Thought Experiments as a Framework for Multi-level Reasoning

David Hibler
Dept. of Computer Science
Christopher Newport University
Newport News, VA 23606.
tele: (804)-594-7065
e-mail: dhibler@pcs.cnu.edu

Gautam Biswas
Dept. of Computer Science
Vanderbilt University
Nashville, Tennessee 37235.
tele: (615)-343-6204
e-mail: biswas@vuse.vanderbilt.edu

June 26, 1992

Abstract

This paper discusses recent developments in the thought experiment methodology and their relation to multi-level reasoning. Thought experiments involve *Simplification* of the original problem, *Solving* the simplified problem, *Conjecturing* an answer to the original problem based on the solution of the simplified problem, and *Verifying* the results. First, we review the methodology briefly. Next we relate simplification methods to hierarchical modeling and reasoning by analogy. We present a catalog of methods. We illustrate some of these methods with a simple example. Next we describe verification as we do it and relate it to work in hierarchical modeling.

Note:: This paper is based on D. Hibler's Ph.D. dissertation.

1 Introduction

As modeling issues and methodologies stabilize, qualitative reasoning is now being applied to model large and complex engineering systems. The amount of information captured in these models is orders of magnitude greater than the simpler models discussed in earlier papers. It is generally recognized[3,4,8,10,19] that analysis of complex models requires the use of *abstractions* and *simplifying assumptions* to manage computational complexity.

A common theme in previous research[1,2,6,10,13] is the use of *abstraction* for speeding up reasoning tasks in computationally complex situations. Korf[12] has shown by theoretical analysis that the use of a hierarchy of problem spaces can transform an exponential problem into a linear one. In an attempt to generalize a number of the previous results, Weld and Addanki[19] develop a framework for defining a number of dimensions along which models can be abstracted: (i) parameter value representations, (ii) component constraints, (iii) temporal abstraction, and (iv) aggregation and structural consolidation. Their goal is to generalize on previous work, and take "first steps toward eliminating the need for prespecified abstract models"[19]. This goal is also targeted by Falkenhainer and Forbus[4] in their work on query-driven compositional modeling. Weld and Addanki create a formal framework for task-driven abstraction based on a set of definitions: the upward- and downward-solution, and the upward- and downward-failure properties.

This paper discusses an alternative approach to reducing computational complexity in analyzing the behavior of complex systems – the thought experiment method. What is a thought experiment? Imaginary, simplified situations are often analyzed by human problem solvers in order to understand the principles behind more realistic situations[14]. We have formalized this heuristic method, and developed a problem solver called TEPS[7,8] for qualitative physics problem solving.

2 TEPS: A Thought Experiment Problem Solver

The first step in a thought experiment involves *simplification.* The time evolution of a physical system is described in terms of the system occupies different qualitative states at different times. Given a state q which specifies a physical system at some time, a simplification S(q) produces a state p of a simplified version of the original system. An example of a simplification is *Population Reduction.* This consists of reducing the number of identical objects in a problem. Given a series of pendulums with metallic bobs (say 25), and a charge c placed on the first pendulum (all others have zero charge), Population Reduction would produce a simplified problem consisting of two pendulums with charge c on the first[7]. Simplification functions, such as Population Reduction are transformations provided to the problem solver by the system designer.

The next step in a thought experiment involves "*solving*" the simplified model. This means that the thought experiment problem solver must contain a reasoning engine which takes a qualitative state q and reasons about it to produce some result R(q). The reasoning engine employed for problem solving and its corresponding results space is independent of the basic thought experiment framework. TEPS, the thought experiment problem solver which has been implemented in Prolog uses a qualitative simulator that is based on Forbus' QPT modeling[5]. It produces a graph of qualitative states, all of which can be reached from the state q. Thus for TEPS, R(q) is the state graph produced by qualitative simulation.

