Reasoning Specialists Should Be Logical Services, Not Black Boxes

Carolyn Talcott
Stanford University
clt@sail.stanford.edu

1. Introduction

An important problem for automated reasoning is to develop mechanisms for integration of special purpose reasoning systems into general purpose reasoning systems, and into other systems. In this abstract we first point out some of the problems to be solved in order to have reasoning modules that can be effectively integrated, and suggest the notion of a logical service as a way of viewing integrable reasoning modules. We then present some preliminary results from ongoing work aimed at gaining a better understanding of this notion. Finally, we illustrate some of the ideas by describing a series of reasoners specializing in linear arithmetic using the concepts developed.

What sort of reasoning components can usefully plug and play? A yes/no type black box decider is not adequate in general. In order for reasoning modules to be usefully integrated they must be able to interact with other components in non-trivial ways. Such modules may need to accept information incrementally and produce results incrementally. They may need to ask for and use information about entities not in their domain of expertise. For example, it may not be good enough to have a decision procedure that can deal with "uninterpreted" function symbols. Useful interaction may require recognition that these function symbols are not uninterpreted, and use of facts about terms involving them.

A general framework is needed for structuring and specifying integrable reasoning modules. Both logical semantics and interaction capabilities and protocols must be specified. Deduction must be treated as an incremental, and interactive process. An integrable module needs wrappings that specify both when, how, and for what purpose it can or should be called; and what services or information it needs or could use. Following [6], we say a reasoning module packaged for integration provides a *logical service*. What is a logical service? We will not presume to give a definitive answer, but rather ask more questions. What features are necessary to interconnect or integrate reasoning devices in useful and semantically meaningful ways? What kind of information must be specified for such a service? How are their interactions and interconnections to be understood? What are the basic operations for composition and integration? The work on reasoning specialists and handles described in the next section provides some partial answers to these questions.

2. Reasoning Specialists and Handles

A *reasoning specialist* is a provider of a logical service. It exists in some implicit context, for example a logic together with a theory within that logic (cf. [4]). It provides access to (a view of) local contexts, for example collections of additional facts or assumptions valid at some point in a formula. The information contained in these local contexts is represented by objects we call *handles*. We take the view that logical services are organized into classes corresponding to kinds of global context. Logical service classes provide the underlying logical semantics. They also determine the kind of expertise provided and the collection of meaningful operations and interactions. A *reasoning specialist* is then an instance of a logical service class.

In the following we sketch some general properties of and operations on handles. We let h, h_1, ...ranges over handles representing local contexts of some fixed but unspecified logical service class.

Ordering on handles. There is a pre-order $\preceq$ on handles. $h_0 \preceq h_1$ means that h_1 is reachable from h_0 by adding new information. $\preceq$ is transitive and reflexive, and $h_0 \cong h_1$ abbreviates $h_0 \preceq h_1 \wedge h_1 \preceq h_0$. `mtH` is the minimal handle: `mtH` $\preceq h$, for any h of the same class. `mtH` represents the empty local context, and contains only information valid for all local contexts of the class. `topH` represents an inconsistent context: $h \preceq$ `topH` for any h. There may be no such handle in some classes – if the language/logic is too weak to express or derive inconsistencies (for example PROLOG).

Telling and Asking. Two important families of operations on handles are telling facts and asking questions.

$tell(h, f) \rightarrow h'$ – telling a handle h a fact f gives new handle h' representing the context of h with the additional information contained in f. Thus $h \preceq tell(h, f)$. In a resource sensitive logic $tell(tell(h, f), f)$ is not necessarily equivalent to $tell(h, f)$. A fact can be simply a formula or set of formulae. It may also carry procedural information or simply introduce entities for consideration.

$ask(h, q) \rightarrow r$ – asking a handle h a question q gives response r. Possible questions include:

(0) Is the information embodied in h consistent?

(1) Is formula f satisfiable/unsatisfiable?

(2) Is formula f entailed?

(3) What is the preferred interpretation / rewriting of term t?

(4) Give a solution (all solutions) to a set of constraints.

The response might be a proper answer, 'I don't know', a request for additional information, or a suggestion of a new question to ask. A response might include a new handle that represents progress towards finding an answer, for example some additional deductions made. The answer domain for questions of the form (1) or (2) could be binary (yes or unknown) or ternary (yes, no, or unknown). In the case of a ternary answer domain for entailment questions: yes means (the context represented by) h

entails f; `no` means h does not entail f; and `unk` means h can not determine whether or not f is entailed. $ask(tell(h, f), entails(f)) =$ `yes`. When a handle is told a fact or asked a question, it may ask questions itself, or interact with auxiliary handles.

