Toward Higher-Order Qualitative Representations

Thomas R. Hinrichs and Kenneth D. Forbus

Department of EECS, Northwestern University

2133 Sheridan Road, Evanston IL 60208

{t-hinrichs, forbus}@northwestern.edu

Abstract

Qualitative reasoning typically proceeds by instantiating type-level constraints to produce a complete propositional model of a scenario, and then reasoning over this propositional representation. As we pursue the problem of learning qualitative models and applying them to open-ended construction domains, the limitations of this approach have become increasingly acute. Consequently, we have been developing representations and techniques for *higher-order qualitative reasoning,* which supports qualitative reasoning without needing to generate a full propositional scenario model. We present our type-level vocabulary and describe our experience with it in learning qualitative influences in a strategy game domain.

Introduction

Qualitative models can be an effective way to reason with partial information about a system. In the standard approach, a *domain theory* consisting of logically quantified descriptions is used to formulate one or more models of a *scenario*, by instantiating concepts from the domain theory to describe the system or situation to be reasoned about. This scenario model is propositional, since all variables have been removed during the instantiation that occurs during model formulation. It has the advantage of simplicity: Once the scenario model has been formulated, all reasoning takes place in it. All potential new entities have been generated in advance, during model formulation, so the scenario model is complete, up to the resolution and correctness of the domain theory (and the assumptions used during model formulation). This has worked well for small scenarios, and even for large scenarios, like complex engineered systems (e.g., Sgouros, 1993; Struss & Price, 2003). Unfortunately, as we explore domains that are more open-ended, involving construction of new artifacts, and especially exploring how to learn domain theories, we are finding that this process breaks down. In engineering tasks, there is typically a schematic (or equivalent) that constrains the number of entities to be considered. In planning tasks, where the point is to generate a set of entities related in effective ways (e.g. a city layout), qualitative reasoning is still useful, but the process is generating as its output what would normally be the scenario model given as input. That is what we mean by more open-ended construction domains and tasks. Using qualitative reasoning to help guide and optimize construction tasks requires a different perspective on the modeling process.

Consider for example computer strategy games, such as Freeciv[1]. Like other games in this genre, players build up civilizations by exploring, creating cities, improving terrain, building transportation networks, conducting scientific research, and engaging in trade (and warfare and diplomacy) with other civilizations as the game progresses. Games like these have many tasks that can benefit from qualitative reasoning, including optimizing city placement, deciding what to build, deciding what to research, and how to interact with other civilizations. These games are much more complex than chess. In the standard game, there are 50 types of units, 17 city improvements, and which are available at any time depends in interesting ways on the 92 technologies that can be researched. Building a civilization involves building a dozen or more cities, on a board size of 4,000 or more tiles. Growing a civilization also involves improving terrain, building transportation networks, and conducting military operations (offensive or defensive) involving other civilizations. Thus the search spaces involved are immense, so harnessing qualitative reasoning to constrain the possibilities that are examined is essential, but propositionalization would be extremely expensive.

The problem becomes more acute when the scope of investigation is expanded to include learning the domain theory, by experimentation and/or interaction with people. For efficient learning, it is important to learn the most

[1] http://www.freeciv.wikia.com

general facts that will have the broadest applicability. Hypothesized facts should be easy to validate or disprove with respect to prior experience with different instances. The representation should also correspond closely to the way we talk about qualitative relations in language. When introducing a new concept or process, we tend to describe it in general terms at the *type level*. Thus, we tend to say "Water freezes at 32°F" rather than "The water in the can freezes at 32°F."

