# Multiple Answer Extraction for Question Answering with Automated Theorem Proving Systems

**Geoff Sutcliffe** and **Aparna Yerikalapudi** and **Steven Trac**
University of Miami, USA

**Abstract**

The Multiple ANSwer EXtraction system is a framework for interpreting a conjecture with outermost existentially quantified variables as a question, and extracting multiple answers to the question by repetitive calls to a base system that can report the bindings for the variables in one proof of the conjecture. This paper describes the framework and demonstrates its use on an illustrative example.

## Introduction

The use of theorem proving to implement question answering has received intermittent attention since the early days of automated reasoning (Slagle 1965; Green 1969; Minker, Fishman, and McSkimin 1973; Baumgartner, Furbach, and Stolzenburg 1997; Burhans 2002; Waldinger 2007). The standard approach is to treat a conjecture with outermost existentially quantified variables - the *answer variables* - as a question, and report the bindings for the answer variables in a proof, as an answer to the question. If multiple proofs are available, correspondingly multiple answers are available (although multiple proofs may return the same answer if they generate the same variable bindings). If the conjecture is used more than once in a proof, a disjunctive answer is returned, with the bindings from each use forming one alternative of the answer. Regardless of the minor variations, the capability of returning such answers is important to users of reasoning systems, particularly for applications in common sense/knowledge based reasoning over real world data, e.g., (Curtis, Matthews, and Baxter 2005; Kasneci et al. 2008). The framework described in this paper has been developed in the context of Automated Theorem Proving (ATP) for first-order logic with equality.

Few modern ATP systems are able to return answers. Examples of systems that do are Otter (McCune 1994), the customized version of Vampire used in the Sigma Knowledge Engineering Environment (SigmaKEE) (Pease 2003), and SNARK (Stickel URL). These systems are all capable of returning multiple answers, by searching for multiple proofs. Their question answering capabilities have been used in more complex reasoning systems, SigmaKEE in the case of Vampire, the Amphion project (Stickel et al. 1994) and BioDeducta (Shrager et al. 2007) (as examples) in the case of SNARK. Sadly Otter and the customized Vampire are older, and now unsupported, systems.

The limited availability of question answering in modern ATP systems, and the desire to be able to use the best state-of-the-art systems for reasoning and question answering, motivated the development of the framework described in this paper.[1] The approach taken is to provide a base system that can return a single definite answer to a question, and to obtain multiple such answers by repeated calls to that system. At each iteration the conjecture is augmented to deny previously extracted answers, so that successive answers are different. The base system is able to use any ATP system that returns a TPTP format proof (Sutcliffe et al. 2006) (or a proof that can be translated into TPTP format), from which proof analysis and further ATP system runs provide the necessary single answers. The base system is thus able to leverage the power of many modern state-of-the-art ATP systems.

The use of a separate underlying system to return single answers means that issues of answer completeness, answer optimality, etc, are largely relegated to the underlying system. Their calculi, implementation, and other properties are important, but beyond the control of the implemented framework per se. The selection of the underlying system to use in the framework is of course within the control of the user of the framework, and that selection can affect the performance and properties of the framework. Investigation of these issues in the context of the framework will be worthwhile. Additionally, the implemented framework does not yet return disjunctive answer when the conjecture is used multiple times.

### Notation and a Running Example

The system framework has been developed using various components of the TPTPWorld (Sutcliffe 2007). All logical data is written in the TPTP language (Sutcliffe and Suttner URL), A TPTP annotated formula has the form *language*(*name*, *role*, *formula*). The *language*s currently supported are `fof` - formulae in full first order form, and `cnf` - formulae in clause normal form. The *name* identifies the formula within the problem, but might not be unique (unique-

[1]There is scant hope that ATP system developers will add question answering to their systems without significant inducement.

ness is not enforced to allow users of the TPTP language to be "lazy" if they want). The *role* gives the user semantics of the *formula*, e.g., `axiom`, `lemma`, `conjecture`, and hence defines its use in an ATP system. The logical *formula*, in either FOF or CNF, uses a Prolog-like syntax: variables start with upper case letters, atoms and terms are written in prefix notation, uninterpreted predicates and functors either start with lower case and contain alphanumerics and underscore, or are in `'single quotes'`. The only unary connective is ~ for negation. The binary connectives are `&`, `|`, `=>`, `<=`, `<=>`, and `<~>`, for conjunction, disjunction, implication, reverse implication, equivalence, and inequivalence respectively. The universal and existential quantifiers are `!` and `?` respectively, with the quantified variables following in `[ ]` brackets.

