BReLS: A System for Revising, Updating, and Merging Knowledge Bases

Paolo Liberatore
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198, Roma, Italy
Email: liberato@dis.uniroma1.it

Abstract

Three distinct operations have been introduced so far to the extent of amalgamating knowledge coming from different sources: revision, merging, and update. In this paper we present a framework in which it is possible to perform these three operations at the same time. We show some examples in which this is actually necessary, and thus cannot be dealt by existent systems. The framework has been implemented, and the examples shown in this paper (as well as other examples from the belief revision literature) have been successfully tested. To our knowledge, this is the first implemented system able to perform revision, merging, and update at the same time.

1 Introduction

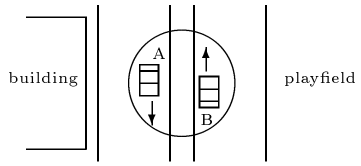
Since the introduction of the belief revision theory by Alchourrón, Gärdenfors, and Makinson, many semantics have been defined. During the last ten years, it has become clear that there are three distinct operations that can be performed for amalgamating knowledge coming from different sources. Namely, we have revision, that is the merging of pieces of information with different priority; merging, that is the incorporation of knowledge bases with the same priority; and update, that deals with knowledge about different time points.

The aim of this paper is to show that revision, merging, and update can be put together in a single framework. Our claim is that it makes sense to build a system able to perform all those operations *at the same time*. For instance, this allows for merging two knowledge bases having the same priority with another base having a greater priority. Updates allow for dealing with information about time, and they make therefore possible to deal with information sources referring to different time points.

Let us consider some examples that can be dealt with in such framework. We begin with an example in which it is necessary to perform revision and merging at the same time.

Example 1 *Let us consider the following scenario: we have a system to control speed and direction of a car. In order to decide what is the current position of the car we have four sources of information: the estimated position of the car using some satellite system; the information coming from sonar sensors mounted on the car (they can percept the presence of buildings along the road); the vision system, which can perform the same operation; and finally, we have the map of the city.*

Suppose that the satellite indicates the position of the car to be inside the circle in the following picture:



As a result, the car may be either in the position marked A, or in the position marked B. In logical terms, this can be represented as $\mathtt{pos}_A \not\equiv \mathtt{pos}_B$*, i.e. the car is either in position A or in position B (but not both). From the map we know that if the car is in position A, there should be a building on its right side, and if it is in position B, there should be a building on its left side. As a result, we have* $\mathtt{pos}_A \rightarrow \neg\mathtt{building\_left} \wedge \mathtt{building\_right}$*, and* $\mathtt{pos}_B \rightarrow \mathtt{building\_left} \wedge \neg\mathtt{building\_right}$*.*

The sensors say that there is a building on the left, and nothing on the right. This piece of information can be represented as $\mathtt{building\_left} \wedge \neg\mathtt{building\_right}$*. The information coming from the vision system is that there are buildings both on the left and on the right. In propositional terms, this can be represented as* $\mathtt{building\_left} \wedge \mathtt{building\_right}$*.*

In this example, there is a pair of sources of information which are fully reliable, that is, the position as given by the satellite and the map of the city. In this case, these pieces of information are consistent to each other, but they are not complete. What we know is that the car is in one of two possible positions. Since we want the exact position of the car, we should use the information coming from the two other sources, which are less reliable.

The problem can be solved by observing that, if the car is in position A, we have to assume that both the sensors and the vision system are faulty. On the other hand, the car being in position B means that only the vision system is wrong. As a result, this second possibility should be preferred.

In this example, we have both revision and merging, as there are a set of sources of information, and some of them have the same degree of reliability. Note that in more complex scenarios, there may be more than two sources all having a different degree of reliability (e.g. three sources each having a degree of reliability different from the other ones).

Let now consider an example in which we need a combination of revision and update.

