A practical approach to hypothetical database queries

Henning Christiansen and Troels Andreasen

Roskilde University, Computer Science Dept.,
P.O.Box 260, DK-4000 Roskilde, Denmark
E-mail: {troels, henning}@ruc.dk

Abstract. Hypothetical queries are queries embedding hypotheses about the database. The embedded hypothesis in a hypothetical query indicates, so to say, a state of the database intended for the rest of the query. Thus the answer to a hypothetical query $h > q$, with a hypothesis h, is in principle the result of evaluating q against the database revised with h. In case h is inconsistent with the database, query evaluation becomes a special case of counterfactual reasoning. However, the possible worlds semantics usually applied for this notion is not relevant for database applications due to reasons of inefficiency. In this paper we discuss and compare different approaches to hypothetical queries, paying special attention to potentials for efficient evaluation. As a central part of the paper we present and discuss our own approach "counterfactual exceptions", which have the important property of, as opposed to the other approaches discussed, requiring only minor overhead in query evaluation. This approach is thus realistic for practical implementation and use in environments supporting large databases. The "price" for efficient evaluation is an altered semantics, as compared to the other approaches. However, it can be argued that this semantics is at least as appropriate for database applications as that of the other approaches mentioned.

1 Introduction

The use of hypotheses is common in natural languages and is highly important in many areas. In legal reasoning, for example, statements such as the following often show up, "If the suspect had not been guilty, she would have ...". Linguists, philosophers and others have been interested in hypothetical expressions and special attention have been payed to counterfactual reasoning in order to assign a meaning to such statements and to be able to check the validity of counterfactual arguments. Hypotheses are used, so to say, to modify reality as a set-up for an expression, for instance by supposing hypothetically the fact that the suspect is not guilty.

While premises of implications in classical logic in some cases may play the role of hypotheses, they are not interesting for counterfacts as any conclusion follows from a false premise. The most common approach to counterfactual reasoning is based on possible worlds models as introduced by Lewis in [17] and

discussed by Ginsberg in [13]. The semantics for a counterfactual expression in the possible worlds' models is: A conclusion follows from a counterfactual premise if it holds in all worlds, satisfiyng the premise and being "sufficiently close" to the real world.

Our main interest in this paper is counterfactual statements posed as queries to potentially large databases and we consider the additional requirements that this implies for a formalization and efficient implementation of the concept. Firstly, we observe that it is undesirable to update a database in order to evaluate a hypothetical query. Support of multiple users, simultaneously posing normal queries will then require locking and roll-back functionality as needed for updating requests. Also from the viewpoint of efficiency, updating the database, perhaps to several possible states, during evaluation of a query, is problematic as databases in general must be assumed to contain large amounts of data.

For knowledge base applications, assuming a small size and complex knowledge base, it may be reasonable to adapt a model-checking approach to counterfactual reasoning as that of Grahne and Mendelzon in [14]. For database applications with a large size base and with efficient query-answering as an ultimate feature, it is unrealistic even in optimized implementations to consider all close possible worlds, cf. the complexity results of [3, 4].

In order to seriously include hypothetical statements in a database query language, we find it essential that the total amount of time spent on the evaluation of a hypothetical query is comparable to that of a conventional query. This makes it irrelevant to suggest a semantics which fully complies with a possible worlds semantics.

In this paper, we compare different approaches to counterfactual reasoning, including our own proposal for a query evaluation mechanism intended for databases [1]. We consider here only a single new "world" which adapts the hypothesis and we do not need to change the database at all, instead we modify the query evaluation procedure so it behaves *as if* the relevant changes had been made.

One important issue in our discussion concerns to what extent the result of a hypothetical query should reflect the actual structure of the database. The structure of the database is the means by which the knowledge in the database is represented in facts, rules and integrity constraints, thus the question is to what extend a semantics should go beyond a purely model-theoretical interpretation of the database. We argue that hypothetical querying and counterfactual reasoning in general, in many cases should exploit more than just the model-theoretical consequences of the database. The actual database structure, especially the database rules representing the "why's", are also of concern, when assimilating hypothesis. To distinguish on this matter we introduce a logical property called *structure independence*, indicating that the structure is ignored. Our mechanism takes into account more database structure than other approaches to counterfactual reasoning, in a way which seems intuitively acceptable for database applications.

We consider databases consisting of positive rules and facts together with integrity constraints that limit the possible updates. We define a deductive database as a finite set of clauses of the form

$$H \leftarrow A_1 \wedge \ldots \wedge A_n,$$

where H is either an atom or the special symbol $\bot$ and $A_1, \ldots, A_n$ are atoms. If $n > 0$ and H an atom, the clause is a *rule*; if $n > 0$ and $H = \bot$, an *integrity constraint*; if $n = 0$ and H an atom, a *fact*; the case with $n = 0$ and $H = \bot$ is not allowed. Atoms are first-order without function symbols. We refer to the usual notions of Herbrand universe, least Herbrand model, and logical consequence. A database is inconsistent if it has $\bot$ as a logical consequence, and in this case we consider any formula as a logical consequence of the database.