Results from ATP systems are expressed using the SZS ontology using the SZS format for reporting results (Sutcliffe 2008). The new formula role `question` has been proposed for conjectures for which answers are required, and an SZS format extension has been proposed for reporting answers.[2] The "instantiated form" of the proposed format has been used in this work, in which an answer is given by the question formula with the existentially quantified variables instantiated to the answer values. Multiple answers are written as a comma separated list of instantiated formulae, in `[ ]` brackets, e.g., `[p(a,b),p(b,Z),p(a,c)]`. (This format thus maintains the TPTP's Prolog compatibility.) The format for reporting answers parallels the format for reporting results, and is

`SZS answers instantiated for` *problem answers*

For example

```
SZS answers instantiated for ANS001+1 [p(a,b),p(b,Z),p(a,c)]
```

The following problem will be used as a running example through this paper.

```
fof(pab,axiom,       p(a,b)           ).
fof(pafbc,axiom,     p(a,f(b,c))      ).
fof(pbZ,axiom,       ! [Z] : p(b,Z)   ).
fof(pZc,axiom,       ! [Z] : p(Z,c)   ).
fof(a_not_b,axiom,   a != b           ).
fof(a_is_c,axiom,    a = c            ).
fof(pXY,question,    ? [X,Y] : p(X,Y) ).
```

One possible list of answers is `[p(a,b),p(b,Z),p(a,c)]`, but there are others, as is explained in the following sections.

## Basic Multiple Answer Extraction

The basic multiple answer extraction framework is built over a base system that returns a single answer. The base system uses an ATP system to produce a TPTP format proof for the conjecture. The proof may be produced natively in TPTP format by the ATP system, e.g., by EP (Schulz 2002), or may be translated from the ATP system's format to the TPTP format using the TPTPWorld's X2tstp utility, e.g., proofs output by Vampire (Riazanov and Voronkov 2002). The proof must include the FOF to CNF translation steps. The TPTP format proof is passed to the One Answer Extraction System (OAESys) – a tool for extracting the bindings for answer variables, from a TPTP format proof of the conjecture.

TPTP format proofs from most ATP system do not record the substitutions made in each inference step, as are needed to extract the bindings for the answer variables. Therefore, OAESys starts by reproving the conjecture using the Metis system (Hurd 2003), from only the axioms used in the original proof. Metis (is the only system that we know of that) outputs TPTP format proofs with details of the substitutions made in each inference step. Metis' proof is analysed by identifying the answer variables in the conjecture, tracing the path from the conjecture to the root of the proof (a $false$ node, as Metis' does proof by contradiction of the clause normal form of the problem), and recording the variable bindings made in each inference step on the path. The composition of these substitutions provides the bindings for the answer variables.

Giving Metis only the axioms used in the original proof (rather than all the axioms in the original problem) makes it most probable that Metis will find a proof with the same answer variable bindings as the original proof. If the axioms used are a subset of the axioms that were available in the original problem, the problem given to Metis could be significantly easier than the original problem. OAESys also has an option to form a potentially even easier problem for Metis, to prove the conjecture from the axioms and lemmas that are side-formulae of inferences on the path from the conjecture to the root of the original proof. Analysis of Metis' proof of this problem also provides the answer variable bindings. It is of note that the reproving by Metis provides a level of verification for the original systems' proof, in the sense of semantic derivation verification (Sutcliffe 2006) - if Metis is trusted then the ATP system that finds the original proof need not be trusted. In the unlikely event that Metis is unable to reprove the conjecture, the framework capitulates, and responds as if a proof had not been found by the original ATP system.

The ATP+OAESys base system can be replaced by any ATP system that can return an answer by itself, e.g., one of those mentioned in the introduction, but of those only SNARK is currently supported and capable of returning an answer in the SZS format.