The input to TEPS is a set of state descriptors for the initial state of the system along with a query that describes the solution sought. For the pendulum example, the input to TEPS is shown below:

```
problem(
  state: [conductor(p), pendulum(p), mobile(p),
          location(p(center(p)),   charge(p(1),c),
          number(p,N,25)],
  query([final states,
         descriptor(location(p(N),X), argd(2))],
         pred,someall)
       ).
```

The state specification indicates that there are 25 pendulums, each is a conductor, is mobile, and is located in the center (vertical) position. The charge on pendulum one is c. The query asks that the final states of the resulting state graph be examined, and that we find the second argument of the descriptors that match `location(p(N),X)`.

The problem is represented in the form of a *problem frame* and sent to the Thought Experiment module. The first step that this module performs is to obtain all the active processes for this problem from the simulation module. It then performs simplification on the initial state by looking up a *catalog of simplifications.* Discussion about the catalog and its characteristics is the primary focus of this paper. For control purposes, possible simplifications are ranked by heuristic analysis, and then performed in the order of ranking. If configured for multiple simplifications the system will use as many as are applicable. The simplified problem is given to the simulation module which returns a state graph for the system with the given initial state.

The thought experiment problem solver is designed to answer specific questions about a physical system given an initial state for the system. For this reason we are not concerned with R(q) directly because it usually does not constitute the answer to a question. The description language is specified by description functions. Such a function is called a *description basis.* A description basis, D, is a mapping from the set of state graphs to some adjective space of labels. If we prefer we may think of a description basis as a way of classifying state graphs with the adjectives being the labels for each category. D(R(q)) is a description of the results of the reasoning process starting with state q, and it constitutes an answer to the specific question asked. In the example above, the initial state consists on a number of pendulums in an initial configuration. R(q) is a state graph representing time evolution of this system. The query provides the label for a description basis D that the system knows about, in this case, `center( )`, `right( )`, and `left( )`. The results are classified using `someall`. A library of description bases and functions for constructing description bases are provided as part of the TEPS implementation. In the above case, the final states are examined to determine the location of the last pendulum in terms of the above description.

A *conjecture*, C, is a guess about the description of the result of the original problem based on the solution obtained on a simplified version of it, D(R(q)) = C(R(S(q))). The simplest conjecture is that the description basis is generic enough to provide a cross description i.e., D(R(q)) = D(R(S(q))). In the pendulum example, the description basis specifies final locations for the pendulums. For the two pendulum problem, the location of the pendulum is `[all(right)]`. The conjecture is that for the multiple pendulum situation the same description holds.

$q \rightarrow R(q) \rightarrow D(R(q))$	Correct Answer is $D(R(q))$
$q \rightarrow S(q) \rightarrow R(S(q)) \rightarrow D(R(S(q)))$	Tentative Result of T.E.
$q \rightarrow S'(q) \rightarrow R(S'(q)) \rightarrow D(R(S'(q)))$	Attempt at confirmation
$D(R(S'(q))) = D(R(S(q)))$	Successful Single Heuristic Verification

Table 1: THOUGHT EXPERIMENT METHOD

Verification can be rigorous or heuristic. It could even be empirical. Rigorous verification consists of establishing that the conjecture is true. Empirical verification consists of comparing the predictions with what actually occurs in the real world. This is not usually practical. The most flexible type of verification is heuristic. With this type of verification other simplifications are tried and the conjectures compared with the original conjecture. If they agree we accept the conjecture as a reasonable belief. In the pendulum problem we might repeat the procedure with a three pendulum simplification instead of two. The steps of the Thought Experiment Problem solver illustrated above are summarized in Table 1.

3 Simplifications and multi-level reasoning

This section outlines how thought experiments provide a novel formalism for describing multi-level reasoning in the form of approximations and abstractions. We then demonstrate how this methodology reduces computational complexity in query-driven problem solving. We first classify simplifications along three dimensions: (i) domain dependence, (ii) correspondence classification, i.e., the relationship between the original model and the simplified one, and (iii) simplification strategies used. This provides a framework for comparing our work with that of others. Next, we discuss specific examples of simplifications of various types, and build a catalog of simplifications.

3.1 Domain Dependence

The applicability of a simplification depends on various characteristics of the problem. Sometimes the simplification can only be applied to a specific physical domain. We consider domain independent simplifications to be much more desirable than domain dependent ones. A problem solver which uses only domain dependent simplifications must be given a different set for each domain with which it deals. Population reduction is an example of a domain independent simplification, whereas structural consolidation (i.e., replacing a set of components by a more abstract component in a physical system schematic) would be an example of a simplification that is domain dependent.