In order to accept something as a database, we assume also a *query evaluation mechanism* (for short, an *evaluator*) which is a device capable of determining whether or not a member of a certain class of formulas is a logical consequence of a given instance of the database. One example of such a mechanism is SLD resolution [15] which is complete for conjunctive queries to consistent databases.

2 Asserting hypotheses in database queries

We will use the notation $\phi > \psi$ for the statement "if ϕ were the case, would ψ be the case" in the context of some database DB; ϕ is called a *hypothesis*, ψ the *(hypothetical) conclusion* and the whole construct a *hypothetical query*. Our main concern in the present paper is to consider the possible truth values of $\phi > \psi$ and to consider how these values can be evaluated when $\phi > \psi$ is posed as a query to DB. For simplicity, we assume ψ to be an atom, and we will consider the impact of different choices for ϕ.

In the discussion, we will refer to a sample database of traveling information including the following clauses.

$travel(X, Y) \leftarrow link(X, Y)$
$travel(X, Y) \leftarrow link(X, Z) \wedge travel(Z, Y)$
$link(X, Y) \leftarrow flight(X, Y)$
$link(X, Y) \leftarrow bus(X, Y)$
$link(X, Y) \leftarrow \ldots$
$flighttravel(X, Y) \leftarrow flight(X, Y)$
$flighttravel(X, Y) \leftarrow flight(X, Z) \wedge flighttravel(Z, Y)$
$flight(copenhagen, brussels)$
$\ldots$

2.1 Positive hypotheses

For planning purposes, for example, it can be relevant to investigate the effects of a possible change in the world by means of a hypothetical query involving a new fact. A query like

$$flight(leuven, brussels) > flighttravel(leuven, mallorca),$$

if answered positively, could be an argument for setting up a new flight connection from Leuven to Brussels.

In the simplified case where we have no integrity constraints and a hypothetical premise consisting of only positive clauses, no potential inconsistency can arise from adding the hypothetical premise to the database. To obtain a logical semantics for $>$, we can observe that $>$ naturally coincides with conventional implication by considering the classical deduction theorem (see, e.g., [7]),

$$\Gamma \models \phi \rightarrow \psi \quad \text{iff} \quad \Gamma \cup \{\phi\} \models \psi.$$

$$\Gamma \models \phi \rightarrow \psi \quad \text{iff} \quad \Gamma \cup \{\phi\} \models \psi.$$

Evaluation of a query $\phi > \psi$ is performed by extending the database with ϕ and evaluating ψ with evaluation method at hand for queries not embedding hypotheses.[1] This principle has been used in the design of several programming languages, we can refer to [9, 20, 11, 12, 21, 19, 5].

Also in this simplified case, with a semantics as indicated, we can observe a nice, logical property that we shall call *structure independence.*

> Whenever DB_1 and DB_2 are logically equivalent databases,
> $DB_1 \models \phi > \psi$ iff $DB_2 \models \phi > \psi$.

(UDDYB LOGICAL EQUIVALENT) In other words, the structure independence means that the structure of the database do not affect the meaning of hypothetical queries.[2] So here, all queries are answered identically when posed to the structurally different, but logically equivalent, databases $\{a, b\}$ and $\{a \leftarrow b, b\}$.

In an attempt to generalize the use of positive hypotheses to databases with integrity constraints, we have to consider the possible inconsistency. Consider the example above assuming an integrity constraint

$$\bot \leftarrow noairport(X) \wedge flight(X, Y)$$

and a fact $noairport(leuven)$. In this case, the hypothetical premise

$$flight(leuven, brussels)$$

[1] It is not necessary actually to add the hypothesis to the database, as an alternative the query evaluation method can easily be modified to behave as if the hypothesis were present in the database.

[2] A little digression about the least Herbrand model semantics is relevant here. The two databases $\{\,\}$ and $\{a \leftarrow b\}$ have identical least Herbrand model, the empty one. However, they are not logically equivalent and the query $b > a$ gives different results when posed to these two databases. So, although the least Herbrand model semantics is perfectly suited for reasoning about positive databases with atomic queries, it should not be trusted in the context of hypothetical reasoning, not even with positive and atomic hypotheses.

added in a direct way to the database will make the database inconsistent, and a semantics based on the deduction theorem as above will make any conclusion hold. In order to assign an interesting meaning to such hypothetical queries, we should consider (at least conceptually) a revised and consistent database, e.g., including a removal of the fact $noairport(leuven)$.