Historically, domain theories for qualitative modeling have been designed to facilitate model instantiation, rather than be the subject of reasoning itself[2]. For example, in the Compositional Modeling Language (Falkenhainer et al, 1996), a model fragment is instantiated from a complex s-expression containing keywords and logic variables. It is more a *template* than a piece of knowledge to be reasoned about. To support reasoning and learning, the representational format of this knowledge must be simple, machine-understandable, constructable via learning algorithms, and yet remain expressive. These are a tough set of constraints to satisfy. Our solution is to develop a *type-level* representational vocabulary for qualitative reasoning. A type-level representation means that statements have arguments that are themselves either concepts or predicates. This has two advantages: 1) it breaks the monolithic templates down into smaller-grained individual statements that can be learned incrementally, and 2) it eliminates the need for variable bindings to coordinate different statements. A major benefit of this is that instances of model fragments, influences, and even quantities, need not be reified except on demand, thereby reducing the burden on a truth-maintenance system and/or a persistent knowledge base.

The next section provides some background on our representational choices and the desirable properties of higher-order qualitative representations. We then present vocabularies and examples of quantities and type-level influences. We briefly describe how our current system learns and exploits qualitative influences and proceed to outline some more provisional representations for process types. We close by discussing open remaining problems and future work.

Background

We use representational conventions from the Cyc Knowledge base (Lenat, 1995). In Cyc, concepts are divided into *collections* and *individuals*. The former are organized into a specialization hierarchy (i.e., set-subset), and the latter into a type hierarchy (i.e., membership). Statements are further organized into logically consistent contextual environments called *microtheories.*

Our use of type-level predicates in particular is inspired by examples in Cyc, where they provide concise ways to state properties of a collection without explicitly enumerating its members. For example, the relationship `relationAllInstance` takes a predicate, a collection, and a value as arguments and asserts that for every member of that collection, `(<pred> <member> <value>)` holds. Of course, internally, this is equivalent to (and may expand into) a rule that universally quantifies over the member. Yet the fact that it can be expressed in a purely structural way without logical variables makes it a simpler target representation for learning than a quantified rule and lends itself to a more efficient procedural implementation.

Desiderata for Higher-Order Representations

Ideally, higher-order representations for qualitative reasoning should be:

- Concise and scalable
- Communicable
- Compositional
- Easily reasoned with
- Learnable

The first two tend to improve together, i.e., concise statements tend to be easier to communicate. Compositionality is crucial because QR concerns the mechanisms by which things change, and being able to combine them is essential for constructing solutions to complex problems.

Reasoning is of course the point of representations. In most QR work, the number of new entities that can appear dynamically is tightly bounded as a function of the original description of the scenario[3]. For instance, in reasoning about contained fluids, the potential set of them can be constructed in advance, and it is bounded by the number of substances and phases being considered in particular containers. This is not the case for constructive domains, e.g. in Freeciv one can build cities, and improvements in cities, and units, leading to a staggering number of possibilities that would be extraordinarily expensive to instantiate.

Supporting learning also places a premium on conciseness. When we learn a domain theory empirically, we need to be able to determine quickly if the hypothesized influences and processes are consistent with the new observations, and when they are not, how best to revise the theory. Moreover, human learners often compare alternate hypotheses using analogy, as well as apply explanations via analogy to new situations. Our structure mapping

[2] There have been analyses of properties of domain theories themselves, e.g. (Forbus 1984;1992), but that is different from using a domain theory to reason through a situation without propositionalization.

[3] QP theory allows domain theories that generate potentially unbounded numbers of entities from a finite description (Forbus, 1992) but no existing qualitative reasoner supports this possibility currently.

model of analogy (Falkenhainer et al, 1989) is better suited to ground structural representations than to representations where structure may be implicit in the unification of logic variables.

Our solution to these requirements is to define higher-order predicates that take collections and relations as arguments, rather than instance-level quantities and entities. As we have mentioned, such representations are common in Cyc as a way to obviate the need for universal quantifiers, variables and skolems. The application of this approach to qualitative reasoning is novel.

