External Sources of Axioms in Automated Theorem Proving

Martin Suda[1,⋆], Geoff Sutcliffe[2,⋆], Patrick Wischnewski[3], Manuel Lamotte-Schubert[3], and Gerard de Melo[3]

[1] Charles University in Prague, Czech Republic
[2] University of Miami, USA
[3] Max-Planck-Institut für Informatik, Germany

Abstract. In recent years there has been a growing demand for Automated Theorem Proving (ATP) in large theories, which often have more axioms than can be handled effectively as normal internal axioms. This work addresses the issues of accessing *external sources of axioms* from a first-order logic ATP system, and presents an implemented ATP system that retrieves external axioms asynchronously, on demand.

1 Introduction

In recent years there has been a growing demand for Automated Theorem Proving (ATP) in large theories. A *large theory* is one that has many functors and predicates, has many axioms of which typically only a few are required for the proof of a theorem, and many theorems to be proved using the axioms. Examples of large theories that are in (or have been translated to) a form suitable for ATP include the SUMO ontology, the Cyc knowledge base, the Mizar mathematical library, the YAGO knowledge base, WordNet, and the MeSH thesaurus. Many of these consist of lots (millions) of atomic facts, mostly positive ground facts. Large theories also arise from dynamic and computational sources of data. Such sources include online knowledge bases, computer algebra systems, lemma discovery tools, bioinformatics databases and tools, and web services. Dynamic and computational sources can provide infinitely many axioms. Large theories pose challenges for ATP, which are different from the challenges of small theory applications of ATP.

This paper addresses the issues of accessing large theories' axioms, stored as *external sources of axioms*, from a first-order ATP system. External sources of axioms provide further challenges for ATP. These include specifying what axioms are (or might be) available, building interfaces to the sources, retrieving and integrating axioms on demand during deduction, and adapting the deduction calculus to deal with axioms becoming available only after deduction has started.

⋆ Work done in the Automation of Logic group, Max-Planck-Institut für Informatik. Martin Suda supported by the Grant Agency of Charles University, grant 9828/2009. Geoff Sutcliffe supported by European Community FP7 grant PIIF-GA-2008-219982.

B. Mertsching, M. Hund, and Z. Aziz (Eds.): KI 2009, LNAI 5803, pp. 281–288, 2009.

2 Abstract System Design

This section describes design choices for a system that accesses external sources of axioms, and highlights the decisions taken for our implementation. (The design choice points are numbered, and the decisions taken are numbered correspondingly.) The general system architecture is composed of a first-order ATP system and external sources of axioms, as shown in Figure 1. The ATP system accepts a problem file containing (i) specifications for external sources of axioms, (ii) axioms to be read and stored internally, and (iii) a conjecture to be proved. It is understood that access to the external sources of axioms will be comparatively slow, and that their retrieval might be incomplete.

The Nature of External Axioms. (1) The *types of axioms* range from simple, e.g., positive ground facts, to highly expressive, e.g., full first-order formulae. The most common type is positive ground facts. (2) *Features of the axioms* can be different from those traditionally found in axiom sets. The axioms may be inconsistent, have varying epistemic and assertional status, and be temporal. (3) The *storage of axioms* can use a range of technologies, such as relational databases, semantic nets, RDF triples, executable programs, and web services.

ATP Systems' Use of External Axioms. (4) An ATP system's *interface to external axioms* has several facets. These include how their availability is specified, what roles they may have, the structure of requests for their retrieval, the format in which they are delivered, and their external manifestation. (5) The decision *when to retrieve* external axioms affects their integration into the ATP process. Advance retrieval allows the external axioms to be treated like internal axioms. Deduction-time retrieval requires that they be integrated dynamically. Deduction-time retrieval can be synchronous, or asynchronous, and can use a pull or push approach. (6) The *criteria for making a request* depend on the nature of the ATP system and its calculus. (7) There are several levels of *granularity for delivery* of external axioms. The finest grain delivery is one axiom at a time. Delivering batches of axioms is possible, but may need to be constrained to deliver less than all the axioms that match a request. Uniqueness requirements might be imposed, and the delivery order might be specified. (8) The ATP system has choices regarding the *storage and retention* of external axioms. These include where they should be stored, and whether they should be stored persistently.