The reference [8] describes a programming language with integrity constraints and hypothetical implication goals with atomic hypotheses, that deals with this kind of revision of the database. The inconsistency problem is solved by assigning a priority to the facts such that an "older" fact that leads to inconsistency should be ignored; in the example above, $flight(leuven, brussels)$ given as a "new" hypothesis will take precedence over $noairport(leuven)$, which, thus, is ignored in the evaluation process.

In this approach, the evaluator provides an illusion of an updated database such that no restructuring needs to take place. However, the evaluator described by [8] appears to be far too slow for database application: Each time a database fact, say a, is used in a proof, consistency needs to be checked by an exhaustive search, using negation-as-failure, to ensure that $\bot$ cannot be proved with a in the database. In other words, hypothetical queries may execute orders of magnitudes slower that ordinary ones.[3]

In order to discuss the structure independence property, we must go a little into detail with the language of [8]. Rules and integrity constraints belong to a *protected part* and facts may appear freely in the protected or another so-called *removable part*. A fact in the removable part can only be used in a proof if it does not cause inconsistency with the database and possible hypotheses in the query. Facts in or derived from clauses in the protected part can always be used in a proof, no test for inconsistency is made. This means that a change in the database structure that moves things between the protected and removable parts, but does not change the overall logical consequence, can change the outcome of a hypothetical query. Consider, e.g., a database with protected part $\{a \leftarrow b,\ \bot \leftarrow b \wedge c\}$ and removable part $\{b\}$; here $c > a$ fails. Deleting $a \leftarrow b$ from the protected part and adding a to the removable part will make $c > a$ succeed.

Structure independence seems to hold when restricting to changes within the protected part. If we decide to place all database clauses (including facts) in the protected part, the semantics becomes rather uninteresting: The consistency check is never activated, which means that the integrity constraints are ignored and, thus, whether or not the hypothesis is inconsistent with the database has no influence at all.

2.2 Negative hypotheses

Negative hypotheses can be relevant for investigating the consequences of discarding, say, a given flight connection. It could, for example, be interesting to

[3] There are several other problems with the proposed operational semantics, as also pointed out by [8], including floundering and loops in the consistency check.

investigate possible connections from Copenhagen assuming that the flight between Copenhagen and Brussels were closed. Negative hypothesis can also specify that we want to disregard a certain part of the database, which somehow is undesirable for a given purpose. A person terrified of flying might want to pose a query such as $\neg flight(x, y) > travel(copenhagen, sidney)$ with the intended meaning to ask whether there exists a travel between the two destinations involving only means of transportation, that does not include flights. As these examples indicate, negative hypotheses that may be of interest to embed in queries to a database tend to be inconsistent with this database.

Semantic optimization and hypothetical reasoning may appear to be notions with quite different impact. The former leading to an increase in evaluation performance and the latter to a more complicated evaluation involving complex expressions. However, as the example above indicates, negative hypotheses may reduce the search space (i.e., the portion of the database that needs to be considered) and if such hypotheses can be derived from integrity constraints for a given query and if the evaluation of hypothetical queries is only minimally slower than for normal queries, the net result can be a speed up. This is the case for the mechanism, we present in section 4.2 below.

Treatment of negative hypotheses is the issue in the reminder of this paper. In section 3 we review firstly the work on counterfactual reasoning, which does not seem to satisfy our needs. In section 4 we introduce a pragmatic "by exception" treatment of negative hypotheses. We discuss and compare a possible generalization of the work of [8] to handle negative hypotheses and our own approach, treating negative hypothesis specifically as exceptions. Apart from our own work, we are not aware of any treatment of this topic which is relevant for database applications. In section 5 we compare to the possible-worlds semantics.

3 Counterfactual reasoning

Reasoning including hypothesis that are potentially inconsistent with our knowledge about the real world is quite common in daily life as well as in sciences and trades. Examples are manifold and well-known and the phenomenon has attracted much attention in a philosophical context. Classical first-order logic is not equipped for reasoning under inconsistency, despite the fact that we all have a clear understanding of some counterfactual arguments being more valid than others.

Lewis' [17] studies of the phenomenon are central in this area. He describes a semantics of counterfactual implication based on possible worlds models. A counterfactual statement $\phi > \psi$ holds whenever ψ holds in all worlds where ϕ holds, that are sufficiently close to the actual world. The notion of being "sufficiently close" is based on an accessibility relation among worlds which is left open, so to speak, as a parameter that confines a given semantics for counterfactual implications.

