Modeling amidst the Microtheories

Kenneth D. Forbus

Qualitative Reasoning Group, Northwestern University
2133 Sheridan Road, Evanston, IL, 60201, USA
forbus@northwestern.edu

Abstract

Integrating qualitative reasoning with large-scale knowledge bases provides new challenges. This paper outlines work in progress on developing a new model formulation system to support qualitative reasoning via compositional modeling that can operate in an environment with over a million available facts. Three ideas are discussed: Exploiting microtheories for modeling, using non-monotonic inference, and speculative inference. An implemented algorithm which runs on QP benchmark examples is outlined, and planned extensions are discussed.

Introduction

Most qualitative reasoning systems have been built as stand-alone pieces of software, as systems or modules which assume a specialized input language for models and/or domain theories. Today, the existence of large-scale knowledge bases such as OpenCyc, and the potential for building even larger knowledge bases via the Semantic Web, provide a new set of opportunities and challenges for qualitative reasoning. The opportunities include the ability to explore more directly how qualitative reasoning can be used in common sense reasoning. Common sense reasoning, after all, involves tying qualitative reasoning into the wide expanse of everyday knowledge of kitchens, swimming pools, parking lots, and cactus. The challenges include the need to keep model formulation tractable in the face of millions of potentially relevant facts. Reasoning techniques which have commonly been used in the past, such as using an ATMS (Falkenhainer & Forbus, 1991) or assuming that the structure of domain theories is highly constrained (Nayak, 1994), do not seem likely to scale.

This paper describes work in progress on a new model formulation algorithm that is implemented as a service within the context of a reasoning system that uses a large-scale knowledge base. First the relevant properties of the knowledge base, reasoning system, and compositional modeling are reviewed. Next the key ideas of the approach are laid out, illustrated by examples. Finally, future work is discussed.

Background

Large-scale knowledge bases. We use the Cyc ontology[1] and KB contents in our research, augmented by an ontology for QP theory (Forbus, 1984) and support for analogical reasoning. Concepts are modeled in the KB as *collections*, which are linked into a hierarchy by the `genls` relation. The OpenCyc KB contents we are using, for example, include over 58,000 collections. A large number of predicates are defined (>14,000), whose argument signatures are specified in terms of these collections. These concepts include many concepts that are directly relevant to traditional qualitative models (e.g., substances, types of quantities) as well as many more concepts which are not obviously relevant (e.g., comic book characters, social relationships, mental states). Thus it provides a good example of an off-the-shelf knowledge resource which can be harnessed for qualitative reasoning.

The contents of the knowledge base are partitioned into *microtheories*. Microtheories provide a way of dealing with context. For example, people are quite capable of answering questions and making predictions about the fictional worlds of TV series and novels, while at the same time knowing that they do not exist. Microtheories provide a means of dealing with such inconsistent contexts (e.g., the `MiddleEarthMt` and `TeletubbiesMt` microtheories in OpenCyc). Every fact is in at least one microtheory. Microtheories themselves are related by `genlMt`, i.e., `(genlMt AMt BMt)` indicates that every fact believed in `BMt` is available in `AMt`. `genlMt` statements themselves are global, i.e. believed in every microtheory.

The *logical environment* of any operation consists of the microtheory it occurs in plus the set of microtheories that can be reached from it via `genlMt` relationships. This can provide considerable filtering: For example, the logical

[1] http://research.cyc.com

environment used in the experiments describe here includes only 640,452 facts out of the 1,722,715 facts currently available in the knowledge base.

Reasoning system. While we use the contents of OpenCyc or ResearchCyc, depending on the project, we use our own reasoning engine instead of Cycorp's. There are a variety of reasons for this: Ours is optimized for our purposes, and we have full source code access, for example. Our FIRE reasoning system implements the knowledge base via a persistent-object database[2]. The KB includes a form of TMS that enables efficient pattern matching retrieval while respecting logical environments. The working memory uses a logic-based TMS for propositional reasoning. FIRE has five primary reasoning mechanisms, but for our purposes only two are relevant: `Ask` provides access to the knowledge base and performs "simple" inferences, using procedural attachments to predicates to implement specialized computations. `Query` performs backchaining, using Horn clause axioms selected from the KB based on the current logical environment. Thus the logical environment of a computation can be used to filter both the available data and the available rules to operate over this data.