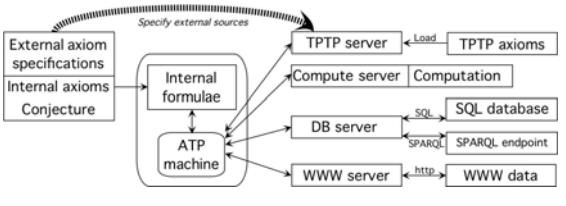

Fig. 1. System Architecture

Design Decisions. The design decisions for this work were often the simple ones, to avoid complication while developing the initial system, and also influenced by the decision to implement in the context of the SPASS ATP system [5], which is a CNF saturation-style ATP system. The choices are:

(1) The external axioms are positive ground facts (ground atoms).
(2) Positive ground facts are necessarily consistent, and it is assumed that the external axioms are consistent with the internal ones. The external axioms are also assumed to be certain, precise, and non-temporal.
(3) No constraints are imposed on the external storage technology.
(4) A TPTP-style specification of the availability of external axioms is used, and formulae with the TPTP role `axiom` are delivered. An extension of the TPTP standard for questions and answers[1] is used for requests and delivery. Initially the external manifestation will be a local process.
(5) External axioms are pulled asynchronously during deduction.
(6) External axioms are requested when a negative literal of the chosen clause (of the ATP system's saturation loop) matches an external specification.
(7) External axioms are delivered in batches, with the possibility of limiting the number of axioms that are delivered. There are no requirements for uniqueness, but no request is issued more than once.
(8) The external axioms are integrated with the internal axioms by adding them to the "Usable" list of the ATP system.

3 Concrete System Design

The availability of external axioms is specified in formulae of the form

```
fof(external_name,external,
   ∀∃template,
   external(type,access,protocol,[useful info]) ).
```

For example, the following specifies the availability of external axioms that relate creators to their creations.

```
fof(creators,external,
   ! [Thing] : ? [Name] : s__creatorOf(Name,Thing),
   external(exec,'DBServer CreatorDB',tptp_qa,
      [xdb(limit,number(1000)),xdb(limit,cpu(4))]) ).
```

The *external_name* is an arbitrary identifier for the external specification. The `external` role separates the external specification from other types of formulae, such as `axiom`s and `conjecture`s. The ∀∃*template* specifies the format of the external axioms. The atom is a template for the external axioms, and may contain structure including functions and constants. The universally quantified variables (`!` is the TPTP universal quantifier) must be instantiated when a request is sent, while the existentially quantified variables (`?` is the TPTP existential quantifier) can be filled in the external axioms that are delivered (see below). In the example, requests must instantiate the `Thing` for which the creators' `Name`s can be

[1] `http://www.tptp.org/TPTP/Proposals/AnswerExtraction.html`

provided in the axioms delivered. The universal quantifications can be thought of as a way of providing a large (potentially infinite) number of external specifications – an external specification for each combination of values for the universally quantified variables.

The `external()` term specifies the manifestation of the external axiom source, the communication protocol, and constraints on the batch delivery. The *type* specifies the nature of the source. At this stage only locally executed processes are supported, using their `stdin` and `stdout` for requests and delivery. This is specified as `exec`, as in the example. The *access* to the source depends on the *type*, e.g., for an `exec` it provides the command that will start the process (as in the example). The *protocol* specifies the format of requests and deliveries. At this stage only the TPTP question and answer protocol is supported, as described below. This is specified by `tptp_qa`, as in the example. The [*useful info*] is an optional Prolog-like list of terms, which can be used to store arbitrary information. In particular, `xdb()` terms are used to store auxiliary information that needs to be passed to the external source when requesting axioms.

A problem file may contain multiple external specifications, specifying multiple external sources, external sources with multiple ∀∃*template*s, or the same ∀∃*template* with different `xdb()` terms.

Requests for external axioms are written as TPTP "questions", of the form

```
fof(request_name,question,
   ∃template,
   source,[useful info] ).
```

For example, the following requests axioms for the creator(s) of YAGO.

```
fof(who,question,
   ? [Name] : s__creatorOf(Name,s__YAGOOntology),
   spass,[xdb(limit,number(1000)),xdb(limit,cpu(4))] ).
```

Most of the fields of a request correspond to those of an external specification. The `question` role specifies the intended use of this formula. The *source* is used generally in the TPTP to record where a formula came from. In the example, it records that the request came from the SPASS ATP system. The `xdb()` terms in the *useful info* are copied from the external specification, and thus passed on to the external source.