After an answer has been found by the base system, the multiple answer extraction framework augments the conjecture to deny previously extracted answers. This can be done in two ways. If the problem contains inequality information, the conjecture is conjoined with inequality constraints that require each previous answer to have an element that is unequal to the corresponding element of the next answer. For the running example, after the first answer `p(a,b)` the question is augmented to

```
fof(pXY,question,
  ? [X,Y] :
    ( p(X,Y)
    & (X != a | Y != b) ) ).
```

This produces the second answer `p(b,Z)`, and the question is augmented to

[2]See the proposal at `http://www.tptp.org/TPTP/Proposals/AnswerExtraction.html`

```
fof(pXY,question,
  ? [X,Y] : ! [Z] :
    ( p(X,Y)
    & (X != a | Y != b)
    & (X != b | Y != Z) ) ).
```

Note that the universally quantified variable from the second answer is universally quantified inside the existentially quantified variables of the question. This produces the third answer `p(a,c)`, and the question is augmented to

```
fof(pXY,question,
  ? [X,Y] : ! [Z] :
    ( p(X,Y)
    & (X != a | Y != b)
    & (X != b | Y != Z)
    & (X != a | Y != c) ) ).
```

This question cannot be proved, so the final list of answers is `[p(a,b),p(b,Z),p(a,c)]`. In these answers `a` and `c` are interchangeable because they are equal. Thus the answers returned depend on the reasoning done by the base system.

Many problems do not contain enough inequality information to prove the augmented questions. This may be because equality is not part of the problem specification, because inequality is not necessary for finding a proof (for the original question), or because there is an implicit or explicit unique names assumption. In this case, rather than requiring answer elements to be unequal, they are required to "look different". After each answer is extracted, axioms are added that specify what terms look different. For each function symbol (including constants) in the answer, new axioms are added to specify that terms with that principle symbol look different to terms with any other principle symbol. Additionally the axiom of symmetry for looking different is provided. The question is then conjoined with constraints that require each previous answer to have an element that looks different to the corresponding element of the next answer, analogous to the inequality constraints. For the running example, after the first answer `p(a,b)`, the axioms

```
fof(ld__symmetry,axiom,
  ! [X,Y] :
    ( ld__(X,Y) => ld__(Y,X) ) ).
fof(ld__a_b,axiom,  ld__(a,b) ).
fof(ld__a_c,axiom,  ld__(a,c) ).
fof(ld__a_f,axiom,
  ! [X,Y] : ld__(a,f(X,Y)) ).
fof(ld__b_c,axiom,  ld__(b,c) ).
fof(ld__b_f,axiom,
  ! [X,Y] : ld__(b,f(X,Y)) ).
```

are added, and the question is augmented to

```
fof(pXY,question,
  ? [X,Y] :
    ( p(X,Y)
    & (ld__(X,a) | ld__(Y,b)) ) ).
```

This results in the second answer `p(a,f(b,c))`, after which the axioms

```
fof(ld__c_f,axiom,
  ! [X,Y] : ld__(c,f(X,Y)) ).
fof(ld__f_f,axiom,
  ! [X11,X12,X21,X22] :
    ( ( ld__(X11,X12) | ld__(X21,X22) )
   => ld__(f(X11,X21),f(X12,X22)) ) ).
```

are added, and the question is augmented to

```
fof(pXY,question,
  ? [X,Y] :
    ( p(X,Y)
    & ( ld__(X,a) | ld__(Y,b) )
    & ( ld__(X,a) | ld__(Y,f(b,c)) ) ) ).
```

This results in the third answer `p(b,Z)`, etc. The final list of answers for the running example is `[p(a,b), p(a,f(b,c)), p(b,Z), p(a,c), p(c,c), p(f(X,Y),c)]`. The first two answers are neither equal nor unequal, the fourth and fifth are equal, and the last is neither equal nor unequal to the fourth or fifth.

The incremental addition of the looks-different axioms after each answer is extracted is important. The number of axioms is polynomial in the number of function symbols, so for problems with a large number of function symbols (as is often the case in common sense/knowledge based reasoning) adding all the axioms in advance is likely to degrade the performance of the base system. Only the axioms for the function symbols in the answers extracted so far are necessary at each iteration, and the number of such axioms is linear in the number of symbols in the axioms and the number of symbols in the answers extracted so far.