A class of handles might supply only one quality of service, or there may be many levels. Arguments to ask and tell operations may supply additional information to specify the quality of service required. If the response is another handle representing a partial solution or progress towards obtaining an answer, then asking the resulting handle, perhaps requesting a higher level service, may produce a proper answer. In the context of levels of service, there is another kind of partial ordering on handles that is of interest to explore: $h_0 \sqsubseteq h_1$ if h_1 is obtained from h_0 by making more implicit information explicit – i.e. by doing some processing of the information h_0 has been told so that less work is required to obtain answers to questions.

Another useful scenario is that of announcing new information to a handle and getting back new discoveries in reply. This can be represented asking questions of the form "what new facts are entailed?" of a handle returned by a tell operation.

Extracting. Some classes may support an operation for extracting the local context represented.

- $extract(h) \rightarrow c$ – an external representation of the context represented by h.

For purposes such as archiving it is useful for extraction to satisfy the following:

If h was obtained by telling some set of facts, then $tell($`mtH`$, extract(h)) \cong h$.

Composing handles. Specialists may use handles from more than one class, and we want to be able to form composite services and handles. Here we will only indicate some of the issues that arise. One is the logical structure of a class. For example we can ask what it means to have operations such as negation, conjunction, disjunction, or quantification, on handles, and whether or not a particular class supports any given logical operation. Another issue is moving between classes and combining handles of different classes. This will require more development of notions of mappings between theories (cf. [4, 2]) to apply not only to theories, but to services.

3. Linear Arithmetic Specialists

As a more concrete illustration of our ideas, let us look at a spectrum of linear arithmetic specialists. In an expanded version of this abstract we give some examples of specialists dealing with equivalence and rewriting relations, and treat the cooperating decision procedures paradigm of [5].

The basic problem is to decide satisfiability or unsatisfiability of sets of literals in the language of linear arithmetic (say over the integers). We first describe three black-box varieties of specialist. Then we look at some of the features of the linear arithmetic module of the Boyer-Moore prover [1] as a an example of a black box opened up.

3.1. Black-box Linear Arithmetic Specialists

The implicit context is the first-order theory of $\{+, =, \leq\}$ over the integers. Let Lit_{LA} be the set of literals (atomic formulas and negations of atomic formulas) of this theory.

The specialist LA_0 decides satisfiability for finite sets from Lit_{LA}. There is no capacity for building up local context, i.e. there are no *tell* operations, and the only handle is $\texttt{mtH}_{LA}$. A request to LA_0 to decide C corresponds to $ask(\texttt{mtH}_{LA}, satisfiable(C))$. Answers are S, for satisfiable, or U, for unsatisfiable.

The specialist LA_1 has handles representing finite sets C from Lit_{LA}. If h represents C, then $tell(h, C') \rightarrow h'$ where h' represents $C \cup C'$. $ask(h, consistent?)$ gives S if the literal set represented by h is satisfiable, and U if it is unsatisfiable. $extract(h) \rightarrow C$ where C is (equivalent to) the set of literals represented by h.

Note that handles hide the internal representation of local contexts. A possible representation in the case of linear arithmetic would be as polynomials, using some standard algorithm to decide satisfiability.

The specialist LA_2 has as implicit context that of LA_1 extended with additional function symbols. LA_2 provides the same service as LA_1 over the extended language, taking the additional function symbols to be uninterpreted, i.e. treating terms containing these symbols as variables. To fill in the specification of LA_2 we must say whether the universe still contains only numbers or whether it too is extended.

3.2. Linear Arithmetic in the Boyer-Moore Prover

A further enhancement of LA_2 is to provide the capacity to ask for or use facts about terms containing non-LA symbols. To describe such a service we need to say more about what facts it might find useful, and how it is provided access to such facts. As a concrete example of this and other aspects of integrated reasoning specialists, we describe some features of the linear arithmetic module of the Boyer-Moore prover, NQTHM. A detailed and enlightening description of the integration of this module is given in [1].