Quantity Representations

Before describing influence and process representations, it is helpful to describe our representation of quantities. A *quantity* is a scalar fluent property of an entity or process. To distinguish fluents from constants, we employ a non-atomic term: in this case, a second-order function applied to a quantity type that in turn may be applied to an entity to denote a time-varying property. For example, `((QPQuantityFn Pressure) canA)` denotes the pressure inside canA. We will sometimes use a variant of this notation that takes a binary predicate as argument, for example, `((MeasurableQuantityFn citySize) FC-City-Boston)`. The predicate provides a way for our reasoning engine to sample an instantaneous value in an external system such as a game simulation. Such measurements are necessary for learning or validating a qualitative model.

Type-level Influence Representations

The idea behind a type-level influence statement is that it should represent a correct relationship about all quantities that hold in a certain relation. Previously, we have used back-chaining rules to capture this notion. However, in a system that learns influences, it is difficult to construct and reason about such rules. Instead, we adopt more concise 2nd order relations that do away with the need for variables and/or skolems.

The form of such relations is:

(i, qt_1, qt_2, c_1, c_2, r)

where:

i	=	a second-order influence predicate
qt_1, qt_2	=	quantity types represented by denotational functions as described above
c_1, c_2	=	collections representing the types of entity arguments to the quantity types
r	=	a binary relationship that must hold between the allowed instances of c_1 and c_2.

qprop+TypeType
qprop-TypeType
c+TypeType
c-TypeType
i+TypeType
i-TypeType
positivelyDependsOn-TypeType
negativelyDependsOn-TypeType

Figure 1: Type-level influence vocabulary

The vocabulary of type-level influence predicates is shown in Figure 1, and for the most part corresponds to the conventional propositional-level influence types. For example, in Freeciv, a statement that there is a direct influence (i+) from the tax revenue rate of each city to the treasure of that city's civilization would look like:

```
(i+TypeType
  (MeasurableQuantityFn currentGold)
  (MeasurableQuantityFn currentTax)
  Freeciv-Player Freeciv-City owner).
```

Clearly this is an implicit universal quantification, except that a) there are no explicit variables, and b) the scope of the statement is restricted to the query microtheory. That is, the logical context provided by the microtheory structure provides additional localization, which helps reduce the number of arguments required in the relationships, thereby improving conciseness.

The two predicates, positivelyDependsOn-TypeType and negativelyDependsOn-TypeType are slightly different in that they represent an under-specified influence caused by a non-quantitative relationship. The predicates are quaternary and take a single quantity type and collection, whose members define the dependent quantities. The remaining arguments are an instance or type of something and a binary relation which must hold between the member and the instance. So for example, the influence of irrigation on food production on tiles in Freeciv might be represented as:

```
(positivelyDependsOn-TypeType
  (MeasurableQuantityFn tileFoodProduction)
  FreecivLocation FC-Special-Irrigation
  specialAt).
```

We have successfully used this representation in a system that learns a qualitative influence model from observation and applies it to improve performance in Freeciv (Hinrichs and Forbus, 2012). In that system, a human teacher plays the game while the learning agent monitors actions and quantity changes, both within a turn and across turns. Within-turn changes are considered to be *instantaneous* events due to actions that change relationships in the game. Changes across turns are

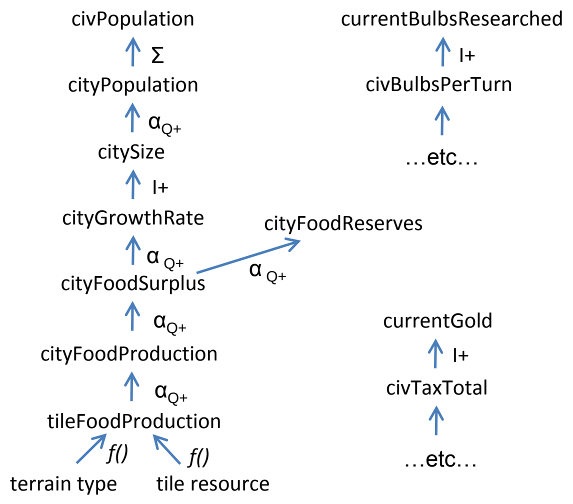

Figure 2: Learned Freeciv model

considered to be *durative*. The implication of this is that instantaneous changes cannot result from direct influences.