The looks-different approach can run into difficulties if used on problems that have positive equality information. Consider the following example:

```
fof(pa,axiom,      p(a) ).
fof(pb,axiom,      p(b) ).
fof(a_is_b,axiom,  a = b ).
fof(pX,question,   ? [X] : p(X) ).
```

This problem does not have any inequality information, which suggests use of the looks-different approach. After the first answer `p(a)` has been returned, the looks-different axioms that are added are

```
fof(ld__symmetry,axiom,
  ! [X,Y] :
    ( ld__(X,Y) => ld__(Y,X) ) ).
fof(ld__a_b,axiom,  ld__(a,b) ).
```

and the question is augmented to

```
fof(pX,question,
  ? [X] : ( p(X) & ld__(X,a) ) ).
```

Equality reasoning between the `a_is_b` and `ld__a_b` axioms can produce the result that `ld__(a,a)`, which in turn allows the question to be answered with `p(a)` again (and again, and again). This undesirable repetition is an artifact of using the looks-different approach due to a lack of inequality information, in the face of some equality information - you can't have your cake and eat it. Possible solutions are to drop back into requiring inequality of new solutions with the existing solution, or terminating the multiple answer extraction. The current implementation adopts the (simpler) latter approach.

Within the multiple answer extraction framework, when the base system times out because it cannot find a proof, a model finder, e.g., Paradox (Claessen and Sorensson 2003), Darwin (Baumgartner, Fuchs, and Tinelli 2006), Mace4 (McCune 2003), etc., or saturating ATP system, e.g., EP, SPASS (Weidenbach et al. 2007), etc., can be used to try show that there are no more answers. If this is successful it assures the user that all answers have been found, either with respect to inequality or with respect to looking different.

This approach to extracting multiple answers differs from that of asking the ATP system to find multiple proofs, in that it does not allow the same answer to be returned multiple times. If looks-different constraints are added then the answers must be syntactically different, and if inequality constraints are added the answers must additionally be semantically different. For real world question answering might be desirable to return multiple answers that look different, even if some of them are equal - the effect is to return synonyms for answers. For example, it might be useful to answer the question `? [X] : author(X,jabberwocky)` with the tuple `[auther('Lewis Carroll',jabberwocky), author('Charles Dodgson',jabberwocky)]`. If such multiple answers are required then the looks-different can be used even if inequality information is given, but it is then necessary to tolerate the possibile undesirable interaction described above.

## Equality Testing Looks-different Answers

The basic system described in the previous section either uses inequality reasoning, or ignores inequality information and uses the looks-different approach. It is possible for a problem to have some equality and inequality information, but not enough to use only the inequality approach. Equality testing of looks-different answers acknowledges this situation, and uses the looks-different approach followed by equality and inequality reasoning to test whether or not each new answer is equal or unequal to previous answers. The tests are in the form of conjectures built from equalities and inequalities, to be proved (by a theorem proving ATP system) or shown unprovable (by a model finder or saturating ATP system) from the axioms. The ATP systems used might fail to solve the problems posed, either because the problem is not solvable, or because the problem is too hard for the system within the CPU limit imposed. Therefore a series of tests of decreasing strength are available, and the extent to which these are used is controlled by a user parameter.

After an answer has been extracted using the looks-different approach, the first test tries to prove that the answer is unequal to all previous answers. If this succeeds the answer is accepted. For the running example, following the first two answers `p(a,b)` and `p(a,f(b,c))`, the next answer returned by the base system is `p(b,Z)`. The conjecture to prove is

```
fof(new,conjecture,
  ( ? [Z] :
      ( b != a | Z != b )
  & ? [Z] :
      ( b != a | Z != f(b,c) ) ) ).
```

which succeeds and the answer is accepted. Universally quantified variables from the new answer are existentially quantified in each conjunct of the conjecture because any instantiation for the variables makes the new answer unequal to the previous answer. Variables in previous answers would be universally quantified inside such existential quantification.

