Capability Discovery for Automated Reasoning Systems*

Alexander Steen[1], Max Wisniewski[1], Hans-Jörg Schurr[13], and Christoph Benzmüller[21]

[1] Freie Universität Berlin, Berlin, Germany
[2] University of Luxembourg, Luxembourg
[3] TU Wien, Vienna, Austria

Abstract

Automated reasoning systems such as theorem provers often employ interaction or cooperation with further reasoning software. Whereas in most cases the concrete choice of cooperating software is, to some extent, irrelevant, these systems are nevertheless often rigid in practice due to compatibility issues. In order to support more flexible cooperation schemes, a machine-readable description format for automated reasoning systems' capabilities is proposed. Additionally, a simple HTTP-based protocol for system and capability discovery is outlined. Both the format and the protocol are designed to be simple, extensible and easy to use with none to minor modifications for existing reasoning systems.

1 Introduction

Automated reasoning systems such as theorem provers or model finders significantly differ in scope and supported language features. While projects such as TPTP [13] and SMT-LIB [2] provide a widely accepted infrastructure for automated reasoning systems within their domain, including standard language representations for input problems and generated outputs (e.g. proof objects), not all principally valid inputs are, in fact, supported by a given system and may thus lead to unexpected errors or results. For example, the current TPTP version supports seven different languages with more or less distinct grammatical representations. During the development of tools (called *meta tools* in the following) that employ reasoning systems as a subroutine, one faces the peculiarity of either fixing one system of choice, or handcrafting a list of supported systems together with their characteristics.

In order to increase the flexibility of an automated reasoning system's employment, we propose a machine-readable format which can be used to describe the capabilities of a reasoning system. Such capability information can then be used by meta tools to appropriately adjust their cooperation behavior according to available features at the target system. We give practically motivated examples for such capabilities. Furthermore, we outline a simple text-based protocol which can be used by meta tools for discovering both locally and remotely installed systems as well as for identifying their respective capabilities.

A protocol for the above scope of operation must meet certain criteria for being instrumental in practical scenarios. First, the protocol needs to be lean, easy to use and admit a fairly effective way of invoking reasoning systems. This way, a meta tool will not pay too much of a price actually using the protocol rather than hard-coding certain systems. Secondly, adaption of existing reasoning systems needs to be simple with none to minor changes. Otherwise no well-established system will, in the end, support the protocol. Finally, the protocol should ideally be extensible in the sense that new languages or language features can be expressed without changing existent capability descriptions.

*This work has been supported by the DFG under grant BE 2501/11-1 (Leo-III).

The availability of a capability discovery protocol is of special interest in the context of the authors' development of the higher-order (HO) automated theorem prover (ATP) Leo-III [11]. The Leo-III prover is designed to cooperate with first-order (FO) provers during proof search. In principle, it is of no importance which FO prover is employed (as long as it supports input and output according to TPTP/TSTP standard) and so the user may register further cooperating provers, e.g. via command-line arguments. Nevertheless, depending on the features of the used prover, the mode of cooperation might be different when e.g. only untyped first-order provers are given to Leo-III. Here, a capability discovery protocol could be used to identify which features the registered provers have. Alternatively, when no particular prover is suggested by the user, the protocol can be used to discover available systems and then automatically choose some of them.

Further HO ATPs which cooperate with external systems are LEO-II [3] and Satallax [5], which use FO provers and both SAT solvers and FO provers, respectively.

Related work. SystemOnTPTP [13] provides an online interface for using various automated reasoning systems. This interface is also capable of recommending systems based on their evaluation results wrt. the TPTP library. The 'System report' of SystemOnTPTP also includes a short (partially) machine readable description of the systems. One of the fields contained in this report is a list of *Specialist Problem Classes* (SPCs) supported by the system. SPCs are used to describe syntactic similar problems. SPCs include information about the supported languages of the system. However, SPC information are not shipped with the reasoning system but added externally as annotation, hence are not available when using a system outside of the SystemOnTPTP context. Additionally, they do not provide enough information about supported syntax features within a TPTP language.