The learner hypothesizes possible propositional influences, and lifts them to the type level by identifying explicit binary relations between entities of the quantities and abstracting entity and quantity types. It verifies consistency by continually testing observations against the set of hypothesized influences with respect to the basic constraints of Qualitative Process Theory (e.g., no mixed influences and the sole mechanism assumption). It identifies counter-examples of hypotheses by retrieving instances of prior quantity changes from specialized influence microtheories, organized by influence type. This mechanism enables the system to learn a model (partially shown in Figure 2) in one 75-turn game trial.

The learned model is used to guide behavior of the game player in autonomous play by helping to decompose high-level quantitative goals into sub-goals. For example, it takes a goal to maximize the civilization population and regresses through the learned model until it finds executable actions that influence quantities in the appropriate direction to, e.g. increase the rate of growth in a city or increase the number of cities. There is, of course, much more scaffolding required to make this work, which is beyond the scope of this paper, but see (Hinrichs and Forbus, 2012) for more detail.

The learned model enabled the Freeciv player to significantly improve its performance over random play on the task of growing a civilization population over 75 turns, where the average improvement was well over a factor of two. As we can see, this type-level influence representation is learnable and relatively easy to reason with, at least for the task of decomposing goals. To show that it is concise and scalable, we ran a Freeciv game for 75 turns using the learned influences and at that point we

Type-level influences		*Propositional Influences*	
i+TypeType	4	i+	36
i-TypeType	0	i-	0
qprop+TypeType	10	qprop	210
qprop-TypeType	7	qprop-	179
total	**21**	total	**425**

Table 1: Type level vs. propositional influences in Freeciv after 75 turns

generated propositional statements for all the possible entity bindings (See Table 1). In total, there were 21 learned type-level influences. When instantiated, these expanded into 425 propositional influences. Because the number of propositional influences grows with the number of entities in the scenario, whereas the number of type-level influences is constant, an order of magnitude reduction is easily achieved.

If the domain under consideration were completely static, with no phase changes, this would suffice entirely. In fact, in modeling cities in Freeciv, this gets us surprisingly far toward a useful model for reasoning about how to improve growth rates. Yet a major purpose of QP theory is to translate a continuous system into qualitatively distinct states. For this, we need processes.

Type-level Process Representations

Qualitative process theory has traditionally used logically quantified representations to define types of processes and views. The modeling languages used by specific implementations tend to include syntactic transformations to simplify the work of human modelers, which get translated into logically quantified statements that are then used in model formulation to produce propositional representations. The instances in the propositional scenario are what is used for qualitative reasoning in existing implementations. The problem is that explicit logical quantifiers are more complex to reason about directly and to learn. Hence we propose here a set of type-level predicates that we believe suffice to define continuous processes, in the usual manner of QP theory. These representations are more preliminary than the influence predicates defined above, as we are still in the process of experimenting with and evaluating them.

Any process representation must somehow associate influences with conditions for activation. Although we might like to simply incorporate and extend the type-level influences presented in the previous section, this turns out to be infeasible. Our first attempt assumed that we could query for influences as before, and dynamically infer active process conditions to filter those influences. This would be a backwards inference from possible influences to conditions, rather than the more typical forwards inference that operates by first instantiating processes and querying conditions before applying influences.

The backwards approach appeared to work in the domain of Freeciv, but as it turns out, real physical domains often have significantly more contextual role bindings that must be coordinated between influences and conditions. For example, in order to model something like fluid flow at the type level, constraints on the path, the substance, the source and destination all must be satisfied. Since any given influence only references two quantities, the query interpreter would have to bind and carry around the additional quantities and participants in order to properly constrain activation conditions. That's what a reified process instance normally provides.