3.2 Correspondence Classification

In terms of the correspondence between the original model and the simplified model we may divide simplifications into three rough categories. These categories are *hierarchical simplifications*, *similarity simplifications*, and *analogical simplifications*.

Hierarchical simplifications produce models which are more abstract or which leave out features of the situation which are irrelevant so far as the current problem is concerned. Hierarchical models have been studied by others(e.g., [2,6,10,13]). The main innovation

the thought experiment method would provide is the emphasis on simplification methods as opposed to fixed models. A thought experiment problem solver would construct its own simplified models. This provides more flexibility in adapting the model to the situation and in changing models. The need for this flexibility has begun to be recognized. (Compare[3] with [4].) The problem of verification with hierarchical models is often just a question of deciding if the simplification is appropriate. Hierarchical models are usually domain dependent.

Similarity simplifications involve finding simpler models in the same domain. Unlike hierarchical models these models are clearly different and not merely more abstract versions of the same model. In some sense, they incorporate both approximation and abstraction. Heuristic verification is often used with these simplifications. At least some simplifications of this type are domain independent. Population Reduction is an example of a similarity simplification.

Analogical simplifications involve rules for specifying an analogous problem in a different domain. We have avoided using analogical simplifications in TEPS in order not to have to deal with difficulties arising from using different domains.

3.3 Simplification Strategies

Many different simplification strategies might be used. Most can probably be classified as: *constraint augmentation*, *increased specificity*, and *variable reduction*. Constraint augmentation maps the problem to one with stronger constraints. Increased specificity makes constraint relationships which already exist more effective in determining behaviors. Variable reduction reduces the number of variables which must be dealt with. The purpose of each of these strategies is to reduce the average outdegree of non terminal nodes of the state graph.

3.4 A Partial Catalog of Simplifications

A primary issue in the thought experiment framework is the use of appropriate simplifications. We will, therefore, present a brief discussion of various possible simplifications organized by the classification scheme described above. We do not claim this to be a complete list of simplifications. However, we present a nontrivial representative set that brings out necessary characteristics the problem solver needs for effective problem solving. These simplifications have been found useful and even necessary by the qualitative physics community to address the complexity problem. This shows that useful and practical simplifications exist. The simplifications we have developed for abstraction[6,10] are based on domain-dependent hierarchies, however, a number of the simplifications that correspond to approximations are domain-independent.

In order to show that the thought experiment method generalizes in a useful way the techniques mentioned in the introduction it is desirable to include in our catalog simplifications of the similarity type which do not correspond to anything discussed in the introduction. Domain independent simplifications of the similarity type are particularly important because of their versatility.

3.4.1 Constraint Augmentation Using Selection Methods

Selection methods involve simplifying the problem by selecting only a part of the state graph generated by qualitative simulation. This can be considered constraint augmentation. The new constraint acts as a filter to eliminate states which would otherwise be generated.

A prime example of a domain independent simplification which is of the selection type is the *Monte Carlo* simplification. This simplification has been implemented in TEPS[8]. The wide applicability of the Monte Carlo simplification makes it extremely important. Monte Carlo techniques have been widely used in other areas of research but do not seem to have been investigated by the qualitative physics community.

Monte Carlo is a term frequently used to refer to methods which generate probabilistic solutions to mathematical or physical problems using statistical sampling techniques[9] What we call Monte Carlo simplification works as follows. The children of the start state are generated and one of the children is picked at random with equal probability. Whenever a state is picked its children are generated and one of them is picked at random with equal probability. This process continues until one of three things happens: a state with no children is picked, a previously picked state is picked, or a specified resource bound on the simulation is reached. The resultant path through the graph is called a Monte Carlo path.