The SPASS-XDB system [12] can use external sources to retrieve axioms. A TPTP-based description format is used to describe the external axiom sources.

Recent versions of the E prover include Deduction as a Service [8], which can potentially mediate between multiple provers, but it does not provide a way for clients to adapt to the target systems.

MathServe [14], which has emerged in the OMEGA project [1] from the earlier MathWeb system [15, 7], uses so-called brokers for finding appropriate reasoning systems for a given input problem wrt. its semantical aspects, e.g. finding systems that are marked as more specialized on certain problem features. However, it does not enumerate the supported language representation and syntax features of the respective language. Also, invocation of systems in MathServe using SOAP or XML-RPC seems rather verbose for local employment.

2 Capability Description Format

To enable the automatic detection of available features, theorem proving systems should provide a machine-readable description. This description can be provided either as a static file as part of the installation (e.g. a `.caps` file), or, preferable, as system output for a specific command line argument. Unix systems print customary a short help text when called with the `--help` command line argument. In this spirit, we suggest the argument `--caps` as switch to request the output of the capability description. To facilitate simple employment, we advocate for the use of the commonly used description language JSON [6]. JSON is a light-weight data-interchange format that is supported, natively or via library, by almost every programming language.

An automated reasoning system can be designed towards one or multiple applications. For instance, it might be a theorem prover or a model finder. A client has to know this intention

since e.g. a model finder might not be very useful for finding proofs and vice versa. Each supported input format, as well as the name and version of the system are features to be included in the description. Represented as JSON, this reads

```
{
  "name"         : "myprover",
  "version"      : 1.1,
  "applications" : [...],
  "languages"    : [...]
}
```

where possible `applications` include `"prover"`, `"model finder"` and `"proof verifier"`. The field `languages` contains a list of concrete language specifications.

A language specification object consists of a member denoting the language name and further members depending on the selected base language. In the case of TPTP languages two additional members are used: `roles` enumerating the supported formula roles and `features` enumerating supported syntax features within the language. While a white-list of supported features might seem overly verbose, it allows for extensions of the languages without any need to modify existing capability descriptions.

The following is an example output from a system that supports theorem proving on untyped and typed FO problems (represented via FOF and TFF syntax, respectively):

```
{
  "name"         : "myprover",
  "version"      : "1.1",
  "applications" : ["prover"],
  "languages"    : [{ "language" : "tptp_fof",
                      "roles"    : ["axiom", "definition", "conjecture"],
                      "features" : ["sequent"]
                    },
                    { "language" : "tptp_tff",
                      "roles"    : ["axiom", "definition", "type", "conjecture"],
                      "features" : ["let", "conditional", "polytypes"]
                    }]
}
```

From this description it is extractable that `myprover` supports TF1 [4], since `polytypes` is a feature of the system wrt. its TFF language capability. Of course, the concrete language features need to be interpreted relative to the language itself. Hence, in the above example, one needs to agree on a standard way of specifying the features of a TPTP language.

Further possible languages include `tptp_cnf` and `tptp_thf` for TPTP CNF format and TPTP higher-order form, respectively. In the case of `tptp_tff`, possible features are `sequent`, `tuple`, `polytypes`, `let`, and `conditional`. The `tptp_thf` language supports the features of `tptp_tff` and additionally `definite_description` and `indefinite_description`. Furthermore, the `tptp_thf` language might also contain support for TH1 combinators [9]. Also, appropriate features for the support of arithmetic in TPTP languages should be chosen.

3 System and Capability Discovery Protocol

The capability description format by itself is very helpful for implementing meta tools as it allows automated detection and seamless employment of a priori unknown systems. Both,

Method	Query address	Effect / Description
GET	`URL/`	Returns all reasoning systems known to the underlying discovery service
GET	`URL/<prover>`	Returns the capability description of `<prover>`, error code 404 if the prover is not known
GET	`URL/?language=X`	Returns the subset of reasoning systems that support language `X`
GET	`URL/?application=X`	Returns the subset of reasoning systems that support application type `X`
GET	`URL/?feature=X`	Returns the subset of reasoning systems that support feature `X` in some language

Table 1: The API specification for system and capability discovery

the capability description and the discovery protocol, are well separated. On one hand, the capability description format allows the clients (i.e. meta tools) to query the appropriate provers. On the other hand, the discovery protocol allows the clients to inspect available systems in the first place.