If the first test fails, the second test is used to try prove that the new answer is equal to any previous answer. If this succeeds the answer is reported to be equal to that previous answer and rejected. It's symbols are however used for adding looks-different axioms for the next iteration of the framework. For the running example, this case occurs after the first four answers `p(a,b)`, `p(a,f(b,c))`, `p(b,Z)`, and `p(a,c)`. The next answer returned by the base system is `p(c,c)`. The conjecture to prove is

```
fof(old,conjecture,
    ( ( c = a & c = b )
    | ( c = a & c = f(b,c) )
    | ? [Z] : ( c = b & c = Z )
    | ( c = a & c = c ) ) ).
```

which succeeds because the last disjunct is true, and the answer is rejected. Universally quantified variables from the previous answer are existentially quantified in each disjunct of the conjecture because any instantiation for the variables makes the previous answer equal to the new answer. Variables in the new answer would be universally quantified outside such existential quantification.

If the second test fails and the user has not specified use of weak tests, the answer is rejected (with it's symbols being retained for adding looks-different axioms). If the user has specified use of weak tests then the third test tries to show that the new answer cannot be proven equal to any previous answer. If this succeeds the answer is accepted. The conjecture to be shown counter-satisfiable is the same as for the second test. For the running example, this case occurs following the first answer `p(a,b)`. The next answer returned by the base system is `p(a,f(b,c))`. The conjecture to be shown counter-satisfiable is

```
fof(old,conjecture,
  ( a = a
  & f(b,c) = b ) ).
```

which succeeds because the second conjunct cannot be proved, and the answer is accepted.

If the third test fails, the fourth test is used to try show that the new answer cannot be proven unequal to all previous answers. If this succeeds the answer is rejected. The conjecture to be shown counter-satisfiable is the same as for the first test. For the running example, this case could occur following the first three answers `p(a,b)`, `p(a,f(b,c))`, and `p(b,Z)`. The next answer returned by the base system is `p(a,c)`. This answer is in fact accepted by the third test, but imagine it's not for the sake of illustration. The conjecture to be shown counter-satisfiable would be

```
fof(new,conjecture,
  ( ( a != a | c != b )
  & ( a != a | c != f(b,c) )
  & ! [Z] : ( a != b | c != Z ) ) ).
```

which succeeds, and the new answer is rejected.

If the fourth test fails the new answer is accepted.

## Conclusion

Figure 1 shows the overall flow of the framework. The framework has been implemented in `perl`, and is available for use through the SystemOnTPTP interface at `http://www.tptp.org/SystemOnTPTP`. The implementation has additional features that allow the user to control the iterative loop of the framework, to ask for, e.g., all answers, all answers until a specified answer is found, all answers while a specified answer has not been found, etc. The user can also perform basic set operations on answer lists, e.g., asking if a particular answer is an element of the list, whether the list is a subset, equal set, or superset of a provided list, etc.

The basic framework has also been implemented in Java within the Sigma Knowledge Engineering Environment (SigmaKEE) (Pease 2003; Trac, Sutcliffe, and Pease 2008). SigmaKEE is used primarily for interacting with the Suggested Upper Merged Ontology (SUMO) (Niles and Pease 2001), in particular for reasoning and question answering over the SUMO. Previously SigmaKEE relied on a customized version of Vampire to answer user's questions. The integration of the basic framework has provided SigmaKEE with more powerful options for question answering. In the SigmaKEE context the answers are linked back into the knowledge base that forms the axioms for the user's queries. This means that both the answer and the proof that produced the answer are needed for presentation to the user. To provide both in a form consistent with the user's original question, the base system has been extended to remove the conjecture augmentations done for the second and subsequent iterations, as follows. After each answer has been extracted by OAESys, the answer variables in the original conjecture, i.e., the conjecture without the augmentations, are instantiated with the answer. This instantiated conjecture and just the axioms used in the proof found by the ATP system are passed to that ATP system. This additional ATP system run finds a proof of the (instantiated form of the) original conjecture, rather than of the augmented conjecture. This extension also adds another level to the verification notion mentioned in the section describing basic multiple answer extraction - now the original system may be trusted rather than Metis.

Future work includes extending the framework to disjunctive answers, and studying to what extent the properties of the underlying system (returning the single answers) are transferred to the multiple answer extraction framework.

**Acknowledgment:** Thanks to Mark Stickel for his helpful input, and adaptation of SNARK to the proposed SZS question answering standards.