Example 2 *We consider a client-server architecture. A client computer sends requests of service to the server; the server receives them and executes some kind of computation, and immediately sends a message back to the client informing it that the request has been processed. The server may not execute the requests either because it is too busy (and then the requests are delayed) or it is crashed (in this case, requests are never executed).*

Suppose that, after sending a service request, the client receives no answer. After some time, it assumes that the server is crashed, and thus the request has not been processed. Expressing such a knowledge in the propositional language, we have, at time 1, the following pieces of knowledge: $\neg$`service_executed` *and* `crashed`*.*

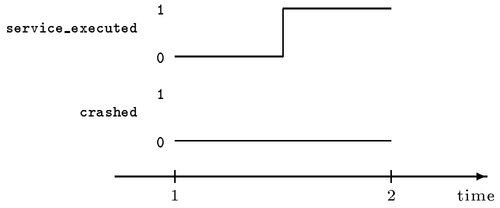
Eventually, the client receives an answer from the server, stating that the service has been executed: this also implies that the server is not crashed. Now, let us consider how reliable our knowledge is:

$\neg$`service_executed` at time 1. *Since the client did not receive any acknowledgment of execution, we can believe that the server did not process the request (this is because the server sends the answer immediately after processing the request and we assume that the server does not crash after the execution but before sending the answer).*

`crashed` at time 1. *This is an assumption made by the client. Indeed, if the answer is late from the server, the client first believe that the server is busy, and only after some time it starts to think that it crashed.*

`service_executed` $\wedge$ $\neg$`crashed` at time 2. *We have received the answer, thus the service has been executed, and the server is not crashed. This piece of information is fully reliable.*

Now, we have also some assumptions on change: a server cannot restart working after a crash. It is quite clear what happened in this scenario. At time 1 the server was not crashed: it was simply forced to delay the processing of the request. The processing was executed between time 1 and 2.



This is a scenario in which both revision and update are necessary. Indeed, the initial knowledge (that is, the knowledge at time 1) was that the server was crashed and the service was not executed. We have a revision at time 1, because we discover that the server was not crashed, after all. We also have an update, since the execution of the request happened between time 1 and 2.

The following example shows the usefulness of having merging and update in the same framework.

Example 3 *Examples involving employees and departments are very common in the database area. Here we consider a scenario in which there are more than one source of information, and time is also involved.*

We have two departments of the same company: administration and marketing. An employee can work in marketing only if he shows good communication skills.

Bill is an employee of this company. When he was hired, he showed poor communication skills, and was thus assigned to administration. This can be represented, in logical terms, as `administration` $\wedge$ $\neg$`marketing` $\wedge$ $\neg$`communication`*. After two months, a new data is inserted in the database of the firm: Bill is moved to marketing. This can be represented as* $\neg$`administration` $\wedge$ `marketing` $\wedge$ `communication`*, since good communication skills are implied by the fact that Bill works in marketing.*

The state of the database can be represented as follows.

time=1	`administration` $\wedge$ $\neg$`marketing` $\wedge$ $\neg$`communication`
time=2	$\neg$`administration` $\wedge$ `marketing` $\wedge$ `communication`

Now, we can assume that a. the skills of Bill did not change in two months, and b. the two sources of information have the same reliability. Namely, if Bill took another test on communication skills, the result is not more believable than the result of the first one.

In such a situation, we have a change (Bill moved from administration to marketing), but also a merging: indeed, we have two sources of information, one stating that Bill is good in communicating, and one saying he is not. Since these sources have the same reliability, this is a case of merging. In this example, update and merging should be done at the same time. There are two solutions of this problem: in both of them we have `administration` $\wedge$ $\neg$`marketing` *at the beginning and* $\neg$`administration` $\wedge$ `marketing` *after two months. However, in the first solution* `communication` *is true in both time points, while in the second one it is false in both time points. As a result, we can prove that Bill worked in administration at time 1, while there is not enough information to find out whether he has good communication skills.*

In the following example (inspired by the Stolen Car Example [Friedman and Halpern, 1994]) we have to deal with information about more than two time points.

Example 4 *In this example, we consider a small part of the database used in a shop of electric appliances. The database contains information about the items in the shop. We consider only two items: a radio and a vacuum cleaner, and consider the part of the database containing information about which items are currently in the shop.*

The clerks of the shop use the following procedure: each time an item is sold, they insert in the database the fact that the item is not in the shop any more. At the end of each week, an inventory of the items in the shop is done. This makes it possible to discover whether items have been stolen.

When the first inventory is done, at time 1, both the radio and the cleaner are in the shop. At time 3, the vacuum cleaner is sold. When the second inventory is done at time 5, neither the radio nor the cleaner are in the shop any more.

Clearly, the radio has been stolen between the time points 1 and 5. "Classical" update cannot deal well with this problem: indeed, if we just iteratively apply Winslett's update, we obtain that the radio has been stolen immediately before the second inventory. The intuitive result, instead, is that the cleaner has been took out of the the shop at time 3, while the radio "disappeared" at some time point between 1 and 5.