For the system and capability discovery protocol, we follow common API design patterns and suggest a RESTful API utilizing the standard HTTP protocol. Not only is this approach system independent, it is also well supported by programming languages and libraries. A small program, which provides the HTTP API, mediates the communication between clients and proving systems regardless whether they are located on the same machine or distributed across a network.

The protocol is given in Table 1. The discovery service must be reachable by a certain URL, denoted `URL`. Then, a GET request to `URL` with the respective arguments returns a prover description for all provers, a subset of provers or the capability description for a particular prover. For example, the GET request to `URL` without any parameters returns

```
[{
    "name"        : "myprover",
    "capabilites" : <capabilities of myprover>,
    "usage"       : [{ "protocol" : "cmd",
                       "address"  : "myprover -t %d %s" },
                     { "protocol" : "http",
                       "address"  : "http://myaddress.com/myprover?t=%d" },
                     { "protocol" : "systemontptp",
                       "address"  : "myprover" }]
  }, ... ]
```

The above list contains one object for each system, which consists of at least the members `name` and `capabilites` containing the system's name and its capability object, respectively. Furthermore, the description object should also provide one member for each supported way of interacting with the system, i.e. how the reasoning process is invoked.

We refrain from predetermining the possible ways to interact with the reasoning systems. A simple form of interaction would be calling the system manually with a given shell command (cf. `cmd`) or via a HTTP call (cf. `http`) where the proof problem is submitted e.g. via the POST method. In this example, the meta constants `%d` and `%s` represent the system timeout and the problem file, respectively. Furthermore, the system name in the SystemOnTPTP infrastructure could be presented. Should a theorem prover support deduction as a service [8], the description

object could contain a `deductionAsAService` member pointing to the address and port of the service.

Since the protocol supports invocation of locally installed reasoning systems, the usage of the discovery does not put a heavy burden on the actual system's usage.

4 Applications

There are at least two applications of the above sketched discovery service. Firstly, as mentioned before, automated HO ATPs such as Leo-III rely on cooperation with FO theorem provers. Thereby, Leo-III can choose between various possible encoding techniques to reduce a HO problem to a FO problem. When translating to a (polymorphically) typed FO form, encoding of FO types can be avoided. During cooperation with a prover for untyped FO logic, on the other hand, type encoding cannot be avoided. Furthermore, various grammatical language features, such as *let*-expressions, conditionals and choice might, or might not, be supported by the target system. For systems that do not support some language constructs, these features need to be encoded as well, possibly resulting in performance loss.

Secondly, a capability discovery protocol could be used by interactive theorem provers that allow discharging of proof tasks using automated reasoning systems. For example, in Isabelle/HOL [10], the Sledgehammer tool invokes automated theorem provers depending on their supported target logic and language features. Currently, Sledgehammer relies on a hard coded database of supported systems for selecting the appropriate problem encoding scheme.

Hard coding specific provers should be avoided, and rather be replaced by support of a whole class of provers with the specific encoding scheme parameterized by the prover's capabilities. Then, if a description of a prover's features can be accessed via a discovery service, also unknown reasoning systems can be employed for cooperation.

5 Summary and Further Work

In this paper, a description of a machine-readable format for describing capabilities of automated reasoning systems is given. Furthermore, a simple protocol is presented that allows clients to discover available reasoning systems and their capabilities. Such a protocol could not only be useful for the ATP community, but in particular, it could foster flexible integrations of theorem proving technology in e.g. larger AI systems such as IBM Watson, etc.