External axioms are delivered as TPTP "answers", which follow the SZS standards [3]. The complete delivery package is of the form

```
% SZS status Success for source
% SZS answers start InstantiatedFormulae for source
fof(external_name,answer,
   atom,
   answer_to(request_name,[useful info]),[useful info]).
fof(external_name,answer,
   atom,
   answer_to(request_name,[useful info]),[useful info]).
  ⋮
% SZS answers end InstantiatedFormulae for source
```

For example, the following delivers axioms for the creators of YAGO.

```
% SZS status Success for spass
% SZS answers start InstantiatedFormulae for spass
fof(creators,answer,
   s__creatorOf(s__FabianMartinSuchanek,s__YAGOOntology),
   answer_to(who,[]),[]).
fof(creators,answer,
   s__creatorOf(s__GjergjiKasneci,s__YAGOOntology),
   answer_to(who,[]),[]).
% SZS answers end InstantiatedFormulae for spass
```

Most of the fields of a delivery correspond to those of an external specification. The `answer` role specifies the intended use of this formula.

The algorithm for a CNF saturation-style ATP system, integrating asynchronous requests and delivery of external axioms, is shown in Figure 2.

```
1  foreach ES ∈ ExternalSources do
2      ES.WaitingForDelivery = False;
3      ES.RequestQueue = ∅;
4  end
5  while ¬SolutionFound & (Usable | *.WaitingForDelivery | *.RequestQueue) do
6      repeat
7          foreach ES ∈ ExternalSources do
8              if ES.WaitingForDelivery & Axioms = ES.CompleteDelivery then
9                  Add Axioms to Usable;
10                 ES.WaitingForDelivery = False;
11             end
12             if ¬ES.WaitingForDelivery & ES.RequestQueue then
13                 Send(Dequeue(ES.RequestQueue));
14                 ES.WaitingForDelivery = True;
15             end
16         end
17         if ¬Usable & *.WaitingForDelivery then sleep(1);
18     until Usable | ¬*.WaitingForDelivery ;
19     if ¬Usable then break;
20     Move ChosenClause from Usable to WorkedOff;
21     foreach NL ∈ ChosenClause.NegativeLiterals do
22         if NL matches any ES ∈ ExternalSources then
23             Build Request for NL;
24             if Request not repeated then Enqueue(ES.RequestQueue,Request);
25         end
26         Do inferencing with ChosenClause and WorkedOff;
27     end
28 end
```

Fig. 2. Integrating external sources into a saturation-style ATP system

The algorithm augments a standard ATP saturation loop algorithm with steps to accept deliveries, send requests, and queue requests. The loop control is modified accordingly. Specific points to note are:

- The condition of the **while** loop is augmented to keep going if more external axioms might be delivered.
- Each time around the **while** loop, each external source is checked for axioms delivered, and for requests to send. Only one request is sent to each external source at a time, so that there is no reliance on buffering in the communication channel.
- The request queue of each external source can be a priority queue.
- All negative literals (rather than, e.g., only the selected literal in ordered resolution) are examined for a match with an external source. This causes external axioms to be requested and delivered "preemptively". It would also be acceptable to simply ensure that at least one request is enqueued.
- The "matches" condition can be as precise as desired. The intention is to identify cases when external axioms might resolve against the negative literal.
- The "not repeated" condition can be as precise as desired, ranging from syntactic equivalence through to notions of unifiability and subsumption.

The soundness of the algorithm follows from its soundness without the steps for accessing external axioms. The notion of completeness is somewhat different in this setting, but if all the external axioms used were internal axioms, and were put in the same place in the Usable list as when delievered as external axioms, the same deductions would be performed. In practice, external sources might not deliver all possible axioms, and therefore a saturation cannot be taken to mean that there is no proof.

4 Implementation and Testing

The algorithm given in Figure 2 has been implemented as SPASS-XDB, by modifying the well-known SPASS ATP system [5].[2] The implementation is based on SPASS 3.5, which can read TPTP format data. Each external source of axioms is represented by a `struct`, containing:

- The execution string and PID of the external source process, which is started using a standard `fork` and `exec` sequence.
- Unnamed pipes for the `stdin` and `stdout` of the external source process.
- A flag indicating if a request has been sent, for which there has not been a delivery.
- A queue of requests waiting to be sent.
- A list of sent requests, used to prevent sending duplicate requests.

The queue of requests is implemented as a priority queue, ordered so that lighter requests with less variables get higher priority. The effect is to give priority to

[2] An implementation based on the E prover [2] is also planned.

requests that are likely to deliver less axioms, which curbs the ATP system's search space. SPASS' constraints on the search space are relaxed in various ways, to overcome various forms of incompleteness that stem from SPASS' ignorance of the availability of external axioms, which are added during the deduction (SPASS, like most (all?) ATP systems, was designed with the assumption that all formulae are in the problem file). For the types of problems that SPASS-XDB is designed to solve, relaxing the constraints does not have a very deleterious effect, because the proofs are typically quite shallow (in contrast to the deep proofs that are common in traditional applications of ATP).