The following example shows why it is useful to have revision involving more than two sources of information having different degree of reliability.

Example 5 *In order to solve complex problems, such as the satisfiability of propositional formulas, different kinds of programs can be used. Some of them are both correct and complete, some are only correct. Suppose that we want to solve the problem of satisfiability using a distributed system. Namely, we have three formulas A, B, and C, and two computers. We use the first computer to run the correct and complete algorithm, while the second one runs the approximate program. The first program finds out that the formula A is satisfiable, but it does not give the solution for formulas B and C in time. The approximate algorithm says that A is unsatisfiable, B is satisfiable, and does not terminate in time on the third one. Finally, we use a very rough algorithm to get an hint on the satisfiability on all the three formulas, such as checking whether the number of clauses is less than* 4.3 *times the number of variables. With this heuristics, we find that the first formula and the third one are satisfiable, while the second one is not.*

As a result, we have three sources of information, with different degrees of reliability.

Reliable: formula A is satisfiable.

Less reliable: A is not satisfiable, while B is.

Rough estimation: formulas A and C are satisfiable, while B is not.

The reliability of these three sources is different. The first one is completely sure, while the second one is only an approximation, and the third one is just an estimation. In this case, the integration of these bases should give that the three formulas are satisfiable: indeed, A is satisfiable because the first algorithm always gives the correct answer. As a result, we cannot fully believe in the second source: however, the second algorithm gives that B is satisfiable, and since the only source in contradiction is still less reliable, we believe it. At this point, we do not have any information about C, and thus we can believe the rough estimation, that is, C is also satisfiable.

A real introduction of belief revision is out of the scope of this paper. We assume the reader to be familiar with the basic concepts of the field, namely revision, iterated revision, arbitration, merging, and the difference between revision and update.

2 Syntax

In this section we show how knowledge is syntactically expressed in BReLS. Let us forget time, and consider knowledge bases referring to a single time point. As shown in Example 1, different pieces of information can have different degrees of reliability. In BReLS, we write:

$$\texttt{source}(i) : K$$

In words, we mean that the piece of knowledge K has reliability i. Example 1 can be encoded as follows. The information given by the satellite system is that either the car is in position A or it is in position B, thus the piece of information provided by this source is $\texttt{pos}_A \not\equiv \texttt{pos}_B$. The information inferred from the map of the city is expressed by the following formula:

$$(\texttt{pos}_A \rightarrow \neg\texttt{building\_left} \wedge \texttt{building\_right}) \wedge$$
$$(\texttt{pos}_B \rightarrow \texttt{building\_left} \wedge \neg\texttt{building\_right})$$

The information provided by the sensors is $\texttt{building\_left} \wedge \neg\texttt{building\_right}$, while the vision system gives $\texttt{building\_left} \wedge \texttt{building\_right}$. Our assumption is that the satellite system and the map always provide fully reliable information, while the sensor and the vision systems may be wrong. This can be formalized by assigning an higher degree of reliability to the information provided by the satellite and inferred from the map. As a result, we have:

$$\begin{array}{ll}
\texttt{source}(2): & \texttt{pos}_A \not\equiv \texttt{pos}_B \\
\texttt{source}(2): & \\
\quad (\texttt{pos}_A \rightarrow \neg\texttt{building_left} \wedge \texttt{building_right}) & \\
\quad \wedge(\texttt{pos}_B \rightarrow \texttt{building_left} \wedge \neg\texttt{building_right}) & \\
\texttt{source}(1): & \texttt{building_left} \wedge \neg\texttt{building_right} \\
\texttt{source}(1): & \texttt{building_left} \wedge \texttt{building_right}
\end{array}$$

In BReLS, information having degree of reliability i is always considered more reliable than any number of pieces of information with reliability $j < i$. In other words, a source saying that a variable is true is always believed, even if there are hundreds of less reliable sources saying that the variable is false.

