that analyses proofs with separated instantiation steps to extract question answers

- Extend finite-model-building to extract question answers from finite models

- Finally, in the case where VAMPIRE can produce multiple answers, one could wrap it up as an *iterator* so that the user has to explicitly ask for the next answer and then VAMPIRE continues proof search to find this answer.

References

[1] Konstantin Korovin. *Inst-Gen – A Modular Approach to Instantiation-Based Automated Reasoning*, pages 239–270. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[2] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV 2013*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35, 2013.

[3] Giles Reger. Better proof output for Vampire. In Laura Kovács and Andrei Voronkov, editors, *Vampire@IJCAR 2016. Proceedings of the 3rd Vampire Workshop, Coimbra, Portugal, July 2, 2016.*, volume 44 of *EPiC Series in Computing*, pages 46–60. EasyChair, 2016.

[4] Giles Reger, Nikolaj Bjørner, Martin Suda, and Andrei Voronkov. AVATAR modulo theories. In Christoph Benzm\"uller, Geoff Sutcliffe, and Raul Rojas, editors, *GCAI 2016. 2nd Global Conference on Artificial Intelligence*, volume 41 of *EPiC Series in Computing*, pages 39–52. EasyChair, 2016.

[5] Giles Reger and Martin Suda. Set of support for theory reasoning. In Thomas Eiter, David Sands, Geoff Sutcliffe, and Andrei Voronkov, editors, *IWIL Workshop and LPAR Short Presentations*, volume 1 of *Kalpa Publications in Computing*, pages 124–134. EasyChair, 2017.

[6] Giles Reger, Martin Suda, and Andrei Voronkov. Playing with AVATAR. In P. Amy Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, pages 399–415, Cham, 2015. Springer International Publishing.

[7] Giles Reger, Martin Suda, and Andrei Voronkov. Finding finite models in multi-sorted first-order logic. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016: 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, pages 323–341. Springer International Publishing, 2016.

[8] Giles Reger, Martin Suda, and Andrei Voronkov. Unification with abstraction and theory instantiation in saturation-based reasoning. EasyChair Preprint no. 1, EasyChair, 2017. `https://easychair.org/publications/preprint/1`.

[9] Geoff Sutcliffe. The TPTP problem library and associated infrastructure. *J. Autom. Reasoning*, 43(4):337–362, 2009.

[10] Geoff Sutcliffe, Aparna Yerikalapudi, and Steven Trac. Multiple answer extraction for question answering with automated theorem proving systems. In *Proceedings of the Twenty-Second International Florida Artificial Intelligence Research Society Conference, May 19-21, 2009, Sanibel Island, Florida, USA*, 2009.

[11] Andrei Voronkov. AVATAR: The architecture for first-order theorem provers. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, volume 8559 of *Lecture Notes in Computer Science*, pages 696–710. Springer International Publishing, 2014.