Basically, a Monte Carlo simplification randomly samples the results of qualitative simulation. The belief is that such a result will tend to be a typical one. Any description of the result which does not explicitly involve the number of final states or the number of paths through the graph is assumed to be the same for the Monte Carlo simplification and the whole graph. Verification is heuristic and is obtained by choosing a new Monte Carlo path through the graph. In the very unlikely event that all random choices were the same, the simplification would be repeated until a different path was chosen. If verification fails and it is desired to continue with Monte Carlo simplification all currently known Monte Carlo paths are combined, a new conjecture based on all known paths is made, and verification is done with yet another new path.

Monte Carlo simplification effectively ignores irrelevant distinctions such as irrelevant timing details which cause the state graph to have an extremely high branching factor. The simplification strategy can be considered to be constraint augmentation. (The new constraint randomly selects states.) It is domain independent and it is almost universally applicable.

Another simplification which is similar to Monte Carlo is *Combined Change* simplification. This has also been implemented in TEPS. Monte Carlo simplification randomly selects a single child state to explore. This can exclude some changes in variables. Combined Change simplification makes sure that all possible variable changes have occurred but selects a minimal set of child states for which this is true. The rationale is that when changes may occur both separately and in combination their timing is probably irrelevant.

Unlike Monte Carlo, one Combined Change simplification cannot be verified with another Combined Change simplification as the two would be identical. The best method of verification for a Combined Change simplification is a Monte Carlo simplification. Like Monte Carlo, Combined Change is a similarity simplification using constraint augmentation. It is domain independent and it is almost universally applicable.

Many domain dependent methods could be developed which are similar to Monte Carlo and Combined Change. They would apply domain dependent heuristics to select a restricted set of children to explore from any given state. These children would be selected as typical or because they were believed to be equivalent to ignored children. Verification for these

methods should probably be done by Monte Carlo simplification.

3.4.2 Other Methods of Constraint Augmentation

There are many other possibilities for constraint augmentation. Many of these are domain dependent. One method which is fairly general is *Dimensional Constraint.* This can be used when objects in the model have freedom to change in three dimensions. Adding a constraint which allows changes only along one dimension effectively reduces the dimensionality of the problem to one. Dimensional Constraint is a special case of the *Variable Blocking* simplification. This simplification adds the constraint that certain variables not be allowed to change. One set of variables is blocked and the problem solved. Verification is performed by doing the same thing but blocking different set of variables. If the same result is obtained in both cases then both sets of variables were irrelevant to the problem. This acts almost like a variable reduction method but may be easier to implement in certain cases.

One technique which may be useful in combination with Variable Blocking is complementary simplifications. Suppose we try Variable Blocking and verification fails. This means that we have two sets of variables. One set was held constant (blocked) and the problem solved. For verification a different set was held constant and the problem solved. The descriptions of the two results do not agree. We may be able to construct a composite of the two results at least in terms of final states.

One method for forming conjectures based on two different simplifications just combines final states using a simple *superposition* method. Physically this works when there are two non interacting systems in the model. Any change in a variable comes from one of the systems but not the other. We block the variables for one system and simulate, then block the variables for the other and simulate. The rule for combining final states is: choose the value for a variable in the combined state by always preferring a changed value over an unchanged value. If the initial state has variables (X1,Y1,Z1) and the two states to be superposed have (X2,Y1,Z1) and (X1,Y2,Z1) the combined state has (X2,Y2,Z1).

A more sophisticated version of superposition would involve heuristic methods for determining when two systems interact and when they do not. During any time when the systems are believed to not be interacting they would be modeled separately using Variable Blocking. When interaction starts the results of the separate simulations would be superposed and the two systems would be modeled together. When interaction ceases variable blocking would be used again. Final states would be formed by superposition. This method has similarities to history based reasoning techniques[20].

Another method for making conjectures based on two different simplifications uses inheritance and is given in detail elsewhere[8]. Basically the method allows processes and individual views to examine previous thought experiments and incorporate information from them to determine relationships. This inheritance method is one way of implementing superposition and is used in TEPS.

Temporal abstraction as discussed by Kuipers[11] is another type of constraint augmentation. From our viewpoint temporal abstraction consists of rules which rank processes by relative strength. Any change generated by a higher rank (stronger process) will be allowed to take place. Changes due to weaker processes will be suppressed until the stronger processes generate no more changes.