Let now introduce time in our framework. Syntactically, it is easy to say that a piece of information refers to a specific time point. Borrowing Sandewall's syntax [Sandewall, 1994], we write time in square brackets:

$$\texttt{source}(i) : [t]K$$

This formula means that the information K is believed to hold at time t with a degree of reliability i. Of course, the fact that K is true at time t does not imply that K is also true at time $t+1$, $t+2$, etc. The assumption made in belief update is that scenarios are static "by default", that is, formulas continue to hold until the contrary is perceived. For instance, in Example 2, the piece of information $\neg$`service_executed`, holding at time 1, would be also believed to hold at time 2. However, at time 2 a new piece of information saying that `service_executed` holds invalidates this default assumption.

The fact that scenarios may be modified is formalized using the idea of *changes*. A change is what in the reasoning about actions field is known as action, and in some belief revision works as events. Essentially, a change is anything that modifies the world of interest. The default assumption is that changes do not happen. However, we can explicitly say that a change may happen with a given "penalty". For example we may have:

$$\begin{array}{ll} \texttt{change}(3): & X \\ \texttt{change}(2): & Y \end{array}$$

This means that changes X and Y may happen, but we are more likely to accept that Y took place, rather than X. In BReLS, changes are expressed as literals: `change`(i): l means that l may become true, and the likeliness of this is given by i (note: the greater the value of i, the unlikely the change is).

Let us consider Example 2 again. The knowledge about time points can be simply expressed as:

$$\begin{array}{ll} \texttt{source}(2): & [1]\neg\texttt{service\_executed} \\ \texttt{source}(1): & [1]\texttt{server\_crashed} \\ \texttt{source}(2): & [2]\texttt{service\_executed} \wedge \neg\texttt{server\_crashed} \end{array}$$

Indeed, the pieces of knowledge with reliability 2 are those expressing things known for certain. On the other hand, [1]`server_crashed` is just an assumption made, thus it is less reliable. Now, any change may happen, except that a server cannot recover from a crash. As a result, we give penalty 1 to any change except that in which the server becomes non-crashed.

```
change(1)
change(4):     server_crashed
```

The first statement says that all changes have penalty 1 (where not otherwise specified), thus they may happen. The second statement says that the variable `server_crashed` cannot switch from false to true. By default, the statement `change(1)` can be omitted, that is, every change whose penalty is not given is assumed to have penalty 1. A penalty 0 for a change means that there is no problem in assuming that the change happened, that is, we do not make the default assumption that the change does not happen.

3 Semantics

In this section we first introduce the notion of dynamic model, and then we give some basic properties a semantics should have. The next section gives the semantics chosen for the system BReLS (the linear semantics).

3.1 Models

Suppose the time points we are dealing with are $\{1, \ldots, T\}$, and the propositional variables are $X = \{x_1, \ldots, x_n\}$.

Definition 1 *A* static model *is a truth assignment to the variables in X.*

A static model represents the state of the world in a given time point. In order to express the state and the changes in all considered time points, we introduce the notion of dynamic model.

Definition 2 *A* dynamic model *is composed of two parts:*

1. *a set of static models, one for each time point;*
2. *a sequence of changes for each pair of consecutive time points.*

This is a very general definition, without any explicit definition of what changes are. In BReLS, the only changes considered are those setting the value of a variable. Changes are represented by literals: for instance the change that makes a false is represented by $\neg a$. It is easy to think of formalizations of changes much more complex than the ones considered in this work.

3.2 Consistency and Entailment

First of all, we have to define consistency of dynamic models. Given two time points t and $t+1$, a dynamic model contains a static model M associated to time t, a static model N associated with time $t+1$, and a sequence of changes associated with the pair $(t, t+1)$.

In order for the dynamic model to be consistent, the model N should be the result of applying the changes to the model M. This must holds for any time point t. In our system, the result of a sequence of changes applied to a static model M is always a single model – however, this may not be the case if changes are more complex.

Of course, not all possible models are equally plausible. Some models may be more likely than others. The specific choice of ordering in BReLS is outlined in the next section. For now, we assume that there is an ordering between dynamic models.

Given a knowledge base, its models are the consistent dynamic models that are minimal with respect to the ordering. A knowledge base is consistent if and only if it has at least one model (in BReLS, any knowledge base is consistent).

A knowledge base implies a formula $[t]F$ if, in any model of the knowledge base, F is true at time t. In other words, we have to take any model of the knowledge base, extract the static model associated to the time point t, and check whether F is satisfied by the static model.

4 The Linear System

In this section, we explain the choice of the semantics made in BReLS, that is, the linear semantics. The ordering between models is determined from the preference of models, which is in turns obtained from the priority of formulas and the penalty of changes.