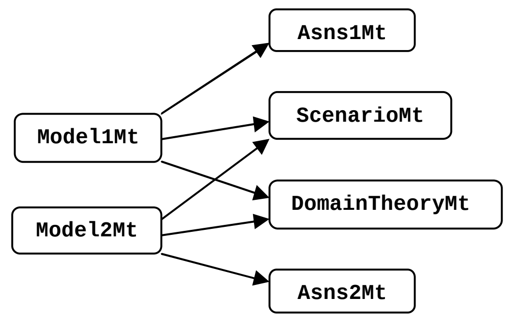

Figure 1: Microtheory scheme for model formulation Two distinct models (`Model1Mt`, `Model2Mt`) are based on the same domain theory and scenario, but different modeling assumptions (`Asns1Mt`, `Asns2Mt`). The arrows indicate `genlMt` statements.

Compositional Modeling. Knowledge about a domain is encoded in *model fragments*, which are logically quantified pieces of knowledge – think of frames or schemas – which are instantiated in the process of building models to reason about a specific *scenario*. Assembling a model from fragments is typically constrained by a task, often consisting of a quantity and some form of query about it. A *domain theory* consists of a set of model fragments, and often includes *assumption classes*. Assumption classes provide a mutually exclusive and collectively exhaustive set of choices for how to model something, when certain relevance conditions hold. Domain theories can consist of models at multiple levels of granularity and multiple, mutually inconsistent perspectives, if they rely on assumption classes to keep incoherent models from being constructed. Consequently, assembling a model given a scenario and a task is potentially quite complicated.

[2] Franz, Inc.'s Allegrocache database.

Microtheories and Modeling

Microtheories make both building compositional domain theories and performing model formulation easier in some ways. Domain theories can be stored in microtheories, or as a whole graph of microtheories, potentially decomposing them into modules that could be combined to create logical environments that contained all and only the knowledge needed for particular tasks. Scenario descriptions can be stored as microtheories as well, along with models built from them for particular purposes. Figure 1 illustrates.

The reasoning for model formulation is conducted in the logical environment of microtheory which will contain that model (e.g., `Model1Mt`). In the example of Figure 1, all of the model fragments and knowledge available from `DomainTheoryMt`, `ScenarioMt`, and `Asns1Mt` will be accessible in creating the model in `Model1Mt`. The alternate model being built in `Model2Mt` will be using most of the same knowledge, but the assumptions in `Asns2Mt` instead of `Asns1Mt`.

This scheme makes it straightforward to compute and compare multiple models for the same scenario without interference. However, there are subtleties. The point of using a large-scale knowledge base is to have access to a lot of knowledge, so domain theory microtheories typically will inherit from other microtheories to supply background knowledge. Consider this model fragment, from a benchmark QP domain theory:

```
ContainedStuffPossibility:
 Participants:
  containerOf: ?can, Container
  phaseOf: ?phase, MatterTypeByPhysicalState
  substanceOf:?sub,
            ChemicalCompoundTypeByChemicalSpecies
 Constraints: None.
 Conditions:
   (canContainSubstance ?can ?phase ?sub)
 Consequences:
   (hasQuantity ?self
       (AmountOfFn ?sub ?phase ?can))
   (qpGreaterOrEqualTo
       (AmountOfFn ?sub ?phase ?can) Zero)
```

The OpenCyc KB has six types of matter (solid, liquid, gas, plasma, Fermonic and Bose/Einstein condensate) and knows about 590 chemical compounds. Thus for every instance of a container in a scenario, unless more careful control is exerted, there will be 3,540 model fragments of this type instantiated. Reorganizing the microtheory contents is not a practical solution even in a curated knowledge base: There are 2,836 non-trivial microtheories in OpenCyc, for instance. Moreover, the point of the Semantic Web is to dynamically exploit existing resources from multiple sources, which makes reorganization impossible. Model formulation algorithms themselves must provide ways to stay focused.

Our solution to this problem is to expand the idea of `consider` assumptions from compositional modeling to include a new predicate, `considerEntity`, a unary predicate indicating that its argument represents an entity that should be included in the model under construction. The set of `considerEntity` assertions derivable within a logical environment thus determines what entities will be allowed into the model. As will be seen shortly, `considerEntity` statements are never introduced directly by modelers. Instead, they are inferred from a lower-level statement, in order to support non-monotonic reasoning.

Non-monotonic reasoning

Sometimes modeling knowledge concerns what should be ignored (e.g., evaporation while drinking coffee) rather than what should be considered. This needs to be handled at the level of entities, model fragments, and particular instances of model fragments. We discuss each in turn.