Five external sources of axioms have been implemented. The YAGO knowledge base has provided a lot (around 14.5 million) of ground facts about the world. The facts were mostly mined from Wikipedia, and were exported in TPTP format with symbols being renamed and axioms transformed to match the SUMO ontology [1]. Two implementations have been written to serve YAGO axioms. The first is a Prolog implementation that reads YAGO axioms into its database, and delivers them based on unification with a request. The second is a Java interface to a MySQL database containing YAGO axioms, with a relation for each predicate. A request is converted into a `SELECT...WHERE` statement that extracts matching tuples, which are then delivered in TPTP format.

Two computational sources of axioms have been implemented in Prolog. The first implements evaluation of ground arithmetic expressions, thus providing arithmetic capabilities to SPASS-XDB (support for arithmetic is notoriously weak in ATP systems). The second implements syntactic difference of terms, which is used to determine if two terms look different. This provides a controlled way to implement a unique names assumption.

A web service that calls the Yahoo Maps Web Service[3] has been implemented via a Prolog mediator. This provides axioms containing the latitude, longitude, official city name, and country code, for a given common city name. The mediator converts a request to an HTTP request that is posted to Yahoo, and converts Yahoo's XML result to a TPTP axiom.

4.1 Testing

SPASS-XDB has been tested on several problems. One illustrative example is given here. The problem is to name an OECD country's capital that is at the same latitude (rounded to the nearest degree) as Moscow, and is subject to flooding. The names of the 27 OECD countries are retrieved from an external source of 4.5 million YAGO entity-property axioms. The capital of each country is retrieved from an external source of YAGO axioms. The YAGO identifier for a capital is translated to its common English name using an external source of YAGO axioms for "meaning in English" - this external source has 2.7 million axioms. Given the common English name of a city, it's latitude is obtained using the Yahoo Maps source of axioms, and the real values obtained from these axioms are rounded to the nearest degree using the external source of ground

[3] `http://developer.yahoo.com/maps/`

arithmetic evaluation. Internal SUMO axioms that specify that coastal cities are near the sea, that a sea is a body of water, a body of water is a water area, things near water areas can get flooded, and class membership is inherited, are used with an external source of YAGO axioms about which cities in the world are coastal, to establish if a city is subject to flooding. This aspect requires full first-order reasoning, rather than simple data retrieval. Finally, the external source of syntactic difference axioms is consulted to prevent Moscow being reported as the OECD capital.[4] In the proof search 84 requests are queued, 52 requests were sent, and 337 external axioms are delivered. The axioms delivered break down as 27 OECD countries, 15 capital cities, 6 common English city names, 4 latitudes, 3 arithmetic evaluations, 280 coastal cities, and 2 confirmations of different city names. The proof takes 11.2 CPU seconds, 10.9s in SPASS and 0.3s in external axiom sources. 4667 clauses were derived by SPASS.

5 Conclusion

This paper has presented the design and implementation of an ATP system that retrieves axioms from external sources, asynchronously, on demand. The design decisions were taken from an analysis of many choice points. The testing shows that interesting problems can be solved, using external sources of axioms that cannot be handled effectively as normal internal axioms in an ATP system.

There is evidence that automated reasoning for large theories is growing in importance. Current classical first-order ATP systems, such as those that compete in the annual CADE ATP system competition [4], are unable to work in extremely large theories. This work provides a step forward in ATP capability – even if access to external axioms is slower than in-memory access, waiting for a proof is better than no hope for a proof at all.

References

1. de Melo, G., Suchanek, F., Pease, A.: Integrating YAGO into the Suggested Upper Merged Ontology. In: Chung, S. (ed.) Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence, pp. 190–193 (2008)
2. Schulz, S.: E- A Brainiac Theorem Prover. AI Communications 15(2-3), 111–126 (2002)
3. Sutcliffe, G.: The SZS Ontologies for Automated Reasoning Software. In: Sutcliffe, G., Rudnicki, P. (eds.) Proceedings of the Workshop on Knowledge Exchange: Automated Provers and Proof Assistants, CEUR Workshop Proceedings, vol. 418, pp. 38–49 (2008)
4. Sutcliffe, G.: The 4th IJCAR Automated Theorem Proving System Competition - CASC. AI Communications 22(1), 59–72 (2009)
5. Weidenbach, C., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P., Dimova, D.: SPASS Version 3.5. In: Schmidt, R. (ed.) Proceedings of the 22nd International Conference on Automated Deduction. LNCS (LNAI), vol. 5663, pp. 140–145. Springer, Heidelberg (2009)

[4] The answer is Copenhagen, in case you are trying to work it out.