4.1 Preference of Static Models.

We begin with the case in which all formulas refer to the same time point. In this case, a dynamic model is composed of one static model only. We have therefore to define a suitable ordering between static models.

Let M be a static model, and F a propositional formulas. As for instance in [Dalal, 1988] we define the distance between M and F as:

$$\text{dist}(M, F) = \min(\{\text{hamm}(M, N) | N \in \text{mod}(F)\}, \leq)$$

where $\text{hamm}(M, N)$ is the Hamming distance between the models M and N (the number of atoms to which they assign a different truth value). In words: the distance between a static model M and a propositional formula F is the minimal Hamming distance between M and any model of F.

Let us consider a knowledge base composed of two formulas with the same priority and referring to the same time point:

$$\begin{array}{ll} \texttt{source(1)}: & K_1 \\ \texttt{source(1)}: & K_2 \end{array}$$

Since we prefer models that are as close as possible to the formulas in the knowledge base, we define the preference of a static model M as:

$$\text{pref}(M) = \text{dist}(M, K_1) + \text{dist}(M, K_2)$$

The models of a knowledge base are those with the least value of the preference function, thus the models in which the sum of the distances is minimal.

Let now consider a knowledge base in which there is another formula with a greater priority, for instance:

$$\begin{array}{ll} \texttt{source(1)}: & K_1 \\ \texttt{source(1)}: & K_2 \\ \texttt{source(2)}: & K_3 \end{array}$$

Now, it is not possible to define the preference of a model just as a sum of distances. The reason is that, since K_3 is the most prioritized formula, all minimal models should satisfy it. As a result, we define the preference as an array having one element for each level of priority in the knowledge base. In our example, there are formulas with priority 1 and one formula with priority 2, thus we need an array with two elements. The array is defined as:

$$\text{pref}(M)[i] = \sum_{F \text{ has priority } i} \text{dist}(M, F)$$

In the specific example above, the array has two elements, whose value is:

$$\begin{array}{rcl} \text{pref}(M)[1] & = & \text{dist}(M, K_1) + \text{dist}(M, K_2) \\ \text{pref}(M)[2] & = & \text{dist}(M, K_3) \end{array}$$

The ordering between two models M_1 and M_2 is defined as follows, where P is the maximal level of priority of formulas in the knowledge base.

$$\begin{array}{lll} M_1 \prec M_2 & \text{iff} & \text{pref}(M_1)[P] < \text{pref}(M_2)[P] \text{ OR} \\ & & \big(\text{pref}(M_1)[P] = \text{pref}(M_2)[P] \text{ AND} \\ & & \text{pref}(M_1)[P-1] < \text{pref}(M_2)[P-1]\big) \text{ OR} \\ & & \vdots \end{array}$$

This definition ensures that if the distance of M_1 to the formulas with the maximal priority is less than the distance of M_2, then M_1 is preferred, regardless of the formulas having lower priority. The distance to formulas with priority $P - 1$ only matters when the distance to formulas at priority P is the same for both models.

As a result, if there is only one formula with the greatest priority, then all minimal models of the knowledge base have static models satisfying that formula (if consistent). This is what is expressed by the AGM postulate of success: if a formula with the greatest priority is consistent, then all models of the revised knowledge base must satisfy it. On the other hand, our ordering satisfies the principle that if a formula with a lower priority is consistent with the ones with greatest priority, the result of revision is the logical conjunction of them.

One can easily verify that the models of $\{\texttt{source(1)}: K_1,\ \texttt{source(2)}: K_2\}$ are the models of the result of Dalal's revision $K_1 * K_2$, which satisfies all AGM postulates.

4.2 Preference between Dynamic Models

So far, we have defined the array of priorities of a static model with respect to a set of formulas referring to the same time point. We now consider knowledge bases with time. In order to define the preference between dynamic models, we have to take into account two facts: 1. there are formulas at different time points, and 2. there are penalties associated with changes.

Let KB be a knowledge base, and let $KB(t, p)$ be the set of its formulas with time t and priority p. Let $DM(t)$ be the static model associated with time t in the dynamic model DM. The preference array of a dynamic model DM is defined as follows.

$$\text{pref}(DM)[p] = DM_p + \sum_{t=1}^{T} \sum_{F \in KB(t,p)} \text{dist}(DM(t), F)$$

where DM_p is the number of changes with penalty p in the model DM. The definition of $\prec$ is exactly as in the case of static models. When all formulas are fully reliable (that is, when their degree of priority is greater than those of changes), this definition can be rephrased as: we consider the minimal set of changes needed to explain the observations. This is similar to the principle of "actions to explain observation", which is studied in reasoning about actions [Li and Pereira, 1996; Baral *et al.*, 1996; McIlraith, 1998].