## References

Baumgartner, P.; Fuchs, A.; and Tinelli, C. 2006. Implementing the Model Evolution Calculus. *International Journal on Artificial Intelligence Tools* 15(1):21–52.

Baumgartner, P.; Furbach, U.; and Stolzenburg, F. 1997. Computing Answers with Model Elimination. *Artificial Intelligence* 90:135–176.

Burhans, D. 2002. A Question Answering Interpretation of Resolution Refutation. Technical Report Technical Report 2002-03, Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, USA.

Claessen, K., and Sorensson, N. 2003. New Techniques that Improve MACE-style Finite Model Finding. In Baumgartner, P., and Fermueller, C., eds., *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*.

Curtis, J.; Matthews, G.; and Baxter, D. 2005. On the Effective Use of Cyc in a Question Answering System. In Benamara, F.; Moens, M.; and Saint-Dizier, P., eds., *Proceedings of the Workshop on Knowledge and Reasoning for Answering Questions, 19th International Joint Conference on Artificial Intelligence*, 61–70.

Green, C. 1969. Theorem Proving as a Basis for Question-answering Systems. *Machine Intelligence* 4:183–205.

Hurd, J. 2003. First-Order Proof Tactics in Higher-Order Logic Theorem Provers. In Archer, M.; Di Vito, B.; and Munoz, C., eds., *Proceedings of the 1st International Workshop on Design and Application of Strategies/Tactics in Higher Order Logics*, number NASA/CP-2003-212448 in NASA Technical Reports, 56–68.

Kasneci, G.; Suchanek, F.; Ifrim, G.; Ramanath, M.; and Weikum, G. 2008. NAGA: Searching and Ranking Knowledge. In Alonso, G.; Blakeley, J.; and Chen, A., eds., *Proceedings of the 24th International Conference on Data Engineering*, 697–706.

McCune, W. 1994. Otter 3.0 Reference Manual and Guide. Technical Report ANL-94/6, Argonne National Laboratory, Argonne, USA.

McCune, W. 2003. Mace4 Reference Manual and Guide. Technical Report ANL/MCS-TM-264, Argonne National Laboratory, Argonne, USA.

Minker, J.; Fishman, D.; and McSkimin, J. 1973. The Q* Algorithm - A Search Strategy for a Deductive Question-Answering System. *Artificial Intelligence* 4:225–243.

Niles, I., and Pease, A. 2001. Towards A Standard Upper Ontology. In Welty, C., and Smith, B., eds., *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems*, 2–9.

Pease, A. 2003. The Sigma Ontology Development Environment. In Giunchiglia, F.; Gomez-Perez, A.; Pease, A.; Stuckenschmidt, H.; Sure, Y.; and Willmott, S., eds., *Proceedings of the IJCAI-03 Workshop on Ontologies and Distributed Systems*, number 71 in CEUR Workshop Proceedings.

Riazanov, A., and Voronkov, A. 2002. The Design and Implementation of Vampire. *AI Communications* 15(2-3):91–110.

Schulz, S. 2002. E: A Brainiac Theorem Prover. *AI Communications* 15(2-3):111–126.