For any application of counterfactual reasoning in a computerized context, we need to decide upon an accessibility relation. Or the other way round, any

claimed implementation of counterfactual reasoning somehow induces an accessibility relation.[4] Ginsberg [13] gives a formal treatment of counterfactual reasoning in the case of predicate logic. Ginsberg's construction does not satisfy the structure independence property as his accessibility relation is based on maximally consistent subsets of formulas, i.e., a query $\phi > \psi$ holds in DB whenever ψ holds in $DB' \cup \{\phi\}$ for all maximal subsets $DB' \subseteq DB$ with $DB' \not\models \neg\phi$.[5] As an example, the query $\neg b > a$ succeeds in $\{a, b\}$ but fails in $\{a \leftarrow b, b\}$ (and similarly for $\neg a > b$, Ginsberg's [13] example). Ginsberg discusses this in depth and concludes that there is no special reason for requiring the property we have called structure independence. (DET FLGENDE M UDDYBES) We will actually go a step further, claiming that the database structure expresses an *intention* that is abstracted away in the model-theoretical considerations, but is significant in the context of counterfactual reasoning. We can illustrate what we mean by a comparison of the two databases considered above. The clause $a \leftarrow b$ indicates that if a holds, then this is *because of* b. This dependency is lost when going to the model-theoretically equivalent database $\{a, b\}$. In our view, this sort of dependency is highly relevant when changing hypotheses, e.g., by a removal of b.

Gärdenfors [10] characterizes counterfactual implication by means of revision of belief sets, i.e., $\phi > \psi$ holds if the conclusion ψ holds in a new belief set revised by the hypothesis ϕ. By definition, belief sets are closed under logical consequence so this view of counterfactual implication satisfies *a priori* the structure independence property and thus disagrees with [13] and our own view as discussed above.

Efficiency

It seems inherent in the nature of counterfactual reasoning based on possible worlds that the only way to evaluate a hypothetical query $\phi > \psi$ is to construct a representation of the required class of possible worlds in which ϕ holds and then check the validity of ψ in each of them. Each such world can be represented as an updated database or as an explicit model, or the set of all these worlds can be represented as a common structure by means of disjunctions. The common structure approach is taken in [13] which we discussed above and in the formulation in a logical programming context by [22] based on an extension of well-founded model semantics. Important results on the complexity of hypothetical queries can be found in [3, 4]. We will however not go into detail on this matter because it refers to evaluations involving multible models or worlds, while our main focus is a practical approach, described in section 4.2 that involves only one model.

[4] The distinction made by [8] between protected and removable program parts can be seen as a way for the programmer to adjust the accessibility relation otherwise determined by the interpreter.

[5] Ginsberg's construction can be extended by a refinement of the subset ordering and an explicit condition to rule out "bad worlds". However, the characterization given here is sufficient for our points.

4 Negative hypothesis considered as exceptions

We consider the special case with hypotheses being negated atoms as discussed in section 2 above, and we will reconsider the following query

$$\neg flight(x, y) > travel(copenhagen, sidney).$$

In principle, we could represent the hypothesis as an integrity constraint

$$\bot \leftarrow flight(x, y)$$

and consider all possible ways to achieve a consistent database using the principle of [13] outlined in section 3 above and then check the conclusion

$$travel(copenhagen, sidney)$$

in each of the resulting databases. We consider below two other alternatives.

4.1 Generalizing an existing evaluation method

We can easily generalize the evaluation method of [8] to handle counterfactual hypothesis given as integrity constraints. For each hypothesis h that *potentially* may appear negated in a hypothetical query

$$\neg h > \ldots.$$

add to the protected part of the database an integrity constraint of the following form,

$$\bot \leftarrow h \wedge \text{active}_k,$$

where k uniquely determines the given integrity constraint, and active_k is not part of the database. If, e.g., the one excluding flights above carries number 17, we can pose the original query as $\text{active}_{17} > travel(copenhagen, sidney)$.

However, as explained earlier the evaluator obtained in this way will be too inefficient for all but trivial databases, and we will compare with our own mechanism, which is considerably more efficient than any of the approaches discussed earlier.

4.2 A practically relevant "one-world" approximation

The basic idea is to avoid the construction of multitudes of worlds and instead consider a single world and check validity of the conclusion in it. Furthermore, the world should be of such a nature that we can evaluate the conclusion efficiently in it (so it is not acceptable to eliminate alternative worlds or databases by the introduction of disjunctions).

A thorough introduction to our construction and a series of motivation examples is given [1]. (OMFORMULER DA FLERE EKSEMPLER MEDTAGES HERI) We use a special implication arrow $\rightarrow\!\!\!\!\!\rightarrow$ to distinguish our specific operator. We consider a restricted form of counterfactual implications which are closed formulas of the form

$$\exists \cdots (\phi \twoheadrightarrow \psi)$$

with

$$\phi = (\forall \cdots \neg\phi_1) \wedge \cdots \wedge (\forall \cdots \neg\phi_n)$$

where $\phi_1, \ldots, \phi_n$ are atoms, ψ a conjunction of atoms; each subformula $\forall \cdots \neg\phi_i$ is called a *counterfactual exception*. Any variable quantified at the outermost level is said to be *global*, all other variables in the ϕ_i's are *local*. For simplicity we begin by considering the case without global variables. This simplification implies that ψ will be ground but we relax this requirement later.