There are two possible ways for defining the models of a knowledge base. We call these two possible semantics *backward* and *pointwise*. The first one is the simplest to explain, and allows for expressing scenarios in which knowledge at later time may rule out some initial models. The second one is based on the principle of "pointwiseness": each initial model should be updated separately (this is one of the basic principles of update according to Katsuno and Mendelzon [1991]).

Backward Semantics. This is the simplest one. We define the ordering $\prec$ between dynamic models from the arrays of preference, as we did for static models. Then, the models of a knowledge base are just defined as the minimal dynamic models.

$$\mathrm{mod}(KB) = \min(\{DM\}, \prec)$$

Pointwise Semantics. This is the case in which update should be pointwise, that is, there should be a different set of minimal (dynamic) models for each minimal initial (static) model. The set of models of a knowledge base is defined as follows:

$$\mathrm{mod}(KB) = \bigcup_{M \text{ minimal initial model}} \min(\{DM \mid DM(1) = M\}, \prec)$$

This way, for each minimal initial (static) model M, we take the set of minimal dynamic models such that $DM(1) = M$.

We can prove that some semantics previously defined for revision and update can be implemented in BReLS. Namely, Dalal's revision and Forbus' update can be easily translated into BReLS.

Theorem 1 *It holds $K * P \models Q$ if and only if $[1]Q$ is implied by* $\{\mathtt{source}(1)[1]K,\ \mathtt{source}(2)[1]P\}$ *(using either the pointwise or the backwards semantics), where $*$ is Dalal's update.*

Theorem 2 *It holds $K \circ P \models Q$ if and only if $[2]Q$ is implied by* $\{\mathtt{source}(2)[1]K,\ \mathtt{source}(2)[2]P\}$*, using the pointwise semantics, where $\circ$ is Forbus' update.*

Many examples introduced in the literature can also be expressed in BReLS. For instance, the litmus test example [Boutilier, 1995] can be expressed using the backwards semantics.

5 Implementation

In the previous sections we defined the semantics of our framework. Let now consider the implementation. There are two things we want to do with a knowledge base. The first is to find all of its (dynamic) models. The second is to check which formulas are implied. Our implementation is a simple C program, that reads both the knowledge base and the queries from standard input. For instance, the knowledge base of Example 2 is represented as

```
source(2): [1]-service_executed
source(1): [1]crashed
source(2): [2]service_executed&-crashed
change(4): -crashed
backward
```

We use `&` for the logical conjunction, `|` for the disjunction, and `-` for negation.

The output is the set of models of this knowledge base. If we want for instance to investigate whether the variable `crashed` is `true` at time 1, we add to the input of the program the following statement.

```
query: [1] crashed
```

Queries without the time argument are true if and only if they are true in all time points.

From a theoretical point of view, we can prove that checking whether a query is implied by a knowledge base is Δ_2^p-complete for the backward semantics, and Π_2^p-complete for the pointwise semantics.

We implemented two algorithms: the first is an exhaustive algorithm, and it is only able to deal with very small problems; the second one is based on a procedure for finding the minimal models of a propositional formula. The first algorithm is mainly for testing (due to its simplicity it is less likely to be faulty), and is better than the second one only on very small instances.

5.1 Exhaustive search

This is the default algorithm. There is a preliminary phase in which the preference of each static model is determined, for each time point: this is done by determining, for each static model M and each time point t, the preference array of M with respect to the formulas holding at time t. The result of this phase is exponentially large, which is a little weird, since the considered problems are in PSPACE. Since we want to solve small instance only, we bounded the number of variables to eight: in this case, there are only at most $2^8 = 256$ static models, and we can afford time and space to build the table.

In the second phase, for the pointwise semantics, all minimal static models for time 1 are considered, and for each of them, the minimal dynamic models are found. For the backward semantics, we just find the minimal dynamic models. The table of preferences of static models is used to determine the penalty of dynamic models more quickly.