Static capability description files for the LEO-II and Leo-III prover can be accessed at the Leo-III project website.[1] Also, as of version 1.1 the Leo-III prover supports the `--caps` switch to print its capability description. A reference implementation of a discovery service is work-in-progress. Further work includes a thorough evaluation of a practical employment scenario. Also, further discussions about the invocation schemes and presented prover capabilites need to be carried out. The capability format itself could be enhanced or modified such that it enhances the already existing SPC information in the context of TPTP languages.

References

[1] Serge Autexier, Christoph Benzmüller, Dominik Dietrich, and Jörg Siekmann. OMEGA: Resource-adaptive processes in an automated reasoning systems. In Matthew W. Crocker and Jörg Siek-

[1] See http://inf.fu-berlin.de/~lex/leo3/#downloads.

mann, editors, *Resource-Adaptive Cognitive Processes*, Cognitive Technologies, pages 389–423. Springer, 2010.

[2] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[3] Christoph Benzmüller, Lawrence C. Paulson, Nik Sultana, and Frank Theiß. The Higher-Order Prover LEO-II. *Journal of Automated Reasoning*, 55(4):389–404, 2015.

[4] Jasmin Christian Blanchette and Andrei Paskevich. TFF1: The TPTP Typed First-Order Form with Rank-1 Polymorphism. In Maria Paola Bonacina, editor, *Automated Deduction, 24th International Conference, Proceedings*, volume 7898 of *LNCS*, pages 414–420. Springer, 2013.

[5] Chad E. Brown. Satallax: An automatic higher-order prover. In *Automated Reasoning*, volume 7364 of *LNCS*, pages 111–117. Springer Berlin Heidelberg, 2012.

[6] Ecma International. The JSON Data Interchange Format, 2013. Standard ECMA-404.

[7] Andreas Franke, Stephan M. Hess, Christoph G. Jung, Michael Kohlhase, and Volker Sorge. Agent-oriented integration of distributed mathematical services. *J. UCS*, 5(3):156–187, 1999.

[8] Mohamed Hassona and Stephan Schulz. Deduction as a Service. In *Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning*, volume 1635 of *CEUR Workshop Proceedings*, pages 32–40. CEUR-WS.org, 2016.

[9] Cezary Kaliszyk, Geoff Sutcliffe, and Florian Rabe. TH1: the TPTP typed higher-order form with rank-1 polymorphism. In Pascal Fontaine, Stephan Schulz, and Josef Urban, editors, *Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning*, volume 1635 of *CEUR Workshop Proceedings*, pages 41–55. CEUR-WS.org, 2016.

[10] Tobias Nipkow, Lawrence C. Paulson, and Makarius Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Lecture Notes in Computer Science. Springer, 2002.

[11] Alexander Steen, Max Wisniewski, and Christoph Benzmüller. Agent-based HOL reasoning. In G.-M. Greuel, T. Koch, P. Paule, and A. Sommese, editors, *Mathematical Software, 5th International Congress, Proceedings*, volume 9725 of *LNCS*, pages 75–81, Berlin, Germany, 2016. Springer.

[12] Martin Suda, Geoff Sutcliffe, Patrick Wischnewski, Manuel Lamotte-Schubert, and Gerard de Melo. External sources of axioms in automated theorem proving. In *KI 2009: Advances in Artificial Intelligence, 32nd Annual German Conference on AI, Paderborn, Germany, September 15-18, 2009. Proceedings*, pages 281–288, 2009.

[13] Geoff Sutcliffe. The TPTP world - infrastructure for automated reasoning. In *Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LPAR'10, pages 1–12, Berlin, Heidelberg, 2010. Springer-Verlag.

[14] Jürgen Zimmer and Serge Autexier. The MathServe System for Semantic Web Reasoning Services. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning: Third International Joint Conference, Proceedings*, pages 140–144. Springer Berlin Heidelberg, 2006.

[15] Jürgen Zimmer and Michael Kohlhase. System description: The mathweb software bus for distributed mathematical reasoning. In Andrei Voronkov, editor, *Automated Deduction - CADE-18, 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27-30, 2002, Proceedings*, volume 2392 of *Lecture Notes in Computer Science*, pages 139–143. Springer, 2002.