As logical semantics, we can use the following generalization of the traditional fixpoint semantics for logical programs (see [18]). Given a database DB, we will recognize a formula $\phi \twoheadrightarrow \psi$ as true if and only if $\psi \in M^{\phi}_{DB}$ where M^{ϕ}_{DB} is *least model* for DB under the exceptions ϕ defined as follows,

$$M^{\phi}_{DB} = \mathrm{lfp}(T^{\phi}_{DB})$$

that is, as the least fixed point of the function T^{ϕ}_{DB}, where T^{ϕ}_{DB} is the following generalized *consequence operator*.

$$\begin{aligned} T^{\phi}_{DB}(I) = \{\alpha \,|\, & DB \text{ has a clause with a ground instance} \\ & \alpha \leftarrow \beta_1 \wedge \cdots \wedge \beta_k \\ & \text{with } \beta_i \in I \text{ for all } i \text{ and } \phi \wedge \alpha \text{ is consistent}\} \end{aligned}$$

In other words, we allow those immediate consequences of clauses in the database that do not conflict with the exceptions.

As an example, for the database

$$DB_0 = \{p(X) \leftarrow q(X),\ p(a),\ q(b)\}$$

we have the following,

$$\begin{aligned} & M^{true}_{DB_0} = \{p(a), q(b), p(b)\}, \\ & M^{\neg p(b)}_{DB_0} = \{p(a), q(b)\}, \\ & M^{\forall Y \neg q(Y)}_{DB_0} = \{p(a)\}. \end{aligned}$$

Suppose we have the following instance of the travel database

$$\begin{aligned} \{ & travel(X,Y) \leftarrow link(X,Y), \\ & travel(X,Y) \leftarrow link(X,Z) \wedge travel(Z,Y), \\ & link(X,Y) \leftarrow train(X,Y), \\ & link(X,Y) \leftarrow boat(X,Y), \\ & link(X,Y) \leftarrow flight(X,Y), \\ & flight(a,b),\ flight(b,c),\ flight(d,e),\ flight(e,a), \\ & train(a,b),\ train(c,d),\ boat(b,c)\ \} \end{aligned}$$

To this database we can pose the query "I want to travel from a to d, but I refuse to sail from b to c", as

$$(\neg boat(b,c)) \rightarrow\!\!\!\!\rightarrow travel(a,d)$$

Within the shown instance the query obviously succeeds since

$$\{flight(a,b), flight(b,c), train(c,d)\} \subseteq M_{DB}^{\neg boat(b,c)}.$$

The query "I want to travel from a to d, but I refuse to fly",

$$(\forall X,Y \neg flight(X,Y)) \rightarrow\!\!\!\!\rightarrow travel(a,d)$$

also succeeds since

$$\begin{aligned} &travel(a,d) \in M_{DB}^{\forall X,Y \neg flight(X,Y)} = \\ &\quad \{train(a,b), train(c,d), boat(b,c), \\ &\quad\ link(a,b), link(c,d), link(b,c), \\ &\quad\ travel(a,b), travel(c,d), travel(b,c), \\ &\quad\ travel(a,c), travel(a,d), travel(b,d)\ \}. \end{aligned}$$

The following expresses "I want to travel from a to d, but I refuse to sail into the harbor of c".

$$(\forall X \neg boat(X,c)) \rightarrow\!\!\!\!\rightarrow travel(a,d)$$

The semantics for hypothetical queries with global variables can be defined by expressing the existential quantification at the meta-level as follows.

> The formula $\exists X_1 \cdots X_n\, (\phi \rightarrow\!\!\!\!\rightarrow \psi)$ follows from a database DB whenever $\phi \rightarrow\!\!\!\!\rightarrow \psi$ has an instance $\phi' \rightarrow\!\!\!\!\rightarrow \psi'$ with $\psi' \in M_{DB}^{\phi'}$.

We can show the use of global variables in the query "I want to travel from a to a place where I do not arrive by train".

$$\exists X((\forall Y \neg train(Y,X)) \rightarrow\!\!\!\!\rightarrow travel(a,X))$$

We should stress that counterfactual exceptions also may concern information which is not represented as facts in the database but implied from other facts. The following example may be relevant if you had all your luggage stolen in c on your last travel. "I want to travel from a to e, but I refuse to pass by c",

$$((\forall X \neg link(X,c)) \wedge (\forall X \neg link(c,X))) \rightarrow\!\!\!\!\rightarrow travel(a,e).$$

Having a careful look at the semantic definition, we observe that it is forbidden to apply any $link(\_,\_)$ via c in the evaluation of $travel(a,e)$ but it is still possible to use, say, $fligth(b,c)$ for other purposes than "linking" our traveler.