This algorithm can process knowledge bases with up to 5 variables and 4 time points in a few seconds, or 8 variables and 1 time point. Consider that state-of-art propositional theorem provers can deal with more than 200 variables. We do not include figures here, but is clear that the running time of this algorithm get worst as the number of variables and time points increases.

5.2 Minimal models search

There are two ways for making BReLS actually able to deal with scalable inputs. The first is to convert the exhaustive search algorithm into a real backtracking algorithm. In other words, we should not try to generate all possible complete models, but rather building partial models and doing backtracking each time the preference of a model is more than the last optimum found.

If we want to deal with scalable problems, we cannot use the preliminary phase in which we determined the preference of static models, as the generated table may become too large. However, we believe that some compilation may turn out to be useful.

The second way for speeding up the solving phase, and the one that has been actually implemented, is to solve the problem by calling a procedure for finding minimal models. This algorithm is invoked by adding the statement `mindp` at the end of the knowledge base.

For the backward semantics, we convert all formulas in the base into a single propositional formula, and a set of weights, each associated to a variable. Then, we run a procedure that finds the models with the minimal weight. The algorithm is described in more details in [Liberatore, 1999]: basically, it is a variant of the algorithm for testing the satisfiability of sets of prositional clauses developed during the 60's by Davis, Putnam, Logemann, and Loveland. The procedure returns the weight of the minimal model. Then, we add the negation of the queries once at time, and check whether the weight of the minimal models is the same of the previous call. The query is satisfied if and only if the weight is different.

The pointwise semantics has been implemented in a less efficient way: we first use the procedure for finding all the minimal initial models. For each of them we run the previous algorithm with a fixed initial state.

The algorithm for the pointwise semantics is not optimal, as it always require a call to the procedure for finding the minimal weight for each static model at time 1, and the number of these models may be exponentially large. A more efficient solution is to convert the knowledge base into a logic program, or a default theory, and then running a specialized system for solving such problems. Since the problem of inference for the pointwise semantics is in Π_2^p, this conversion is feasible in polynomial time. Such a solution has not been implemented yet.

5.3 Running BReLS

The system can be run locally by compiling everything and then running the program over some input files. There is also a CGI interface that allows the system to be run over the Internet. The URL is:

```
http://www.dis.uniroma1.it/~liberato/brels
```

We describe the interface (which is indeed quite simple): there is a web form composed of a textarea and two buttons "Run BReLS" and "Reset". The user inserts a knowledge base in the textarea, and then presses the "Run BReLS" button. If the input contains no query, the set of the models of the knowledge base is printed. If there are some queries, the system returns whether each query is implied from the knowledge base. The models can be printed, even if there are some queries, by inserting the command `printmodels` in the input. Note that, if the algorithm used is `mindp`, a model may be printed more than once.

6 Examples

In this section we show how the examples presented in this paper can be encoded in BReLS. Example 1 and Example 2 have already been discussed, therefore we consider only the remaining ones.

6.1 Example 3

The knowledge about time is represented as:

```
source(2): [1]administration&-marketing&
             -communication
source(2): [2]-administration&marketing&
             communication
```

We assumed that these sources have the same reliability. There are also some knowledge about time: first, skills do not change; and second, department may change:

```
change(1)

change(5): communication
change(5): -communication
```

This way, the variable `communication` cannot change, while the other two ones can. Since the initial state may be modified, we use the backward semantics, thus we add the statement

```
backward
```

In this case, we have two models. In both of them, `administration&-marketing` is true at time 1 and `-administration&marketing` is true at time 2. In the first model, `communication` is true in both time points, while in the second model, it is false at any time.

6.2 Example 4

Since both the radio and the cleaner are in the shop at time 1, while they are not at time 5, the first fragment of the knowledge base is as follows.

```
source(3): [1]radio_in_shop&cleaner_in_shop
source(3): [5]-radio_in_shop&-cleaner_in_shop
```

A problem is "how to express that the cleaner has been sold at time 3?" We just say that there is no penalty in the change of `cleaner_in_shop` between time 2 and 3, and that `cleaner_in_shop` is false at time 3. This way, we can actually implement the standard semantics update [Winslett, 1990]. A statement like `change(0)` : $[t]l$ corresponds to a release of the truth value of the literal l.

```
change(0): [2]-cleaner_in_shop
source(3): [3]-cleaner_in_shop
```

The result is composed of four models. In all of them, `cleaner_in_shop` is true at time points 1 and 2, and false in the other ones. Each model differs from the other ones for the time point in which the variable `radio_in_shop` change from true to false.