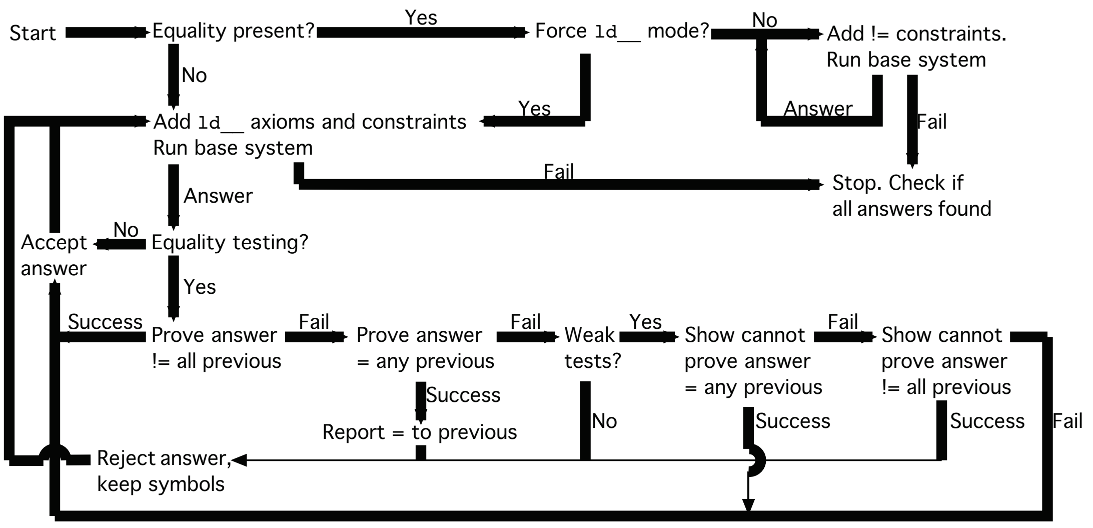
Start
Equality present?
Yes
Force ld__ mode?
No
Add != constraints. Run base system
No
Yes
Add ld__ axioms and constraints Run base system
Answer
Fail
Fail
Stop. Check if all answers found
Answer
Accept answer
No
Equality testing?
Yes
Success
Prove answer != all previous
Fail
Prove answer = any previous
Fail
Weak tests?
Yes
Show cannot prove answer = any previous
Fail
Show cannot prove answer != all previous
Success
Report = to previous
No
Success
Success
Fail
Reject answer, keep symbols

Figure 1: Multiple ANSwer Extraction framework

Shrager, J.; Waldinger, R.; Stickel, M.; and Massar, P. 2007. Deductive Biocomputing. *PLoS ONE* 2(4).

Slagle, J. 1965. A Proposed Preference Strategy using Sufficiency Resolution for Answering Questions. Technical Report UCRL-14361, Lawrence Radiation Laboratory, Livermore, USA.

Stickel, M.; Waldinger, R.; Lowry, M.; Pressburger, T.; and Underwood, I. 1994. Deductive Composition of Astronomical Software from Subroutine Libraries. In Bundy, A., ed., *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in Lecture Notes in Artificial Intelligence, 341–355. Springer-Verlag.

Stickel, M. URL. SNARK - SRI's New Automated Reasoning Kit. http://www.ai.sri.com/ stickel/snark.html.

Sutcliffe, G., and Suttner, C. URL. The TPTP Problem Library. http://www.TPTP.org.

Sutcliffe, G.; Schulz, S.; Claessen, K.; and Van Gelder, A. 2006. Using the TPTP Language for Writing Derivations and Finite Interpretations. In Furbach, U., and Shankar, N., eds., *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, 67–81.

Sutcliffe, G. 2006. Semantic Derivation Verification. *International Journal on Artificial Intelligence Tools* 15(6):1053–1070.

Sutcliffe, G. 2007. TPTP, TSTP, CASC, etc. In Diekert, V.; Volkov, M.; and Voronkov, A., eds., *Proceedings of the 2nd International Computer Science Symposium in Russia*, number 4649 in Lecture Notes in Computer Science, 7–23. Springer-Verlag.

Sutcliffe, G. 2008. The SZS Ontologies for Automated Reasoning Software. In Sutcliffe, G.; Rudnicki, P.; Schmidt, R.; Konev, B.; and Schulz, S., eds., *Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and The 7th International Workshop on the Implementation of Logics*, number 418 in CEUR Workshop Proceedings, 38–49.

Trac, S.; Sutcliffe, G.; and Pease, A. 2008. Integration of the TPTPWorld into SigmaKEE. In Schmidt, R.; Konev, B.; and Schulz, S., eds., *Proceedings of the Workshop on Practical Aspects of Automated Reasoning, 4th International Joint Conference on Automated Reasoning*, Accepted.

Waldinger, R. 2007. Whatever Happened to Deuctive Question Answering? In Dershowitz, N., and Voronkov, A., eds., *Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 4790 in Lecture Notes in Artificial Intelligence, 15–16.

Weidenbach, C.; Schmidt, R.; Hillenbrand, T.; Rusev, R.; and Topic, D. 2007. SPASS Version 3.0. In Pfenning, F., ed., *Proceedings of the 21st International Conference on Automated Deduction*, number 4603 in Lecture Notes in Artificial Intelligence, 514–520. Springer-Verlag.