3.4.3 Increased Specificity

Increased specificity of the initial state causes constraints to be more effective in reducing the number of possibilities. An example is the spatial regions hierarchy given by (region, connected region, convex region, sphere), where sphere is the simplest (most specific).

Increased specificity methods are domain specific, similarity simplifications. They can easily be built into modules describing individual views or processes. This might be called a *Simple Stereotype* method. For example, an individual view describing a generic building structure may contain descriptors for a stereotype building that could replace those for the less specific building structure.

3.4.4 Variable Reduction

Most alternate physical models of a situation or object can be arranged as a simplification hierarchy. These models come from the specific domain. For example, a solid object might be modeled as a collection of molecules, an elastic solid, a rigid solid or a point particle. These models usually achieve simplification by reducing the number of variables. Many approximations also can be characterized as variable reduction. Some of these types of simplifications have been mentioned in section 1 in connection with current research[3,4,1,19]. These simplifications are hierarchical and domain specific. Verification often consists simply of checking if the simplified model is consistent, and if it contains the necessary variables to answer the question.

Sometimes variable reduction is achieved by eliminating certain classes of variables, all of which are associated with a specific *ontological perspective.* For example, eliminating all processes and variables which are tagged with the thermodynamic perspective. This can be done if a query does not refer to anything tagged with this perspective[4]. *Structural Consolidation*[3,4,19] consists of a systematic organization of models such that the components of a mechanical system at any given level may be considered as "black-boxes" with no internal structure.

It should be pointed out that the Simple Stereotype approach mentioned earlier can be considered a variable reduction method if the original model is specific but complicated. For example, replacing a building with many rooms by a building with only a few rooms reduces the number of quantities associated with each room. This, however, is a similarity simplification and not a hierarchical simplification as described above.

Since domain independent methods are particularly desired, the *Aggregation* simplification of Weld[16] is especially important although it is only applicable in certain cases. Aggregation is a domain independent, hierarchical simplification.

Another variable reduction method which is domain independent is *Population Reduction.* This is another method implemented in TEPS. Given a set of identical objects (a population of those objects) we reduce the number of objects. This method is domain independent. Like aggregation it is applicable only in certain cases. We have applied it in TEPS to electrostatic problems. It is possible that this method could be extended by first examining a problem for a collection of objects which are of the same type even though they differ in details. These objects would be described by the same individual view module and each could be replaced by the stereotype obtained from that module. The result would be a population of identical objects to which Population Simplification could be applied.

Variables can often be essentially eliminated or at least their effects can be simplified

NAME	SIMPLIFICATION STRATEGY	NATURE OF CORRESPONDENCE	DOMAIN DEPENDENCE
Monte Carlo	Constraint Augmentation	Similarity	Independent
Combined Change	Constraint Augmentation	Similarity	Independent
Dimensional Constraint	Constraint Augmentation	Similarity	Independent
Variable Blocking	Constraint Augmentation	Similarity	Independent
Superposition	Constraint Augmentation	Similarity	Independent
Simple Stereotype	Increased Specificity	Similarity	Dependent
Simple Stereotype	Variable Reduction	Similarity	Dependent
Ontological Perspective	Variable Reduction	Hierarchical	Dependent
Structural Consolidation	Variable Reduction	Hierarchical	Dependent
Aggregation	Variable Reduction	Hierarchical	Independent
Population Reduction	Variable Reduction	Similarity	Independent
Exaggeration	Variable Reduction	Similarity	Independent

Table 2: SOME BASIC SIMPLIFICATIONS

by assuming they have extreme values such as zero, infinitesimal, and infinite. A systematic technique which uses this is the *exaggeration* method of Weld[18]. We summarize the simplifications we have discussed in Table 2.

4 The Pendulum Example

It is useful illustrate the use of different simplifications by continuing the pendulum example described in section 2. Additional simplifications which are applicable include Monte Carlo and Combined Change.

First we run TEPS with just the Monte Carlo method on this problem. This can be done by giving Monte Carlo simplification the highest ranking of any in the simplification library and specifying single simplifications in a configuration file.