6.3 Example 5

Since the first theorem prover is reliable, it has the greatest degree of priority. The approximate theorem prover is less reliable, and the estimate is still less reliable. As a result, we have the following knowledge base.

```
source(3): first_consistent
source(2): -first_consistent&second_consistent
source(1): first_consistent&-second_consistent
           &third_consistent
```

This knowledge base has only one model, namely, the one in which all variables are true.

7 Conclusions

In this paper we have shown how revision, merging and update can be implemented in a system able to perform all of them on the same knowledge base. We have shown some examples in which this is necessary.

We began with considering knowledge about a single time point, where only revision and merging are needed. When we added time, the necessity of updates arises. We defined a very general notion of dynamic model, something we think that many may agree: a static model for each time point and a set of changes for each consecutive pair of time points.

The way we define the semantics of a knowledge base is by considering only the minimal models. The ordering of models in BReLS is fixed, and this is for sure one of the points on which the reader may disagree. Indeed, one can easily produce examples that cannot be correctly dealt by BReLS. For instance, while we can express a problem similar to the Stolen Car Example [Friedman and Halpern, 1994], we cannot express the problem itself, because there is no way to state that the mileage of a car can increase only if the car is outside the parking lot. More work is needed to fully understand how the semantics can be extended to deal with such scenarios.

Further work is also needed in the analysis of more efficient algorithms. Indeed, in its current implementation, BReLS can deal only with small problems. For this reason, if we want to really use it in real-world domains, efficient algorithms should be developed and implemented.

Acknowledgements

Most of the work reported in this paper has been done while the author was visiting Linköping University. He thanks Erik Sandewall for his hospitality and support. Many thanks to Marco Schaerf for revising an earlier version of this paper.

References

[Baral *et al.*, 1996] C. Baral, M. Gelfond, and A. Provetti. Representing actions: Laws, observations and hypothesis. *J. of Logic Programming*, 1996.

[Boutilier, 1995] C. Boutilier. Generalized update: belief change in dynamic settings. In *Proc. of IJCAI'95*, pages 1550–1556, 1995.

[Dalal, 1988] M. Dalal. Investigations into a theory of knowledge base revision: Preliminary report. In *Proc. of AAAI'88*, pages 475–479, 1988.

[Friedman and Halpern, 1994] N. Friedman and J. Y. Halpern. A knowledge-based framework for belief change: Part II: Revision and update. In *Proc. of KR'94*, pages 190–200, 1994.

[Katsuno and Mendelzon, 1991] H. Katsuno and A. O. Mendelzon. On the difference between updating a knowledge base and revising it. In *Proc. of KR'91*, pages 387–394, 1991.

[Konieczny and Pino Perez, 1998] S. Konieczny and R. Pino Perez. On the logic of merging. In *Proc. of KR'98*, pages 488–498, 1998.

[Li and Pereira, 1996] R. Li and L. Pereira. What is believed is what is explained. In *Proc. of AAAI'96*, pages 550–555, 1996.

[Liberatore and Schaerf, 1998] P. Liberatore and M. Schaerf. Arbitration (or how to merge knowledge bases). *IEEE Trans. on Knowledge and Data Engineering*, 10(1):76–90, 1998.

[Liberatore, 1999] P. Liberatore. Algorithms and experiments on finding minimal models. Technical Report 09–99, Dip. di Inf. e Sist., Univ. di Roma "La Sapienza", 1999.

[Lin, 1996] J. Lin. Integration of Weighted Knowledge Bases. *Artificial Intelligence*, 83(2):363–378, 1996.

[McIlraith, 1998] S. McIlraith. Explanatory diagnosis: Conjecturing actions to explain observations. In *Proc. of KR'98*, pages 167–179, 1998.

[Revesz, 1993] P. Z. Revesz. On the semantics of theory change: Arbitration between old and new information. In *Proc. of PODS'93*, pages 71–82, 1993.

[Sandewall, 1994] E. Sandewall. *Features and Fluents.* Oxford University Press, 1994.

[Williams, 1994] M. Williams. Transmutations of knowledge systems. In *Proc. of KR'94*, pages 619–629, 1994.

[Winslett, 1990] M. Winslett. *Updating Logical Databases.* Cambridge University Press, 1990.