Suppressing entities. Scenarios sometimes contain entities that are irrelevant for a particular analysis, so there must be some means for suppressing the consideration of entities. The unary predicate `ignoreEntity` indicates that its argument represents an entity that must not be included in a model. It is a contradiction for `considerEntity` and `ignoreEntity` to be true of the same entity within a logical environment. Consequently, the derivation of `considerEntity` rests on a more primitive statement, `includeEntity`, and a non-monotonic derivation as follows:

```
(<== (considerEntity ?e)
     (includeEntity ?e)
     (uninferredSentence (ignoreEntity ?e)))
```

In FIRE, the predicate `uninferredSentence` is true only if its argument cannot be derived within the current logical environment. Thus logical contradictions are avoided, and default inclusion can be overridden by an inference that something should be ignored.

Suppressing model fragments. In cases where an entire type of phenomena should be ignored (e.g. ignoring thermal properties means suppressing heat flow and other processes that involve heat and temperature), the unary predicates `considerMF` and `ignoreMF` are defined and used analogously to `considerEntity` and `ignoreEntity`, with `ignoreMF` trumping `considerMF`. Such queries are used during model formulation when deriving the list of model fragments to look for.

Suppressing model fragment instances. Sometimes even finer-grained control is needed in building a model. For example, when considering where smoke goes if we burn something while cooking, we may need to include the air in the room in our model, but at the same time we may want to ignore transfers of heat to the air, while considering transfers of heat from the burners of the stove to the pots and pans on them. The binary predicates `considerMFInstance` and `ignoreMFInstance` provide this level of control. The first argument is a model fragment type (e.g. `HeatFlowProcess`), and the second argument is a binding list of participants for a proposed instance of that process. As with the other non-montonic predicates, ignoring trumps considering:

```
(<== (considerMFInstance ?mft ?bindings)
     (uninferredSentence
       (ignoreMFInstance ?mft ?bindings))

(<== (ignoreMFInstance ?mf ?given-bindings)
     (ignoreMFInstance ?mf ?other-bindings)
     (subsetOfBindings ?other-bindings
                       ?given-bindings))
```

Notice that the second rule allows concise constraints like "ignore heat flow to the atmosphere" to be expressed, since only the binding of the destination of heat flow to the atmosphere needs to be included. The test for `considerMFInstance` is used as the last step just before a model fragment instance is created, so that there is the most information available upon which to base the decision. As with the other consider/ignore pairs, this pattern enables the declarative specification of rules for ignoring phenomena, but here, the properties of the proposed binding list can be used to rule out instances involving specific entities.

Speculative Inference

QR systems have tended to treat model formulation as a distinct phase of reasoning from deriving conclusions with a model, because most qualitative reasoning requires closed-world assumptions about the relevant phenomena in order to construct the appropriate network of constraints. For example, boiling is only possible when a contained liquid is present. To determine whether boiling might occur in a pot requires determining that water could be in that pot. Thus all of the model fragments that will go into a model, whether or not they hold in the initial state(s), must be derived up front. In Gizmo and QPE, two implementations of QP theory, this was done by antecedent rules which triggered on facts simply being in the database, whether or not they were believed. This is essentially a form of speculative inference, where mentioning a fact is used as a heuristic that it might become true. This solution does not scale well because such antecedent rules have indefinite temporal extent, which means they will continue to clog memory even when they are no longer relevant. Consequently, we have developed a different approach.

For every microtheory *M* representing a model under construction, a new microtheory is defined:

```
(ModelFormulationScratchpadMtFn M)
```

This microtheory has *M* as its sole `genlMt`, and is used as a scratchpad by asserting as true every potential consequence and condition of every model fragment instance that gets created. For example, if `mf-0` is an instance of `ContainedStuffPossibility`, then the appropriate `hasQuantity`, `qpGreaterThanOrEqualTo`, and `canContainSubstance` statements will be believed to be true in the scratchpad. Given that processes often come in opponent pairs, this scratchpad will quickly become contradictory. That doesn't matter, since FIRE does not aggressively seek out contradictions by default – a choice made because of the inefficiency of complete reasoning. This gives us exactly the kind of speculative reasoning we want: By using this scratchpad for model formulation queries, we will get answers based on the possibility of propositions being true. In general this will lead to over-generation, because combinations of facts believed in the scratchpad may be mutually incompatible in any consistent logical environment. This is a relatively small price to pay, given the alternative of having to reconsider possible new model fragments after essentially every reasoning step.