The semantics obtained as illustrated above and explained in more detail in [1] is equivalent to checking the conclusion in a single, revised database in which each clause is extended with a "filter" to prevent it from producing conclusions that conflict with the exceptions. To express this, we extend the database formalism with a constraint interpreted as syntactic inequality. The exception $\neg p(b)$ leads to the following modification of DB_0.

$$DB_0' = \{p(X) \leftarrow X \neq b \wedge q(X),\ p(a),\ q(b)\}$$

In [1] this is formalized as a generalized completion construction in the sense of [6].

4.3 An evaluator for the "one-world" approximation

As we have put forward several times, it is essential to avoid actually constructing a new database in order to answer a hypothetical query. Instead, the evaluator will be equipped in such a way that it behaves *as if* it was executing in the revised database. We can characterize an evaluator for our construction by means of the following, modified Vanilla interpreter.

```
% prove( φ ->> ψ ) if and only if φ ⇝ ψ
prove(_ ->> true):- !.
prove(Cf ->> (A,B)):-
    !, prove(Cf ->> A), prove(Cf ->> B).
prove(Cf, A) :-
    clause(A,B),
    consistent(A, Cf),
    prove(Cf ->> B).
```

The `consistent` condition means that the selected atom `A` must satisfy a condition of non-unifiability with each atom appearing negatively in the exceptions. To this end, we use a declarative `dif(-,-)` predicate as it is found in, e.g., Sicstus Prolog [23]. A call `dif(`s`,`t`)` will delay a test for syntactic inequality until the moment that s and t are sufficiently instantiated to tell them either identical or non-unifiable. Each exception gives rise to a condition derived in the following way (a straightforward call to `dif` between two atoms is not sufficient). We have to distinguish between local and global variables and also take into account any possible sharing expressed by local variables. We analyze the arguments in each exception's atom in the following way.

- An argument which is a constant `c` or global variable `G` must always be different from the corresponding argument in the selected atom. This amounts to a test `dif(c,X)` or `dif(G,X)` where `X` refers to the corresponding argument in the selected atom `A`.
- A local variable occurring only once will always unify with the corresponding argument of the selected atom. This corresponds to always `fail`.
- A local variable which occurs as the ith as well as the jth argument implies a test `dif(`$\mathtt{X}_i$`,`$\mathtt{X}_j$`)` where $\mathtt{X}_i$ and $\mathtt{X}_j$ refer to the corresponding arguments of the selected atom.

Finally, these tests are merged together in a single call to `dif(-,-)` to express their disjunction.

Consider, as an example, the exception $\forall L_1 L_2 \neg p(a, G, L_1, L_2, L_2)$ where a is a constant and G a global variable. When `A` refers to the selected atom, the consistency test can be implemented by the following piece of Prolog code.

```
A = p(X1, X2, X3, X4, X5)
  -> dif((X1,X2,X4), (a,G,X5)) ; true
```

The syntax $\cdots$ `->` $\cdots$ `,` $\cdots$ denotes an if-then-else construction in Prolog.

In case of an "exhaustive" exception such as $\forall X, Y \neg flight(X, Y)$, the consistency test falls down to the following.

```
A = flight(_,_) -> fail ; true
```

The answers provided by this interpreter consist of bindings to variables together with a (perhaps empty) collection of unresolved `dif(-,-)` calls.[6] Concerned with completeness, it is easy to prove to following statement: "If it were the case that Prolog had used breadth-first search when choosing a clause, then this meta-interpreter would have been logically complete."

The `dif(-,-)` primitive is implemented by an efficient message passing method that does not slow down the Prolog engine, so our metainterpreter executes only a small constant factor slower than the straightforward Vanilla interpreter evaluating a conventional, positive query to an entirely positive database.

In fact, this evaluator embeds an inherent semantic optimization in the sense that the exceptions effectively reduce the search space as the consistency condition prevents the evaluator from considering the body of a clause which anyhow would not lead to new answers. In other words, the evaluation of $\phi \twoheadrightarrow \psi$ can be much faster than evaluation of ψ using a conventional evaluator. This implies that our notion of exceptions can be of interest for semantic optimizations even in conventional databases without hypothetical queries.

It should also be emphasized that these considerations about correctness and efficiency are valid for first-order clauses and queries with global variables as shown in this slightly speculative example,

$$\exists X((\forall Y \neg train(Y, X)) \twoheadrightarrow travel(a, X))$$

with the intuitive meaning "I want to travel from a to a place where I do not arrive by train". In addition, it seems possible to generalize the approach to logic programs with function symbols.