Instead, we've adopted a more traditional forward inference model that first associates role relations with entities, computes activation for that particular set of role-entity substitutions, and then gathers active influences. Rather than rely on a self-contained type-level predicate that can be implemented in terms of backchaining rules, we must assume the existence of a more sophisticated interpreter that may need to maintain some local state, in the form of role bindings.

This can be achieved with four new type-level predicates, listed in Figure 3. Each of these predicates takes a *process type* as its first argument and associates with it an individual role-type, conjunct of an activation condition, or influence. The idea is to break the definition of the process type into the smallest, incrementally learnable pieces.

The first predicate, `participantType`, associates a *role-relation* name and an entity type that constrains its possible bindings. A role relation is a named accessor of a reified concept, not unlike a slot in a frame. Possible bindings are limited to instances in the microtheory containing the scenario.

```
participantType
participantConstraint
conditionOf-TypeType
consequenceOf-TypeType
```

Figure 3: Type-level process vocabulary

`ParticipantConstraints` allow further filtering of bindings. The arguments to the constraints are themselves names of role relations, which the interpreter must substitute at query time with their bindings.

`ConditionOf-TypeType` specifies a condition that must be true for a process instance to be active. These may be *quantity conditions*, in which case they are individual binary inequalities, or they may be more arbitrary Boolean conditions on activation. The arguments to Boolean conditions are also role relation names. The arguments to inequalities must be quantity fluents (or constants). Syntactically, that means a quantity type applied to a role relation which designates an entity.

`ConsequenceOf-TypeType` specifies the influences that hold when all the role bindings satisfy the process type conditions. As with quantity conditions, the arguments are quantity types applied to role relations. In some cases, an argument may need to be a function applied to a role relation, to indicate, e.g., the volume of the liquid contents of a container. Unlike the type-level influences, the influence predicates are the standard propositional-level predicates, since the arguments denote particular entities, rather than collections.

Figure 4 shows what a process type description might look like for the LiquidFlowProcessType. We can see that this is a type-level description, since the predicates are all ground and take a process type and other predicates (role relations) as arguments.

```
(participantType LiquidFlowProcessType fromLocation LiquidContainer)
(participantType LiquidFlowProcessType toLocation LiquidContainer)
(participantType LiquidFlowProcessType substanceOf ChemicalSubstanceType)
(participantType LiquidFlowProcessType pathOf FlowPath)
(participantConstraint LiquidFlowProcessType
    (fluidPathBetween fromLocation toLocation pathOf))
(conditionOf-TypeType LiquidFlowProcessType
  (qGreaterThan ((QPQuantityFn FluidPressure) fromLocation)
                ((QPQuantityFn FluidPressure) toLocation)))
(conditionOf-TypeType LiquidFlowProcessType
  (qGreaterThan ((QPQuantityFn AmountFn) (LiquidContentsFn fromLocation)) 0))
(conditionOf-TypeType LiquidFlowProcessType (aligned pathOf))
(consequenceOf-TypeType LiquidFlowProcessType
  (i+ ((QPQuantityFn AmountFn) (LiquidContentsFn toLocation))
      ((QPQuantityFn Rate) processInstanceOf))
(consequenceOf-TypeType LiquidFlowProcessType
  (i- ((QPQuantityFn AmountFn) (LiquidContentsFn fromLocation))
      ((QPQuantityFn Rate) processInstanceOf))
```

Figure 4: Liquid Flow process type

Related Work

A number of efforts have explored learning qualitative representations, albeit with very different task constraints. For example, Padé (Zabkar et al, 2011) learns qualitative models given complete data sets, whereas our learning operates incrementally and via demonstration. Suc & Bratko (1999) used qualitative reasoning to generate strategies by cloning traces of experts solving continuous control problems. Our domain involves more discrete actions, and multiple qualitative states. QLAP (Mugan & Kuipers, 2012) uses dynamic Bayesian networks to learn qualitative representations via incrementally introducing landmarks. The representations it is constructing are still state-based, unlike our type-level representations.