One advantage of trying a Monte Carlo method first is that the outdegree of nodes along a Monte Carlo path through the graph together with the path's length would enable TEPS to estimate the complexity of the graph. This estimate could be used for control purposes. This will be implemented in the near future.

Application of the Monte Carlo method produces a tentative result as follows. TEPS picks Monte Carlo as highest ranked, calls the associated applicabilility test, discovers that Monte Carlo is applicable and uses the Monte Carlo simplification. This simplification merely adds the predicate *mcarlo* to the predicates for the initial state. This new predicate informs the simulation engine to randomly select children of new states for further exploration and to stop when a leaf is reached, a previously explored state is reached, or the graph size specified in the configuration file is exceeded. The description basis is applied to this graph and a tentative answer to the query is obtained.

Monte Carlo performs well on the pendulum problem given in section 2 although not as well as Population Reduction. In this problem 25 pendulums are swinging simultaneously. The timing turns out to be irrelevant. Monte Carlo effectively chooses a particular order for the swings and this reduces the complexity enormously.

When TEPS calls the verification routine associated with the Monte Carlo method the same state is sent to the simulator again. The simulator always checks to determine if the state graph for an initial state is already known before performing any envisionment. When it finds the old graph it takes one of the unexamined children at random and proceeds with the Monte Carlo simulation on it. The verification method checks that the new, larger graph has the same description.

The effect in our pendulum problem is to check whether the assumption, that the order of pendulum swings is irrelevant to the query, is true. Obviously this will not be a complete check but it will show whether this assumption is plausible. A more sophisticated control structure (not employed) could continue checking until some reasonable resource bound was reached.

Combined Change works in an almost identical manner and for similar reasons. It is faster however. All three types of simplification produce the correct answer. Population Reduction, Combined Change, and Monte Carlo is the order of efficiency of the three methods with Population Reduction the most efficient.

The above results indicate a reasonable ranking for these simplification methods. When all three methods are included in the simplification library and the multiple simplification flag is set we have Population Reduction chosen first because it is most highly ranked. Combined Change is then applied. Monte Carlo will never be used with this ranking as it is not compatible with Combined Change. The result using multiple simplifications is also correct and appreciably faster than any single simplification.

5 Comparison with Other Methods

Any form of multi-level reasoning that involves abstractions and approximations can be framed in a simplification terminology. The major difference between the similarity simplifications which are implemented in TEPS and hierarchical simplifications which are commonly used by others is that verification seems trivial for hierarchical simplifications. The verification method can be termed verification by simplification type. Since this method is important we need to discuss it and to compare its strength's and weaknesses with heuristic verification.

5.1 Verification by Simplification Type

With some simplifications we are "guaranteed" that if the simpler model produces any answer at all it is a correct answer. Thus the only way a thought experiment based on such a simplification can fail is by description failure. The basic idea is that the simplification produces a model which is less detailed than the original model, and, therefore, may not be capable of answering certain questions. However, the model is just as valid and just as applicable to the original situation. This means that either the model gives no answer to our question (description failure) or it gives the correct answer.

An example of verification by simplification type is given by Falkenhainer and Forbus[4]. They mention both empirical and heuristic verification as possibilities (without using this terminology). The method they employ, however, is consistency of the model. They feel that this is the only method that is needed for the simplifications they use. Consistency is detected by envisionment. If a null envisionment is produced then a new model must be constructed. If a successful (non empty) envisionment can be performed the simplified model is verified. In our terminology we would say that a null envisionment produced description failure. Falkenhainer and Forbus do not have to worry about description failure in any other case than this since they construct their models based on the query itself.

Since verification by simplification type is important let us formalize it in our system in order to make sure our notions are precise. We adapt some terminology from Weld and Addanki[19] who in turn adapted it from some ideas of Tenenberg[15].

> A simplification, S, will have the UPWARD-SOLUTION PROPERTY relative to a set of description bases, L, iff for any description basis, D, in L and any state, q, it must be true that either (a) D(R(q)) = D(R(S(q))) or (b) D fails on R(S(q)).

The upward solution property is just the property in which we are interested. If a thought experiment problem solver has simplifications which are tagged as having this property and the description is in the appropriate set, L, then verification consists of no more than noticing the tags and that there was no description failure. This is verification by simplification type.