(HER SKAL INDFJES EKSEMPLER)

5 Exceptions vs. possible worlds counterfactuals

(MSKE BR ARGUMENTATIONEN HERUNDER DREJES S DET IKKE LSES SOM ET AD HOC APPROACH) The most remarkable difference between our approach and counterfactual reasoning in the sense of Lewis and Ginsberg [17, 13] is that of efficiency. The restriction to examining a single world together with an efficient evaluator makes our approach appropriate for practical database application, which is certainly not the case for any treatment of the possible worlds semantics that we are aware of.

The interesting question is, then, whether this single world (that we imitate, but actually do not construct) really represents an intuitively acceptable,

[6] If we assume infinitely many (or just sufficiently many) constant symbols, such an answer, a substitution plus a finite set of inequalities, always represents a nonempty set of ground answer substitutions.

archetypal representative for the worlds in which the given hypothesis hold — or at least an acceptable approximation thereof, considering the efficiency gained.

As we have argued in this paper, there is no reason to insist on the structure independence property, actually we consider it undesirable as it rules out important intentions embedded in database clauses.

As we have shown, the two model-theoretically equivalent programs $\{a \leftarrow b, b\}$ and $\{a, b\}$ are considered to be different in our approach as well as that of [13] with respect to counterfactual reasoning.

We can indicate a difference between our approach and [13] by another example. Consider the two databases

$$DB_1 = \{a \leftarrow b,\ b \leftarrow c,\ c\}$$

and

$$DB_2 = DB_1 \cup \{a \leftarrow c\}$$

With our approach, $\neg b \nrightarrow a$ fails in DB_1 and succeeds in DB_2. With the mechanism of Ginsberg [13] or our adaptation of [8], it fails in DB_1 as well as in DB_2.[7]

In other words, our approach respects the intentions embedded in the clause $a \leftarrow c$, stating that if we have c, we must also have a (and this indifferently of whether some other phenomenon such as b is present or not). We can say, that if someone decides to put the (model-theoretically redundant) clause $a \leftarrow c$ into the database, he should have some reason for doing so, namely to express such an intention. In the other, referenced works, this dependency is ignored or abstracted away, and whether this really is based on philosophical grounds or it is an inadvertent consequence of the technical definitions is difficult to say.

We can strengthen our point by instantiating the example above to something more intuitive.

$$DB_1' = \{happy \leftarrow cake, cake \leftarrow money, money\}$$

We have money, we spend it on the cake and we become happy. If there is no cake to buy, can we then be happy? The query $\neg cake \nrightarrow happy$ fails in ours as well as Ginsberg's approach expressing the intuitively correct conclusion, that there is no possibility of being happy without the cake. Consider, now, the following additional clause

$$happy \leftarrow money.$$

It states very clearly, that if we have money we are happy no matter whether we expect to buy something or not. Adding this clause to the database will make

[7] Proof: The database $DB_1 \setminus \{c\}$ is a maximal subset of DB_1 which is consistent with $\neg b$ and in which a does not hold, the same thing can be said about $DB_2 \setminus \{c\}$. Referring to the semantics of [8], the hypothesis $\neg b$ implies that c cannot be used in a proof because it leads to inconsistency due to the presence of $b \leftarrow c$ in DB_1 as well as in DB_2.

$\neg cake \rightarrow\!\!\!\!\rightarrow happy$ succeed in our approach, intuitively correct, the cake is gone, but we are happy due to our money. With Ginsberg's approach the query fails also in the extended database, but why should we be unhappy due to lack of cake when the database explicitly states that we can be happy with our money?

The approach of [8] is similar to ours in the sense that only a single world (i.e., one modified database) is considered when checking the hypothetical conclusion. Where Ginsberg [13] considers the collection of maximally consist subsets, the "world" considered by [8] can be understood as the intersection of all such maximal subsets (however, maximality here with respect to removal of facts only). If, e.g., consistency can be obtained by removing *either a or b*, [8] may loose some conclusions that are found by [13].

The single world considered in our approach can be described by a systematic change of the formulas in the database. In the propositional case, this world is equivalent to the maximally consistent subset in which any clause is removed whose application *immediately* would produce an inconsistency. Thus we ignore the possible effects of the counterfactual hypothesis implied "indirectly", e.g., by contraposition from the database clauses. For first-order clauses, our approach is somewhat more precise than deleting entire clauses, we so to speak disable only the fraction of each clause that can produce inconsistency in this immediate sense.

Our counterfactual exceptions are related to Kowalski and Sadri's notion of Logic Programming with Exceptions [16]. They consider exceptions to predicates as part of the program and not as hypotheses posed dynamically in the queries. On the other hand, they can define exceptions by arbitrary clauses which, thus, implies that answer sets are not unique, as it is the case in our approach. At the implementation level, our use of the `dif(-,-)` technology makes queries terminate in meaningful states in cases where the corresponding programs of [16] give up due to floundering. The flexibility in our approach to have common variables appearing in the hypotheses as well as in the conclusions has no counterpart in the language of [16] or any other language we are aware of.