There has been prior work on learning higher-order horn clauses in Inductive Logic Programming (Pahlavi and Muggleton, 2009). There, the goal was to learn new higher-order predicates given a set of training instances and background knowledge. Here, we are designing a particular higher-order representation for qualitative models and using that as background knowledge to learn and reason from instances of models. Our goal is to investigate the role of qualitative representations as an inductive bias to allow learning to be more incremental and interactive.

Conclusions

The representations we have presented here are a step toward more general higher-order qualitative reasoning. Type-level influences have proven to be very effective for learning and scalable reasoning. The type-level process representation remains somewhat preliminary as algorithms are still under development and there is not yet empirical evidence to support claims of scalability or learnability for it. Yet, our experience with type-level influences leads us to believe this will ultimately bear fruit.

One expected benefit of a type-level process representation is that it provides a better "impedance match" to everyday language. Prior work integrating qualitative reasoning with language understanding (Kuehne & Forbus, 2004) focused on learning instance-level models. As we work toward learning-by-reading and advice-taking systems, we find this often avoids the need for obscure skolems or quantifiers in the target representation, simplifying both reasoning and analogical matching.

There is still much work to be done, especially on learning new process types. There are a number of open questions, such as: If we must hypothesize new role relations, where do their names come from? Can there be a shared vocabulary of roles? How can we efficiently test whether a learned model is satisfied by given observations? Is it necessary to first learn a type-level influence model and then translate into process types when limit points are discovered? We will be addressing these questions as we continue to refine and extend our learning system.

Acknowledgements

This material is based upon work supported by the Air Force Office of Scientific Research under Award No. FA2386-10-1-4128.

References

Falkenhainer, B., Farquhar, A., Bobrow, D., Fikes, R., Forbus, K., Gruber, T., Iwasaki, Y., and Kuipers, B., 1996. CML: A Compositional Modeling Language. *Working Notes of The Tenth International Workshop on Qualitative Reasoning*, Iwasaki & Farquhar (eds). AAAI Technical Report WS-96-01. AAAI Press.

Falkenhainer, B., Forbus, K. and Gentner, D. 1989. The Structure Mapping Engine: Algorithm and examples. *Artificial Intelligence*, 41, 1-63.

Forbus, K.D., 1984. Qualitative Process Theory. *Artificial Intelligence* 24:85-168.

Forbus, K., 1992. Pushing the edge of the (QP) envelope. In *Recent Progress in Qualitative Physics*, Faltings, B. and Struss, P. (eds). MIT Press.

Hinrichs, T.R., and Forbus, K.D., 2012. Learning Qualitative Models by Demonstration. To appear: *Proceedings of AAAI 12.*

Kuehne, S. and Forbus, K., 2004. Capturing QP-relevant information from natural language text. In *Proceedings of the 18th International Qualitative Reasoning Workshop*.

Lenat, D.B., 1995. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM* 38(11):33–38.

Mugan, J. and Kuipers, B., 2012. Autonomous learning of high-level states and actions in continuous environments. *IEEE Transactions on Autonomous Mental Development* 4(1): 70-86, 2012.

Pahlavi, N. and Muggleton, S., 2009. Higher-order Logic Learning. In *Proceedings of the 19th International Conference on Inductive Logic Programming (ILP '09).*

Sgouros, N. 1993. Representing physical and design knowledge in innovative design. Ph.D. diss., Department of Computer Science, Northwestern University, Evanston, IL.

Struss, P., & Price, C. 2003. Model-based systems in the automotive industry. *AI Magazine*, 24(4).

Suc, D. and Bratko, I. 1999. Modeling of control skill by qualitative constraints. In *Proceedings of the 13th International Workshop on Qualitative Reasoning*, Loch Awe, Scotland.

Žabkar, J., Možina, M., Bratko, I., and Demšar, J. 2011. Learning qualitative models from numerical data. *Artificial Intelligence* 175:1604-1619.