5.2 Problems with Verification by Simplification Type

Hierarchical modeling is popular at present in qualitative physics. All such systems use (at least implicitly) verification by simplification type. Thought experiments include this method but are more general. It is our feeling that such generality is needed so we would like to point out some problems with this method. Despite the problems, verification by simplification type is an important and useful method simply because the verification is rigorous.

An important point is that the upward solution property of a simplification can only be defined relative to a fixed set of description bases, L. If queries are not limited in this way then it is always possible to make a model dependent description or a model dependent query. There has to be some difference between the models and we can always make the query dependent on this unless it is forbidden to ask such questions. For example, we can always ask for the number of variables or the number of parts. These usually vary in hierarchical simplifications.

Ideally, L should be as large a set as possible and should include all descriptions needed to answer intuitively natural questions. The obvious choice for L is to formulate all description bases in terms of some sort of first order modeling language. Such a language would prevent meta level queries and yet would provide reasonable tools for constructing descriptions.

One difficulty with using a first order language for L involves models which already exist in the physical sciences. Most simplified models which are traditionally used in the physical sciences cannot be considered as simplifications with the upward solution property with respect to descriptions in any reasonable first order language. Consider a glass full of water. At one level it may be modeled as a collection of molecules. At another level it may be modeled as a continuous fluid. Do we have the upward solution property for every reasonable description? Most emphatically not. Certainly if we mention the position of a molecule in the fluid model we get description failure. This is to be expected. What about the density of the water at any location in the glass? At the fluid level it is uniform and never zero. At the molecular level it is exactly zero at one place and enormous at another. There is no simple logical language which determines whether the result at the higher level is the same as that at the other. The criteria as to whether the descriptions must be the same, even qualitatively, are complicated and often numerical. For this reason, specifying L can present a problem for verification by simplification type.

Another problem is that even if we can specify L easily, and even if L turns out to be based on a nice first order language, many intuitively interesting queries could not be formulated in terms of L. Questions such as: "Is the system in equilibrium?" and "Is the behavior cyclic?" implicitly quantify over all variables and are not first order queries. Let us consider the question: "Is the system in equilibrium?". In physics terminology we would say that this question asks whether all variables have constant values. Thus it implicitly quantifies over all variables. A variable is not an object but a way of describing objects and so this query is not a first order query. In order to be sure that we are being precise let us translate this argument into the terminology of predicate calculus. In physics it is common to refer to variables and to suppress the role of predicates by using them in the "definition of the variable". For example, if we say T1 has value t our definition of T1 may make this statement about T1 equivalent to saying temperature(object1,t) is true. If we introduce the defining predicates for the variables, the question of equilibrium can be formulated by saying that the set of all predicates needed to characterize the state of the system at any instant does not change. This set is determined partly by the objects in the system but also by the types of predicates which are needed to characterize the system. We are quantifying over kinds of predicates as well as objects and do not have a first order theory.

If we allow queries which quantify over types of predicates we may get different results from hierarchical models which reduce the types of predicates. For example, if the model is simplified by not considering temperature then it might be in equilibrium whereas a more detailed model may not because temperature is not constant.

If structural consolidation or any similar technique which reduces the number of objects is used then ordinary first order quantification over objects might produce different results in different models. A question such as "What is the warmest temperature in this house?" might have various answers depending on how detailed our model of a house is. Do we model at the room level? Do we model in terms of components of rooms such as the water heater in the utility room? Do we model the water heater in terms of components including the flame of a gas heating element? Do we model this flame in terms of its constituent parts?

(Some of which are hotter than others.)

Another problem with verification by simplification type is that it seriously limits the applicable simplifications. In fact, it appears to rule out most domain independent simplifications. ¿From the thought experiment point of view, this verification technique is useful only in special cases.