There are two basic specialists, one dealing with typeset information and the other with polynomials. Typeset reasoning is fundamental to the operation of NQTHM. The data structures denoted by terms of the NQTHM language are organized into disjoint types (shells), each with an associated recognizer (characteristic function). Typeset information handles (TI-handles) represent the typeset information obtained by assuming a set of literals. The typeset specialist uses this information to determine the possible types of a term in the context of these assumptions.

- $tell(ti, lit) \rightarrow ti'$ encoding the additional typeset information implied by *lit*.
- $ask(ti, t)$ gives a set of types that is an upper bound on the possible types of t in the context embodied in ti.

The polynomial information handles (PI-handles) provides access to a polynomial database.

tell(pi, $poly$) $\rightarrow$ pi' obtained by adding to the database represented by pi, $poly$ and additional polynomials obtained by basic polynomial reasoning (such as cross-multiply and add).

ask(pi, *consistent*?) gives a `yes` answer or an impossible polynomial.

There is additional information encoded in a polynomial database, and there are several additional operations needed to support full integration of linear arithmetic reasoning. Polynomials contain not only a mathematical polynomial inequation, but also a field called linear-hyps, and a history giving the facts from which the polynomial inequation was derived. The linear-hyps are assumptions needed for the inequation to be valid. Linear-hyps and histories may contain holes (schematic variables). Manipulation of polynomials by a PI-handle is uniform in these variables and there is a further operation on PI-handles to fill such holes. (This is used to add polynomials before the linear-hyps and derivation history is completely known.) A PI-handle can be asked what terms it needs information about. Finally, there is a hiding operation on PI-handles. Hiding a fact produces a PI-handle containing only polynomials that do not have that fact as part of their derivation history.

The specialist LA accepts requests to add a set of literals to a PI-handle using typeset information embodied in a TI-handle. The result is a new PI-handle. The assumption is that the PI-handle and TI-handle correspond to information extracted from (essentially) the same set of literals. LA uses the TI-handle to linearize the literals, and tells the resulting polynomials to PI-handle. In the process, the LA specialist asks the PI-handle for terms about which information is needed, and tries to obtain useful information about these terms. Thus the LA specialist needs to be provided access to a source of additional information. In the case of NQTHM, this is a database of linear rules. Since these rules are conditional, additional reasoning is required to validate the application of the rule. Thus LA needs access to some reasoner. This reasoner will be most useful if it can make use of the information contained in the PI-handle and TI-handle. In the case of NQTHM, LA invokes the rewriter (which may in turn invoke LA).

4. Concluding remarks

The work on reasoning specialists and handles described above is part of a larger project to develop an open architecture for mechanized reasoning systems. We want an architecture that supports development of mechanisms for interoperation and integration of disparate reasoning systems: based on different logics; having different domain models; or using different vocabularies, representations of information, and reasoning strategies. The dream is to be able to compose reasoning systems from basic modules, to add new modules to existing systems, to form composite systems that support complex reasoning problems, and to embed reasoning devices in other systems, all in a plug and play manner. Another part of this project [3] develops the concept of reasoning structures and reasoning theories to address the problem of providing a framework for structuring reasoning modules and specifying their deductive and interactive capability.

Acknowledgements. The author would like to thank Jussi Ketonen, Ian Mason, Jos Marlowe, Ruth Davis, David Cyrluk, Fausto Giunchiglia, and Paolo Pecchiari for much useful feedback during the course of this work.

This work was partially supported by ARPA/NASA contract NAG2-703.

5. References

[1] R. S. Boyer and Moore. J. S. Integrating decision procedures into heuristic theorem provers: A case study with linear arithmetic. In *Machine Intelligence 11*. Oxford University Press, 1988.

[2] W. M. Farmer, J. D. Guttman, and F. J. Thayer. Little theories. In D. Kapur, editor, *Automated Deduction—CADE-11*, volume 607 of *Lecture Notes in Computer Science*, pages 567–581. Springer-Verlag, 1992.

[3] F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning structures: An architecture for open mechanized reasoning systems, 1994. in preparation.

[4] José Meseguer. General logics. In H.-D. Ebbinghaus et al., editor, *Logic Colloquium'87*, pages 275–329. North-Holland, 1989.

[5] G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2), October 1979.

[6] Ian Sutherland and Richard Platek. A plea for logical infrastructure. In *TTCP XTP-1 Workshop on Effective Use of Automated Reasoning Technology in System Development*, pages 1–3, 1992.