In [1] we describe also a language where implications $\phi \rightarrow\!\!\!\!\rightarrow \psi$ can appear as goals in program clauses. This is analogous to the way negation-as-failure is used in Prolog in the sense that essentially metalinguistic statements are moved into the language and, thus, a model-based semantics has to make use of suitable modal operators.

The property we have called structure independence is analogous (but not identical) to the notion called "model-based semantics" by Winslett [24] in order to characterize approaches to the related problem of database update. "Model-based" is contrasted with "formula-based", which means that consistency is re-established by removal of formulas, analogous to [8, 13] discussed above.

Finally, we will mention that we came over this notion of counterfactual exceptions due to our interest in the area of flexible query answering systems [2], where it is often relevant to go beyond a strictly logical semantics in order to imitate some of the actions that we expect when querying a human domain expert.

References

1. Andreasen, T., Christiansen, H. Counterfactual exceptions in deductive database queries. *Proc. ECAI'96, 12th European Conference on Artificial Intelligence* pp. 340–344, 1996.
2. Andreasen, T., Christiansen, H., and Larsen, H.L., eds. *Flexible Query-Answering Systems*, Kluwer, to appear 1997.
3. Bonner, A.J. Intuitionistic Deductive Databases and the Polynomial Time Hierarchy. *Journal of Logic Programming (JLP)*, 33(1):1-47,1997.
4. Bonner, A.J. Hypothetical Datalog: Complexity and Expressibility. *Theoretical Computer Science (TCS)*, 76:3-51, 1990. (FYLDES UD)
5. Christiansen, H., A complete resolution method for logical meta-programming languages. *Lecture Notes in Computer Science* 649, pp. 205–219, 1992.
6. Clark, K.L., Negation as failure. *Logic and Data Bases*, Gallaire, H., and Minker, J. (eds.), Plenum Press, pp. 293–322, 1978.
7. Enderton, H.B., *A Mathematical Introduction to Logic*. Academic Press, 1972.
8. Gabbay, D.M., Giordano, L., Martelli, A., and Olivetti, N.,. Hypothetical updates, priority and inconsistency in a logic programming language. *Lecture Notes in Computer Science (LN in Artificial Intelligence)* 928, Springer-Verlag, pp. 203–216, 1995.
9. Gabbay, D.M. and Reyle, U., N-Prolog: An extension of Prolog with hypothetical implications. *Journal of Logic Programming* 2, pp. 319–355, 1984.
10. Gärdenfors, P., *knowledge in the Flux: Modeling the Dynamics of Epistemic States*, MIT Press, 1988.
11. Giordano, L. and Martelli, A., A modal reconstruction of blocks and modules in logic programming. *International Logic Programming Symposium*, 1991.
12. Giordano, L., Martelli, A., and Rossi, G., Extending Horn clause logic with implication goals. *Theoretical Computer Science*, 1991.
13. Ginsberg, L.M., Counterfactuals. *Artificial Intelligence* 30, pp. 35–79, 1986.
14. Grahne, G., Mendelzon, A., Updates and subjunctive queries. *Information and computation* 116(2), pp. 241–252, 1995.
15. Kowalski, R.A., Predicate logic as a programming language. *Information Processing* 74, pp. 569–574, 1974.
16. Kowalski, R.A., and Sadri, F., Logic programming with exceptions. *Proc. of Eighth International Conference on Logic Programming*, MIT Press, pp. 598–613, 1991.
17. Lewis, D, *Counterfactuals*. Harward University Press, 1973.
18. Lloyd, J.W., *Foundations of logic programming*, Second, extended edition. Springer-Verlag, 1987.
19. Miller, D., Lexical scoping as universal quantification, *Proc. of Sixth International Conference on Logic Programming*, MIT Press, pp. 268–283, 1989.
20. Monteiro, L. and Porto, A., Contextual Logic Programming, *Proc. of Sixth International Conference on Logic Programming*, MIT Press, pp. 284–302, 1989.
21. Nait Abdallah, M.A., Ions and local definitions in logic programming, *Lecture Notes in Computer Science* 210, pp. 60–72, Springer-Verlag, 1986.
22. Pereira, L.M., Aparício, J.N., and Alfares, J.J., Counterfactual reasoning based on revising assumptions. Logic Programming, Proceedings of the 1991 Internal Symposium, MIT Press 1991.
23. *SICStus Prolog user's manual*. Version 3 #5, SICS, Swedish Institute of Computer Science, 1996.
24. Winslett, M., Updating Logical Databases. *Cambridge Tracts in Theoretical Computer Science*, vol. 9, Cambridge University Press, 1990.