5.3 Heuristic Verification

The third method of verification is heuristic. Heuristic verification consists of trying other simplifications and comparing the descriptions for consistency. The details of this are tied to the simplification originally used. This is enormously flexible. We can use virtually anything for a simplification. Simplification does not have to be as complicated a process. Implicitly meta level questions about such things as equilibrium can be asked and answered. Explanation of reasoning would be clear to humans who deal with analogies quite well. Of course, we pay for the flexibility in that the conclusion is no longer certain. Heuristic verification is analyzed in terms of a probability model elsewhere[8].

6 Conclusions

We have presented some of the current work on the thought experiment methodology. We have included currently implemented simplifications as well as proposed new methods. A unifying framework has been presented which allows a comparison of the strengths and weaknesses of this methodology with other techniques for multi-level reasoning. This includes a classification of simplifications and a comparison and analysis of verification methods.

References

[1] S. Addanki, S. Cremonini, and, J.S. Penberthy. Reasoning about Assumptions in Graphs of Models. *Proc. Eleventh IJCAI,* Detroit, MI, pp. 1432-1438, August 1989.

[2] J. Collins and K.D. Forbus. Reasoning about Fluids via Molecular Collections. *Proc. AAAI-87*, Seattle, WA, pp. 590-595, 1987.

[3] B. Falkenhainer and K.D. Forbus. Setting up Large Scale Qualitative Models. *Proc. AAAI-88*, St. Paul, MN, pp. 301–306, 1988.

[4] B. Falkenhainer and K.D. Forbus. Compositional Modeling: Finding the Right Model for the Job. *Artificial Intelligence*, vol. 51, pp. 95–143, 1991.

[5] K.D. Forbus. Qualitative Process Theory. *Artificial Intelligence*, vol. 24, pp. 85–168, 1984.

[6] D.L. Hibler and G. Biswas. TEPS: The Thought Experiment Approach to Qualitative Physics Problem solving. to appear in, *Recent Advances in Qualitative Physics*, B. Faltings and P. Struss, eds., MIT Press, Cambridge, MA, 1992.

[7] D.L. Hibler and G. Biswas. The Thought Experiment Approach to Qualitative Physics. *Proc. Eleventh IJCAI*, Detroit, MI, pp. 1018-1026, 1989.

[8] D. Hibler The Thought Experiment Method, A New Approach to Qualitative Reasoning Ph.D. Dissertation, Univ. of South Carolina, Columbia, S.C., 1992.

[9] G. James and R. James (Eds.). *Mathematics Dictionary*, D. Van Nostrand Co., 1967.

[10] N. Kaul, G. Biswas, and B. Bhuva. Multi-level Qualitative Reasoning applied to CMOS Digital Circuits. to appear, *Intl. Journal of AI in Engineering*, 1992.

[11] B. Kuipers. Abstraction by Time-Scale in Qualitative Simulation. *Proceedings AAAI-87*, Seattle, WA, pp. 621-625, 1987.

[12] R. Korf. Planning as Search: A Quantitative Approach. *Artificial Intelligence*, vol. 33. pp. 65-88, 1987.

[13] Z.Y. Liu and A.M. Farley. Shifting Ontological Perspectives in Reasoning about Physical Systems. *Proc. AAAI-90*, Boston, MA, pp. 395-400, 1990.

[14] I. Prigogine and I. Stengers, *Order Out of Chaos*, Bantam Books, p.43, 1984.

[15] J. Tenenberg. Inheritance in Automated Planning *Proc. of the International Conference on Knowledge Representation*, pp. 475-485, 1989.

[16] D.S. Weld. The Use of Aggregation in Causal Simulation. *Artificial Intelligence*, vol. 30, pp. 1-34, 1986.

[17] D.S. Weld. Comparative analysis. *Artificial Intelligence*, vol. 36, pp. 333–374, 1988.

[18] D.S. Weld. Exaggeration *Proc. of the Seventh National Conference on Artificial Intelligence*, Minneapolis, MN, pp. 291-295, 1988.

[19] D.S. Weld and S. Addanki. Task-Driven Model Abstraction. *Proc. Fourth Intl. Workshop on Qualitative Physics*, Lugano, Switzerland, pp. 16-30, 1990.

[20] B. Williams. Doing time: putting qualitative reasoning on firmer ground. In: *Proc. AAAI '86*, Philadelphia, PA, pp. 105-112, 1986.