Algorithm

We have combined these ideas to create a model formulation algorithm for QP models that can operate over large-scale knowledge bases. The basic model formulation algorithm is:

1. Gather relevant model fragment types, by querying for `considerMF`.
2. Sort model fragment types according to dependency
    a. If `MFa` introduces a statement that unifies with a participant constraint of `MFb`, then `MFb` depends on `MFa`, and so instances of `MFa` should be sought before instances of `MFb`.
3. For each relevant model fragment type, find instances by
    a. Find participants by searching for relevant entities (i.e., those that satisfy `considerEntity`) which satisfy the participant constraints, using speculative inference.
    b. Query for `considerMFInstance`. If not true, ignore this instance.
    c. Otherwise, instantiate the model fragment instance, asserting appropriate logic in the model microtheory, and asserting as true the consequences, conditions, and participant constraints in the scratchpad microtheory.

The dependency sort in Step 2 is an optimization made possible by speculative inference. Again, it can overgenerate, but no more so than prior antecedent-rule implementations. Experience with prior implementations indicates that this is typically a small price to pay.

At present the basic algorithm described above has been implemented, and operates correctly on the standard benchmark examples for QP theory implementations. We are planning to use it to replace the application-specific model formulation algorithms that we have used in our Companion experiments (Forbus, Klenk, & Hinrichs, 2009) and with CogSketch (Wetzel & Forbus, 2009).

Discussion

Integrating qualitative reasoning into large-scale knowledge-based systems provides interesting new challenges. For example, as we have seen, model formulation must be very carefully controlled to maintain focus. Such systems also provide new opportunities via new resources, including support for microtheories to provide a notion of logical environment and non-monotonic reasoning to support default reasoning with overrides. These facilities combined with using microtheories to make a scratchpad for the kind of speculative inference needed in model formulation, provide a robust foundation for a simple and elegant model formulation algorithm.

While the basic algorithm above is already sufficient for supporting our recent and current projects, it does not capture most of the more sophisticated model formulation ideas explored in the 1990s. Those ideas were, in some sense, ahead of their time, since they presumed efforts to build large-scale domain theories. With the widespread availability of OpenCyc and the rise of the Semantic Web, it is time to revisit those ideas and rework them to fit our new knowledge-rich environment. This paper represents the first step in that process.

There are several directions to explore next. First, we plan to add support for assumption classes, to allow reasoning with multiple perspective and multiple granularity domain theories. We suspect that the criteria for evaluating model quality used previously were somewhat oversimplified, since they were one-dimensional (fewest assumptions (Falkenhainer & Forbus 1991), lowest cost assumptions (Nayak 1994), single direct influence (Rickel & Porter, 1997). Types of available data, desired form of answers, reasoning cost, and explanatory clarity all seem like relevant factors, whose relative importance will vary with task. The more extensive ontologies in large knowledge-based systems provide potentially useful infrastructure for this. Accumulating and reusing modeling assumptions from experience (Falkenhainer, 1992; Klenk et al 2008) provides another source of reasoning we plan to incorporate. To given these ideas a more complete test, we are planning to rebuild the large-scale steam plant model (Falkenhainer & Forbus, 1991) to test reasoning with assumption classes, and to extend the reasoning to include time-scale abstraction (Rickel & Porter, 1997), using a port of their Botany Knowledge Base for testing.

Acknowledgements

This work is supported by the Intelligent and Autnomous Systems Program and the Cognitive Science Program of the Office of Naval Research.

References

Falkenhainer, B. 1992. Modeling without amnesia: Making experience-sanctioned approximations. *Proceedings of QR02*

Falkenhainer, B. and Forbus, K. 1991. Compositional modeling: finding the right model for the job. *Artificial Intelligence* 51:95–143.

Forbus, K. 1984. Qualitative Process Theory *Artificial Intelligence* (24)85-168

Forbus, K., Klenk, M., and Hinrichs, T. , 2009. Companion Cognitive Systems: Design Goals and Lessons Learned So Far. *IEEE Intelligent Systems*, vol. 24, no. 4, pp. 36-46, July/August

Klenk, M., Friedman, S, & Forbus, K. 2008. Learning Modeling Abstractions via Generalization. *Proceedings of QR08*.

Nayak, P. 1994. Causal approximations. *Artificial Intelligence* 70:277–334.

Rickel, J. and Porter, B. 1997. Automated modeling of complex systems to answer prediction questions. *Artificial Intelligence* 93:201-260.

Wetzel, J. and Forbus, K. (July 2009) Automated Critique of Sketched Mechanisms. *Proceedings of IAAI09*.